

Draft Standard for Information Technology— Portable Operating System Interface (POSIX®)

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

and

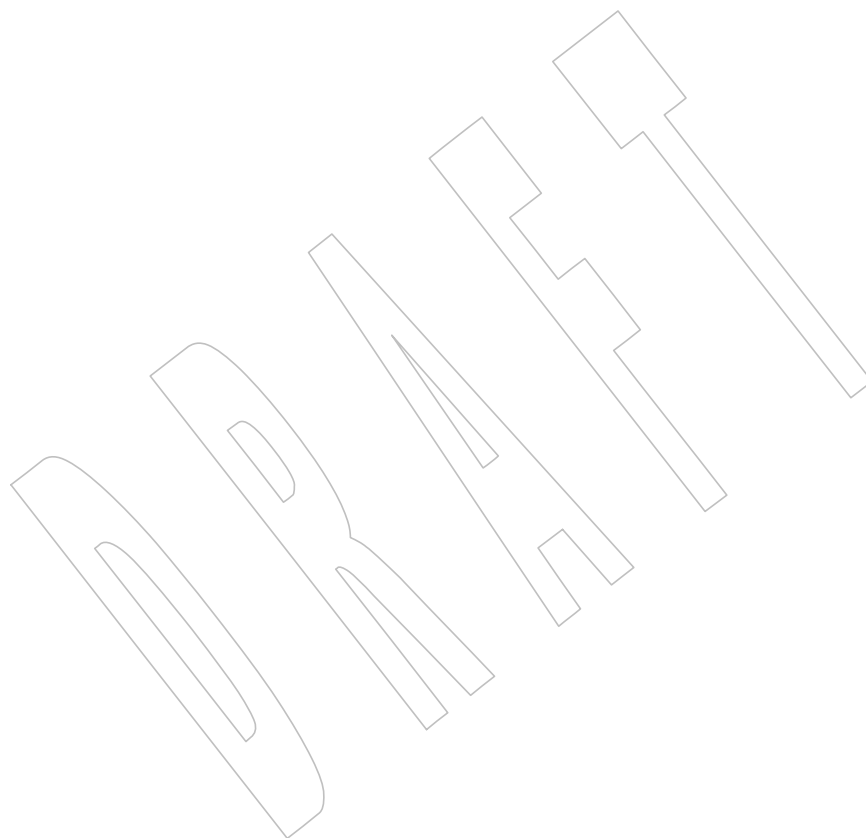
The Open Group

Copyright © 2007 The Institute of Electrical & Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2007 The Open Group
Thames Tower, Station Road, Reading, Berkshire RG1 1LX, UK
All rights reserved.

Except as permitted below, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the copyright owners. This is an unapproved draft, subject to change. Permission is hereby granted for Austin Group participants to reproduce this document for purposes of IEEE, The Open Group, and JTC1 standardization activities. Other entities seeking permission to reproduce this document for standardization purposes or other activities must contact the copyright owners for an appropriate license. Use of information contained within this unapproved draft is at your own risk.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.



Draft Standard for Information Technology— Portable Operating System Interface (POSIX®)

Draft Technical Standard: Base Specifications, Issue 7

Prepared by the Austin Group (www.opengroup.org/austin)

Copyright © 2007 The Institute of Electrical & Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2007 The Open Group
Thames Tower, Station Road, Reading, Berkshire RG1 1LX, UK

All rights reserved.

Except as permitted below, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the copyright owners. This is an unapproved draft, subject to change. Permission is hereby granted for Austin Group participants to reproduce this document for purposes of IEEE, The Open Group, and JTC1 standardization activities. Other entities seeking permission to reproduce this document for standardization purposes or other activities must contact the copyright owners for an appropriate license. Use of information contained within this unapproved draft is at your own risk.

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Abstract

POSIX.1-200x defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. POSIX.1-200x is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-200x and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of POSIX.1-200x:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-200x describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Keywords

application program interface (API), argument, asynchronous, basic regular expression (BRE), batch job, batch system, built-in utility, byte, child, command language interpreter, CPU, extended regular expression (ERE), FIFO, file access control mechanism, input/output (I/O), job control, network, portable operating system interface (POSIX[®]), parent, shell, stream, string, synchronous, system, thread, X/Open System Interface (XSI)

Feedback

POSIX.1-200x has been prepared by the Austin Group. Feedback relating to the material contained in POSIX.1-200x may be submitted using the Austin Group web site at www.opengroup.org/austin/bugreport.html.

Contents

1	Volume 1	Base Definitions, Issue 7	1
2	Chapter 1	Introduction	3
3	1.1	Scope	3
4	1.2	Conformance.....	4
5	1.3	Normative References	4
6	1.4	Change History	5
7	1.5	Terminology	5
8	1.6	Definitions and Concepts.....	6
9	1.7	Portability	6
10	1.7.1	Codes	7
11	1.7.2	Margin Code Notation	13
12	Chapter 2	Conformance	15
13	2.1	Implementation Conformance	15
14	2.1.1	Requirements	15
15	2.1.2	Documentation	15
16	2.1.3	POSIX Conformance.....	16
17	2.1.3.1	POSIX System Interfaces	16
18	2.1.3.2	POSIX Shell and Utilities	18
19	2.1.4	XSI Conformance	19
20	2.1.4.1	XSI System Interfaces	19
21	2.1.4.2	XSI Shell and Utilities Conformance.....	20
22	2.1.5	Option Groups.....	20
23	2.1.5.1	Subprofiling Considerations.....	20
24	2.1.5.2	XSI Option Groups.....	21
25	2.1.6	Options	25
26	2.1.6.1	System Interfaces.....	26
27	2.1.6.2	Shell and Utilities	26
28	2.2	Application Conformance.....	28
29	2.2.1	Strictly Conforming POSIX Application.....	28
30	2.2.2	Conforming POSIX Application	29
31	2.2.2.1	ISO/IEC Conforming POSIX Application.....	29
32	2.2.2.2	<National Body> Conforming POSIX Application.....	29
33	2.2.3	Conforming POSIX Application Using Extensions.....	29
34	2.2.4	Strictly Conforming XSI Application	30
35	2.2.5	Conforming XSI Application Using Extensions	30
36	2.3	Language-Dependent Services for the C Programming Language	30
37	2.4	Other Language-Related Specifications.....	31
38			
39	Chapter 3	Definitions	33
40	3.1	Abortive Release.....	33
41	3.2	Absolute Pathname.....	33
42	3.3	Access Mode	33

43	3.4	Additional File Access Control Mechanism.....	33
44	3.5	Address Space.....	33
45	3.6	Advisory Information.....	33
46	3.7	Affirmative Response	34
47	3.8	Alert	34
48	3.9	Alert Character (<alert>).....	34
49	3.10	Alias Name.....	34
50	3.11	Alignment	34
51	3.12	Alternate File Access Control Mechanism	34
52	3.13	Alternate Signal Stack.....	34
53	3.14	Ancillary Data.....	35
54	3.15	Angle Brackets.....	35
55	3.16	Application.....	35
56	3.17	Application Address.....	35
57	3.18	Application Program Interface (API)	35
58	3.19	Appropriate Privileges	35
59	3.20	Argument	36
60	3.21	Arm (a Timer)	36
61	3.22	Asterisk.....	36
62	3.23	Async-Cancel-Safe Function.....	36
63	3.24	Asynchronous Events.....	36
64	3.25	Asynchronous Input and Output	36
65	3.26	Async-Signal-Safe Function.....	36
66	3.27	Asynchronously-Generated Signal.....	37
67	3.28	Asynchronous I/O Completion.....	37
68	3.29	Asynchronous I/O Operation.....	37
69	3.30	Authentication.....	37
70	3.31	Authorization	37
71	3.32	Background Job.....	37
72	3.33	Background Process.....	37
73	3.34	Background Process Group (or Background Job).....	37
74	3.35	Backquote	38
75	3.36	Backslash	38
76	3.37	Backspace Character (<backspace>)	38
77	3.38	Barrier	38
78	3.39	Basename.....	38
79	3.40	Basic Regular Expression (BRE).....	38
80	3.41	Batch Access List	38
81	3.42	Batch Administrator	38
82	3.43	Batch Client.....	39
83	3.44	Batch Destination	39
84	3.45	Batch Destination Identifier.....	39
85	3.46	Batch Directive.....	39
86	3.47	Batch Job	39
87	3.48	Batch Job Attribute.....	40
88	3.49	Batch Job Identifier	40
89	3.50	Batch Job Name	40
90	3.51	Batch Job Owner.....	40
91	3.52	Batch Job Priority	40
92	3.53	Batch Job State	40
93	3.54	Batch Name Service	40
94	3.55	Batch Name Space.....	40

Contents

95	3.56	Batch Node.....	41
96	3.57	Batch Operator.....	41
97	3.58	Batch Queue.....	41
98	3.59	Batch Queue Attribute.....	41
99	3.60	Batch Queue Position.....	41
100	3.61	Batch Queue Priority.....	41
101	3.62	Batch Rerunability.....	41
102	3.63	Batch Restart.....	42
103	3.64	Batch Server.....	42
104	3.65	Batch Server Name.....	42
105	3.66	Batch Service.....	42
106	3.67	Batch Service Request.....	42
107	3.68	Batch Submission.....	42
108	3.69	Batch System.....	42
109	3.70	Batch Target User.....	43
110	3.71	Batch User.....	43
111	3.72	Bind.....	43
112	3.73	Blank Character (<blank>).....	43
113	3.74	Blank Line.....	43
114	3.75	Blocked Process (or Thread).....	43
115	3.76	Blocking.....	43
116	3.77	Block-Mode Terminal.....	43
117	3.78	Block Special File.....	44
118	3.79	Braces.....	44
119	3.80	Brackets.....	44
120	3.81	Broadcast.....	44
121	3.82	Built-In Utility (or Built-In).....	44
122	3.83	Byte.....	44
123	3.84	Byte Input/Output Functions.....	45
124	3.85	Carriage-Return Character (<carriage-return>).....	45
125	3.86	Character.....	45
126	3.87	Character Array.....	45
127	3.88	Character Class.....	45
128	3.89	Character Set.....	45
129	3.90	Character Special File.....	46
130	3.91	Character String.....	46
131	3.92	Child Process.....	46
132	3.93	Circumflex.....	46
133	3.94	Clock.....	46
134	3.95	Clock Jump.....	46
135	3.96	Clock Tick.....	46
136	3.97	Coded Character Set.....	46
137	3.98	Codeset.....	47
138	3.99	Collating Element.....	47
139	3.100	Collation.....	47
140	3.101	Collation Sequence.....	47
141	3.102	Column Position.....	47
142	3.103	Command.....	48
143	3.104	Command Language Interpreter.....	48
144	3.105	Composite Graphic Symbol.....	48
145	3.106	Condition Variable.....	48
146	3.107	Connected Socket.....	48

147	3.108	Connection	48
148	3.109	Connection Mode	48
149	3.110	Connectionless Mode	48
150	3.111	Control Character	49
151	3.112	Control Operator	49
152	3.113	Controlling Process	49
153	3.114	Controlling Terminal	49
154	3.115	Conversion Descriptor	49
155	3.116	Core File	49
156	3.117	CPU Time (Execution Time)	49
157	3.118	CPU-Time Clock	50
158	3.119	CPU-Time Timer	50
159	3.120	Current Job	50
160	3.121	Current Working Directory	50
161	3.122	Cursor Position	50
162	3.123	Datagram	50
163	3.124	Data Segment	50
164	3.125	Deferred Batch Service	50
165	3.126	Device	50
166	3.127	Device ID	50
167	3.128	Directory	51
168	3.129	Directory Entry (or Link)	51
169	3.130	Directory Stream	51
170	3.131	Disarm (a Timer)	51
171	3.132	Display	51
172	3.133	Display Line	51
173	3.134	Dollar Sign	51
174	3.135	Dot	52
175	3.136	Dot-Dot	52
176	3.137	Double-Quote	52
177	3.138	Downshifting	52
178	3.139	Driver	52
179	3.140	Effective Group ID	52
180	3.141	Effective User ID	52
181	3.142	Eight-Bit Transparency	53
182	3.143	Empty Directory	53
183	3.144	Empty Line	53
184	3.145	Empty String (or Null String)	53
185	3.146	Empty Wide-Character String	53
186	3.147	Encoding Rule	53
187	3.148	Entire Regular Expression	53
188	3.149	Epoch	53
189	3.150	Equivalence Class	54
190	3.151	Era	54
191	3.152	Event Management	54
192	3.153	Executable File	54
193	3.154	Execute	54
194	3.155	Execution Time	54
195	3.156	Execution Time Monitoring	54
196	3.157	Expand	55
197	3.158	Extended Regular Expression (ERE)	55
198	3.159	Extended Security Controls	55

Contents

199	3.160	Feature Test Macro	55
200	3.161	Field.....	55
201	3.162	FIFO Special File (or FIFO)	55
202	3.163	File	56
203	3.164	File Description	56
204	3.165	File Descriptor	56
205	3.166	File Group Class	56
206	3.167	File Mode.....	56
207	3.168	File Mode Bits	56
208	3.169	Filename	56
209	3.170	File Offset	57
210	3.171	File Other Class	57
211	3.172	File Owner Class	57
212	3.173	File Permission Bits.....	57
213	3.174	File Serial Number	57
214	3.175	File System	57
215	3.176	File Type	57
216	3.177	Filter	58
217	3.178	First Open (of a File)	58
218	3.179	Flow Control.....	58
219	3.180	Foreground Job.....	58
220	3.181	Foreground Process	58
221	3.182	Foreground Process Group (or Foreground Job).....	58
222	3.183	Foreground Process Group ID	58
223	3.184	Form-Feed Character (<form-feed>).....	59
224	3.185	Graphic Character	59
225	3.186	Group Database.....	59
226	3.187	Group ID.....	59
227	3.188	Group Name	59
228	3.189	Hard Limit.....	59
229	3.190	Hard Link	59
230	3.191	Home Directory	60
231	3.192	Host Byte Order.....	60
232	3.193	Incomplete Line.....	60
233	3.194	Inf.....	60
234	3.195	Instrumented Application	60
235	3.196	Interactive Shell.....	60
236	3.197	Internationalization	60
237	3.198	Interprocess Communication	60
238	3.199	Invoke	61
239	3.200	Job.....	61
240	3.201	Job Control	61
241	3.202	Job Control Job ID	61
242	3.203	Last Close (of a File).....	61
243	3.204	Line.....	61
244	3.205	Linger	61
245	3.206	Link	62
246	3.207	Link Count	62
247	3.208	Local Customs	62
248	3.209	Local Interprocess Communication (Local IPC).....	62
249	3.210	Locale	62
250	3.211	Localization.....	62

251	3.212	Login	62
252	3.213	Login Name	62
253	3.214	Map	62
254	3.215	Marked Message	63
255	3.216	Matched	63
256	3.217	Memory Mapped Files	63
257	3.218	Memory Object	63
258	3.219	Memory-Resident.....	63
259	3.220	Message	63
260	3.221	Message Catalog.....	64
261	3.222	Message Catalog Descriptor	64
262	3.223	Message Queue.....	64
263	3.224	Mode	64
264	3.225	Monotonic Clock	64
265	3.226	Mount Point	64
266	3.227	Multi-Character Collating Element	64
267	3.228	Mutex	64
268	3.229	Name.....	65
269	3.230	Named STREAM.....	65
270	3.231	NaN (Not a Number)	65
271	3.232	Native Language	65
272	3.233	Negative Response.....	65
273	3.234	Network.....	65
274	3.235	Network Address	65
275	3.236	Network Byte Order	66
276	3.237	Newline Character (<newline>)	66
277	3.238	Nice Value	66
278	3.239	Non-Blocking.....	66
279	3.240	Non-Spacing Characters	66
280	3.241	NUL.....	66
281	3.242	Null Byte.....	66
282	3.243	Null Pointer.....	67
283	3.244	Null String.....	67
284	3.245	Null Wide-Character Code	67
285	3.246	Number Sign.....	67
286	3.247	Object File.....	67
287	3.248	Octet	67
288	3.249	Offset Maximum	67
289	3.250	Opaque Address.....	67
290	3.251	Open File	67
291	3.252	Open File Description.....	67
292	3.253	Operand.....	68
293	3.254	Operator	68
294	3.255	Option	68
295	3.256	Option-Argument	68
296	3.257	Orientation	68
297	3.258	Orphaned Process Group.....	68
298	3.259	Page.....	69
299	3.260	Page Size.....	69
300	3.261	Parameter	69
301	3.262	Parent Directory	69
302	3.263	Parent Process.....	69

Contents

303	3.264	Parent Process ID	69
304	3.265	Pathname.....	70
305	3.266	Pathname Component.....	70
306	3.267	Path Prefix	70
307	3.268	Pattern.....	70
308	3.269	Period.....	70
309	3.270	Permissions	70
310	3.271	Persistence.....	70
311	3.272	Pipe.....	71
312	3.273	Polling.....	71
313	3.274	Portable Character Set	71
314	3.275	Portable Filename Character Set.....	71
315	3.276	Positional Parameter.....	71
316	3.277	Preallocation	71
317	3.278	Preempted Process (or Thread).....	72
318	3.279	Previous Job	72
319	3.280	Printable Character	72
320	3.281	Printable File	72
321	3.282	Priority	72
322	3.283	Priority Band.....	72
323	3.284	Priority Inversion	72
324	3.285	Priority Scheduling.....	72
325	3.286	Priority-Based Scheduling.....	73
326	3.287	Privilege.....	73
327	3.288	Process	73
328	3.289	Process Group.....	73
329	3.290	Process Group ID	73
330	3.291	Process Group Leader.....	73
331	3.292	Process Group Lifetime	73
332	3.293	Process ID.....	74
333	3.294	Process Lifetime.....	74
334	3.295	Process Memory Locking.....	74
335	3.296	Process Termination.....	74
336	3.297	Process-To-Process Communication	74
337	3.298	Process Virtual Time	74
338	3.299	Program	75
339	3.300	Protocol.....	75
340	3.301	Pseudo-Terminal	75
341	3.302	Radix Character	75
342	3.303	Read-Only File System	75
343	3.304	Read-Write Lock.....	75
344	3.305	Real Group ID.....	75
345	3.306	Real Time	76
346	3.307	Realtime Signal Extension	76
347	3.308	Real User ID	76
348	3.309	Record	76
349	3.310	Redirection	76
350	3.311	Redirection Operator	76
351	3.312	Reentrant Function	76
352	3.313	Referenced Shared Memory Object.....	76
353	3.314	Refresh	76
354	3.315	Regular Expression	77

355	3.316	Region	77
356	3.317	Regular File	77
357	3.318	Relative Pathname	77
358	3.319	Relocatable File	77
359	3.320	Relocation	77
360	3.321	Requested Batch Service	77
361	3.322	(Time) Resolution	77
362	3.323	Robust Mutex	78
363	3.324	Root Directory	78
364	3.325	Runnable Process (or Thread)	78
365	3.326	Running Process (or Thread)	78
366	3.327	Saved Resource Limits	78
367	3.328	Saved Set-Group-ID	78
368	3.329	Saved Set-User-ID	78
369	3.330	Scheduling	78
370	3.331	Scheduling Allocation Domain	79
371	3.332	Scheduling Contention Scope	79
372	3.333	Scheduling Policy	79
373	3.334	Screen	79
374	3.335	Scroll	79
375	3.336	Semaphore	79
376	3.337	Session	79
377	3.338	Session Leader	80
378	3.339	Session Lifetime	80
379	3.340	Shared Memory Object	80
380	3.341	Shell	80
381	3.342	Shell, the	80
382	3.343	Shell Script	80
383	3.344	Signal	80
384	3.345	Signal Stack	80
385	3.346	Single-Quote	81
386	3.347	Slash	81
387	3.348	Socket	81
388	3.349	Socket Address	81
389	3.350	Soft Limit	81
390	3.351	Source Code	81
391	3.352	Space Character (<space>)	82
392	3.353	Spawn	82
393	3.354	Special Built-In	82
394	3.355	Special Parameter	82
395	3.356	Spin Lock	82
396	3.357	Sporadic Server	82
397	3.358	Standard Error	82
398	3.359	Standard Input	82
399	3.360	Standard Output	82
400	3.361	Standard Utilities	83
401	3.362	Stream	83
402	3.363	STREAM	83
403	3.364	STREAM End	83
404	3.365	STREAM Head	83
405	3.366	STREAMS Multiplexor	83
406	3.367	String	83

Contents

407	3.368	Subshell.....	84
408	3.369	Successfully Transferred	84
409	3.370	Supplementary Group ID	84
410	3.371	Suspended Job	84
411	3.372	Symbolic Constant	84
412	3.373	Symbolic Link	84
413	3.374	Synchronized Input and Output.....	85
414	3.375	Synchronized I/O Completion	85
415	3.376	Synchronized I/O Data Integrity Completion.....	85
416	3.377	Synchronized I/O File Integrity Completion	85
417	3.378	Synchronized I/O Operation	85
418	3.379	Synchronous I/O Operation.....	85
419	3.380	Synchronously-Generated Signal	85
420	3.381	System.....	86
421	3.382	System Boot.....	86
422	3.383	System Clock.....	86
423	3.384	System Console	86
424	3.385	System Crash	86
425	3.386	System Databases.....	86
426	3.387	System Documentation	86
427	3.388	System Process.....	86
428	3.389	System Reboot	87
429	3.390	System Trace Event	87
430	3.391	System-Wide	87
431	3.392	Tab Character (<tab>).....	87
432	3.393	Terminal (or Terminal Device)	87
433	3.394	Text Column.....	87
434	3.395	Text File.....	88
435	3.396	Thread.....	88
436	3.397	Thread ID	88
437	3.398	Thread List	88
438	3.399	Thread-Safe	88
439	3.400	Thread-Specific Data Key	88
440	3.401	Tilde.....	89
441	3.402	Timeouts.....	89
442	3.403	Timer	89
443	3.404	Timer Overrun	89
444	3.405	Token.....	89
445	3.406	Trace Analyzer Process.....	89
446	3.407	Trace Controller Process.....	89
447	3.408	Trace Event.....	89
448	3.409	Trace Event Type	89
449	3.410	Trace Event Type Mapping	90
450	3.411	Trace Filter.....	90
451	3.412	Trace Generation Version	90
452	3.413	Trace Log	90
453	3.414	Trace Point.....	90
454	3.415	Trace Stream.....	90
455	3.416	Trace Stream Identifier	90
456	3.417	Trace System	90
457	3.418	Traced Process.....	90
458	3.419	Tracing Status of a Trace Stream	90

459	3.420	Typed Memory Name Space	91
460	3.421	Typed Memory Object	91
461	3.422	Typed Memory Pool	91
462	3.423	Typed Memory Port	91
463	3.424	Unbind	91
464	3.425	Unit Data	91
465	3.426	Upshifting	91
466	3.427	User Database	92
467	3.428	User ID	92
468	3.429	User Name	92
469	3.430	User Trace Event	92
470	3.431	Utility	92
471	3.432	Variable	92
472	3.433	Vertical-Tab Character (<vertical-tab>)	93
473	3.434	White Space	93
474	3.435	Wide-Character Code (C Language)	93
475	3.436	Wide-Character Input/Output Functions	93
476	3.437	Wide-Character String	93
477	3.438	Word	94
478	3.439	Working Directory (or Current Working Directory)	94
479	3.440	Worldwide Portability Interface	94
480	3.441	Write	94
481	3.442	XSI	94
482	3.443	XSI-Conformant	94
483	3.444	Zombie Process	94
484	3.445	±0	94
485	Chapter 4	General Concepts	95
486	4.1	Concurrent Execution	95
487	4.2	Directory Protection	95
488	4.3	Extended Security Controls	95
489	4.4	File Access Permissions	96
490	4.5	File Hierarchy	96
491	4.6	Filenames	97
492	4.7	Filename Portability	97
493	4.8	File Times Update	97
494	4.9	Host and Network Byte Orders	98
495	4.10	Measurement of Execution Time	98
496	4.11	Memory Synchronization	98
497	4.12	Pathname Resolution	99
498	4.13	Process ID Reuse	100
499	4.14	Scheduling Policy	100
500	4.15	Seconds Since the Epoch	100
501	4.16	Semaphore	101
502	4.17	Thread-Safety	101
503	4.18	Tracing	101
504	4.19	Treatment of Error Conditions for Mathematical Functions	104
505	4.19.1	Domain Error	104
506	4.19.2	Pole Error	104
507	4.19.3	Range Error	104
508	4.19.3.1	Result Overflows	104
509	4.19.3.2	Result Underflows	104

Contents

510	4.20	Treatment of NaN Arguments for the Mathematical	
511		Functions	105
512	4.21	Utility	105
513	4.22	Variable Assignment.....	106
514	Chapter 5	File Format Notation.....	107
515	Chapter 6	Character Set	111
516	6.1	Portable Character Set	111
517	6.2	Character Encoding	114
518	6.3	C Language Wide-Character Codes	115
519	6.4	Character Set Description File.....	115
520	6.4.1	State-Dependent Character Encodings	118
521	Chapter 7	Locale	121
522	7.1	General.....	121
523	7.2	POSIX Locale	122
524	7.3	Locale Definition	122
525	7.3.1	LC_CTYPE	124
526	7.3.1.1	LC_CTYPE Category in the POSIX Locale	128
527	7.3.2	LC_COLLATE.....	132
528	7.3.2.1	The collating-element Keyword.....	133
529	7.3.2.2	The collating-symbol Keyword.....	133
530	7.3.2.3	The order_start Keyword.....	134
531	7.3.2.4	Collation Order.....	134
532	7.3.2.5	The order_end Keyword	137
533	7.3.2.6	LC_COLLATE Category in the POSIX Locale	137
534	7.3.3	LC_MONETARY	139
535	7.3.3.1	LC_MONETARY Category in the POSIX Locale.....	142
536	7.3.4	LC_NUMERIC.....	143
537	7.3.4.1	LC_NUMERIC Category in the POSIX Locale	144
538	7.3.5	LC_TIME	144
539	7.3.5.1	LC_TIME Locale Definition.....	144
540	7.3.5.2	LC_TIME C-Language Access.....	146
541	7.3.5.3	LC_TIME Category in the POSIX Locale.....	148
542	7.3.6	LC_MESSAGES	150
543	7.3.6.1	LC_MESSAGES Category in the POSIX Locale.....	150
544	7.4	Locale Definition Grammar	151
545	7.4.1	Locale Lexical Conventions	151
546	7.4.2	Locale Grammar.....	152
547	Chapter 8	Environment Variables.....	159
548	8.1	Environment Variable Definition.....	159
549	8.2	Internationalization Variables	160
550	8.3	Other Environment Variables.....	163
551	Chapter 9	Regular Expressions.....	167
552	9.1	Regular Expression Definitions.....	167
553	9.2	Regular Expression General Requirements.....	168
554	9.3	Basic Regular Expressions	169
555	9.3.1	BREs Matching a Single Character or Collating Element	169
556	9.3.2	BRE Ordinary Characters.....	169

557	9.3.3	BRE Special Characters	169
558	9.3.4	Periods in BREs	170
559	9.3.5	RE Bracket Expression.....	170
560	9.3.6	BREs Matching Multiple Characters	172
561	9.3.7	BRE Precedence	173
562	9.3.8	BRE Expression Anchoring.....	173
563	9.4	Extended Regular Expressions.....	174
564	9.4.1	EREs Matching a Single Character or Collating Element	174
565	9.4.2	ERE Ordinary Characters.....	174
566	9.4.3	ERE Special Characters	174
567	9.4.4	Periods in EREs	175
568	9.4.5	ERE Bracket Expression	175
569	9.4.6	EREs Matching Multiple Characters	175
570	9.4.7	ERE Alternation.....	176
571	9.4.8	ERE Precedence	176
572	9.4.9	ERE Expression Anchoring.....	176
573	9.5	Regular Expression Grammar	177
574	9.5.1	BRE/ERE Grammar Lexical Conventions.....	177
575	9.5.2	RE and Bracket Expression Grammar.....	178
576	9.5.3	ERE Grammar.....	180
577	Chapter 10	Directory Structure and Devices	183
578	10.1	Directory Structure and Files.....	183
579	10.2	Output Devices and Terminal Types.....	184
580	Chapter 11	General Terminal Interface	185
581	11.1	Interface Characteristics	185
582	11.1.1	Opening a Terminal Device File.....	185
583	11.1.2	Process Groups	185
584	11.1.3	The Controlling Terminal.....	186
585	11.1.4	Terminal Access Control	186
586	11.1.5	Input Processing and Reading Data	187
587	11.1.6	Canonical Mode Input Processing.....	187
588	11.1.7	Non-Canonical Mode Input Processing.....	188
589	11.1.8	Writing Data and Output Processing.....	189
590	11.1.9	Special Characters	189
591	11.1.10	Modem Disconnect	190
592	11.1.11	Closing a Terminal Device File.....	190
593	11.2	Parameters that Can be Set	191
594	11.2.1	The termios Structure	191
595	11.2.2	Input Modes.....	191
596	11.2.3	Output Modes.....	192
597	11.2.4	Control Modes	194
598	11.2.5	Local Modes	195
599	11.2.6	Special Control Characters.....	196
600	Chapter 12	Utility Conventions.....	199
601	12.1	Utility Argument Syntax.....	199
602	12.2	Utility Syntax Guidelines.....	201
603	Chapter 13	Headers	205

604	Volume 2	System Interfaces, Issue 7	443
605	Chapter 1	Introduction	445
606	1.1	Relationship to Other Formal Standards.....	445
607	1.2	Format of Entries.....	445
608	Chapter 2	General Information	447
609	2.1	Use and Implementation of Functions.....	447
610	2.2	The Compilation Environment.....	448
611	2.2.1	POSIX.1 Symbols.....	448
612	2.2.1.1	The <code>_POSIX_C_SOURCE</code> Feature Test Macro.....	448
613	2.2.1.2	The <code>_XOPEN_SOURCE</code> Feature Test Macro.....	448
614	2.2.2	The Name Space.....	449
615	2.3	Error Numbers.....	456
616	2.3.1	Additional Error Numbers.....	462
617	2.4	Signal Concepts.....	463
618	2.4.1	Signal Generation and Delivery.....	463
619	2.4.2	Realtime Signal Generation and Delivery.....	464
620	2.4.3	Signal Actions.....	465
621	2.4.4	Signal Effects on Other Functions.....	469
622	2.5	Standard I/O Streams.....	469
623	2.5.1	Interaction of File Descriptors and Standard I/O Streams.....	470
624	2.5.2	Stream Orientation and Encoding Rules.....	472
625	2.6	STREAMS.....	473
626	2.6.1	Accessing STREAMS.....	474
627	2.7	XSI Interprocess Communication.....	474
628	2.7.1	IPC General Description.....	475
629	2.8	Realtime.....	476
630	2.8.1	Realtime Signals.....	476
631	2.8.2	Asynchronous I/O.....	476
632	2.8.3	Memory Management.....	478
633	2.8.3.1	Memory Locking.....	478
634	2.8.3.2	Memory Mapped Files.....	478
635	2.8.3.3	Memory Protection.....	478
636	2.8.3.4	Typed Memory Objects.....	479
637	2.8.4	Process Scheduling.....	479
638	2.8.5	Clocks and Timers.....	484
639	2.9	Threads.....	485
640	2.9.1	Thread-Safety.....	486
641	2.9.2	Thread IDs.....	486
642	2.9.3	Thread Mutexes.....	487
643	2.9.4	Thread Scheduling.....	488
644	2.9.5	Thread Cancellation.....	490
645	2.9.5.1	Cancelability States.....	490
646	2.9.5.2	Cancellation Points.....	491
647	2.9.5.3	Thread Cancellation Cleanup Handlers.....	494
648	2.9.5.4	Async-Cancel Safety.....	494
649	2.9.6	Thread Read-Write Locks.....	494
650	2.9.7	Thread Interactions with Regular File Operations.....	495
651	2.9.8	Use of Application-Managed Thread Stacks.....	495
652	2.10	Sockets.....	496
653	2.10.1	Address Families.....	496

654	2.10.2	Addressing	496
655	2.10.3	Protocols	496
656	2.10.4	Routing	496
657	2.10.5	Interfaces	496
658	2.10.6	Socket Types.....	497
659	2.10.7	Socket I/O Mode.....	498
660	2.10.8	Socket Owner.....	498
661	2.10.9	Socket Queue Limits.....	498
662	2.10.10	Pending Error	498
663	2.10.11	Socket Receive Queue.....	498
664	2.10.12	Socket Out-of-Band Data State.....	499
665	2.10.13	Connection Indication Queue	499
666	2.10.14	Signals.....	499
667	2.10.15	Asynchronous Errors.....	500
668	2.10.16	Use of Options	500
669	2.10.17	Use of Sockets for Local UNIX Connections	503
670	2.10.17.1	Headers.....	503
671	2.10.18	Use of Sockets over Internet Protocols.....	503
672	2.10.19	Use of Sockets over Internet Protocols Based on IPv4.....	504
673	2.10.19.1	Headers.....	504
674	2.10.20	Use of Sockets over Internet Protocols Based on IPv6.....	504
675	2.10.20.1	Addressing	504
676	2.10.20.2	Compatibility with IPv4.....	505
677	2.10.20.3	Interface Identification	505
678	2.10.20.4	Options	506
679	2.10.20.5	Headers.....	507
680	2.11	Tracing	507
681	2.11.1	Tracing Data Definitions	509
682	2.11.1.1	Structures.....	509
683	2.11.1.2	Trace Stream Attributes.....	513
684	2.11.2	Trace Event Type Definitions.....	513
685	2.11.2.1	System Trace Event Type Definitions.....	513
686	2.11.2.2	User Trace Event Type Definitions	516
687	2.11.3	Trace Functions.....	517
688	2.12	Data Types.....	518
689	2.12.1	Defined Types.....	518
690	2.12.2	The char Type.....	519
691	2.12.3	Pointer Types	519
692	Chapter 3	System Interfaces.....	521
693	Volume 3	Shell and Utilities, Issue 7	2225
694	Chapter 1	Introduction.....	2227
695	1.1	Relationship to Other Documents	2227
696	1.1.1	System Interfaces.....	2227
697	1.1.1.1	Process Attributes	2227
698	1.1.1.2	Concurrent Execution of Processes	2227
699	1.1.1.3	File Access Permissions.....	2228
700	1.1.1.4	File Read, Write, and Creation	2228
701	1.1.1.5	File Removal	2230
702	1.1.1.6	File Time Values.....	2230

Contents

703	1.1.1.7	File Contents	2230
704	1.1.1.8	Pathname Resolution.....	2231
705	1.1.1.9	Changing the Current Working Directory.....	2231
706	1.1.1.10	Establish the Locale.....	2231
707	1.1.1.11	Actions Equivalent to Functions.....	2231
708	1.1.2	Concepts Derived from the ISO C Standard	2231
709	1.1.2.1	Arithmetic Precision and Operations.....	2231
710	1.1.2.2	Mathematical Functions.....	2233
711	1.2	Utility Limits.....	2233
712	1.3	Grammar Conventions.....	2235
713	1.4	Utility Description Defaults.....	2235
714	1.5	Considerations for Utilities in Support of Files of Arbitrary Size.....	2243
715			
716	1.6	Built-In Utilities	2244
717	Chapter 2	Shell Command Language	2245
718	2.1	Shell Introduction.....	2245
719	2.2	Quoting.....	2246
720	2.2.1	Escape Character (Backslash).....	2246
721	2.2.2	Single-Quotes.....	2246
722	2.2.3	Double-Quotes.....	2246
723	2.3	Token Recognition.....	2247
724	2.3.1	Alias Substitution.....	2248
725	2.4	Reserved Words.....	2249
726	2.5	Parameters and Variables.....	2249
727	2.5.1	Positional Parameters	2249
728	2.5.2	Special Parameters	2250
729	2.5.3	Shell Variables.....	2250
730	2.6	Word Expansions	2253
731	2.6.1	Tilde Expansion.....	2253
732	2.6.2	Parameter Expansion.....	2254
733	2.6.3	Command Substitution.....	2256
734	2.6.4	Arithmetic Expansion.....	2257
735	2.6.5	Field Splitting	2258
736	2.6.6	Pathname Expansion.....	2259
737	2.6.7	Quote Removal.....	2259
738	2.7	Redirection	2259
739	2.7.1	Redirecting Input	2260
740	2.7.2	Redirecting Output	2260
741	2.7.3	Appending Redirected Output	2260
742	2.7.4	Here-Document.....	2260
743	2.7.5	Duplicating an Input File Descriptor	2261
744	2.7.6	Duplicating an Output File Descriptor	2261
745	2.7.7	Open File Descriptors for Reading and Writing.....	2262
746	2.8	Exit Status and Errors	2262
747	2.8.1	Consequences of Shell Errors	2262
748	2.8.2	Exit Status for Commands	2262
749	2.9	Shell Commands	2263
750	2.9.1	Simple Commands.....	2263
751	2.9.1.1	Command Search and Execution.....	2264
752	2.9.2	Pipelines	2265
753	2.9.3	Lists	2266

754	2.9.3.1	Asynchronous Lists	2266
755	2.9.3.2	Sequential Lists.....	2267
756	2.9.3.3	AND Lists.....	2267
757	2.9.3.4	OR Lists	2267
758	2.9.4	Compound Commands.....	2268
759	2.9.4.1	Grouping Commands.....	2268
760	2.9.4.2	The for Loop.....	2268
761	2.9.4.3	Case Conditional Construct.....	2269
762	2.9.4.4	The if Conditional Construct.....	2269
763	2.9.4.5	The while Loop.....	2270
764	2.9.4.6	The until Loop	2270
765	2.9.5	Function Definition Command	2270
766	2.10	Shell Grammar.....	2271
767	2.10.1	Shell Grammar Lexical Conventions.....	2271
768	2.10.2	Shell Grammar Rules.....	2272
769	2.11	Signals and Error Handling.....	2277
770	2.12	Shell Execution Environment.....	2277
771	2.13	Pattern Matching Notation	2278
772	2.13.1	Patterns Matching a Single Character	2278
773	2.13.2	Patterns Matching Multiple Characters.....	2279
774	2.13.3	Patterns Used for Filename Expansion.....	2279
775	2.14	Special Built-In Utilities.....	2280
776	Chapter 3	Batch Environment Services.....	2319
777	3.1	General Concepts	2319
778	3.1.1	Batch Client-Server Interaction.....	2319
779	3.1.2	Batch Queues	2320
780	3.1.3	Batch Job Creation.....	2320
781	3.1.4	Batch Job Tracking.....	2320
782	3.1.5	Batch Job Routing.....	2320
783	3.1.6	Batch Job Execution	2321
784	3.1.7	Batch Job Exit.....	2321
785	3.1.8	Batch Job Abort.....	2321
786	3.1.9	Batch Authorization.....	2322
787	3.1.10	Batch Administration	2322
788	3.1.11	Batch Notification	2322
789	3.2	Batch Services	2322
790	3.2.1	Batch Job States.....	2323
791	3.2.2	Deferred Batch Services.....	2324
792	3.2.2.1	Batch Job Execution	2324
793	3.2.2.2	Batch Job Routing.....	2331
794	3.2.2.3	Batch Job Exit.....	2331
795	3.2.2.4	Batch Server Restart.....	2332
796	3.2.2.5	Batch Job Abort.....	2332
797	3.2.3	Requested Batch Services.....	2333
798	3.2.3.1	Delete Batch Job Request.....	2333
799	3.2.3.2	Hold Batch Job Request.....	2334
800	3.2.3.3	Batch Job Message Request.....	2334
801	3.2.3.4	Batch Job Status Request	2335
802	3.2.3.5	Locate Batch Job Request	2335
803	3.2.3.6	Modify Batch Job Request.....	2335
804	3.2.3.7	Move Batch Job Request.....	2336

Contents

805	3.2.3.8	Queue Batch Job Request.....	2336
806	3.2.3.9	Batch Queue Status Request.....	2337
807	3.2.3.10	Release Batch Job Request.....	2337
808	3.2.3.11	Rerun Batch Job Request.....	2338
809	3.2.3.12	Select Batch Jobs Request.....	2338
810	3.2.3.13	Server Shutdown Request.....	2338
811	3.2.3.14	Server Status Request.....	2339
812	3.2.3.15	Signal Batch Job Request.....	2339
813	3.2.3.16	Track Batch Job Request.....	2339
814	3.3	Common Behavior for Batch Environment Utilities.....	2340
815	3.3.1	Batch Job Identifier.....	2340
816	3.3.2	Destination.....	2341
817	3.3.3	Multiple Keyword-Value Pairs.....	2341
818	Chapter 4	Utilities.....	2343
819	Volume 4	Rationale (Informative), Issue 7.....	3309
820	Part A	Base Definitions.....	3311
821	Appendix A	Rationale for Base Definitions.....	3313
822	A.1	Introduction.....	3313
823	A.1.1	Scope.....	3313
824	A.1.2	Conformance.....	3316
825	A.1.3	Normative References.....	3316
826	A.1.4	Change History.....	3316
827	A.1.5	Terminology.....	3316
828	A.1.6	Definitions and Concepts.....	3318
829	A.1.7	Portability.....	3318
830	A.1.7.1	Codes.....	3318
831	A.1.7.2	Margin Code Notation.....	3319
832	A.2	Conformance.....	3319
833	A.2.1	Implementation Conformance.....	3319
834	A.2.1.1	Requirements.....	3319
835	A.2.1.2	Documentation.....	3319
836	A.2.1.3	POSIX Conformance.....	3320
837	A.2.1.4	XSI Conformance.....	3321
838	A.2.1.5	Option Groups.....	3321
839	A.2.1.6	Options.....	3322
840	A.2.2	Application Conformance.....	3322
841	A.2.2.1	Strictly Conforming POSIX Application.....	3322
842	A.2.2.2	Conforming POSIX Application.....	3322
843	A.2.2.3	Conforming POSIX Application Using Extensions.....	3322
844	A.2.2.4	Strictly Conforming XSI Application.....	3323
845	A.2.2.5	Conforming XSI Application Using Extensions.....	3323
846	A.2.3	Language-Dependent Services for the C Programming Language.....	3323
847			
848	A.2.4	Other Language-Related Specifications.....	3323
849	A.3	Definitions.....	3323
850	A.4	General Concepts.....	3345
851	A.4.1	Concurrent Execution.....	3345

852	A.4.2	Directory Protection.....	3345
853	A.4.3	Extended Security Controls.....	3346
854	A.4.4	File Access Permissions.....	3346
855	A.4.5	File Hierarchy.....	3346
856	A.4.6	Filenames.....	3346
857	A.4.7	Filename Portability.....	3348
858	A.4.8	File Times Update.....	3348
859	A.4.9	Host and Network Byte Order.....	3348
860	A.4.10	Measurement of Execution Time.....	3348
861	A.4.11	Memory Synchronization.....	3349
862	A.4.12	Pathname Resolution.....	3350
863	A.4.13	Process ID Reuse.....	3351
864	A.4.14	Scheduling Policy.....	3352
865	A.4.15	Seconds Since the Epoch.....	3352
866	A.4.16	Semaphore.....	3353
867	A.4.17	Thread-Safety.....	3353
868	A.4.18	Tracing.....	3353
869	A.4.19	Treatment of Error Conditions for Mathematical Functions.....	3353
870			
871	A.4.20	Treatment of NaN Arguments for Mathematical Functions.....	3353
872			
873	A.4.21	Utility.....	3353
874	A.4.22	Variable Assignment.....	3354
875	A.5	File Format Notation.....	3354
876	A.6	Character Set.....	3354
877	A.6.1	Portable Character Set.....	3354
878	A.6.2	Character Encoding.....	3355
879	A.6.3	C Language Wide-Character Codes.....	3355
880	A.6.4	Character Set Description File.....	3355
881	A.6.4.1	State-Dependent Character Encodings.....	3355
882	A.7	Locale.....	3357
883	A.7.1	General.....	3357
884	A.7.2	POSIX Locale.....	3358
885	A.7.3	Locale Definition.....	3358
886	A.7.3.1	LC_CTYPE.....	3359
887	A.7.3.2	LC_COLLATE.....	3360
888	A.7.3.3	LC_MONETARY.....	3362
889	A.7.3.4	LC_NUMERIC.....	3363
890	A.7.3.5	LC_TIME.....	3363
891	A.7.3.6	LC_MESSAGES.....	3364
892	A.7.4	Locale Definition Grammar.....	3364
893	A.7.4.1	Locale Lexical Conventions.....	3365
894	A.7.4.2	Locale Grammar.....	3365
895	A.7.5	Locale Definition Example.....	3365
896	A.8	Environment Variables.....	3368
897	A.8.1	Environment Variable Definition.....	3368
898	A.8.2	Internationalization Variables.....	3369
899	A.8.3	Other Environment Variables.....	3370
900	A.9	Regular Expressions.....	3371
901	A.9.1	Regular Expression Definitions.....	3371
902	A.9.2	Regular Expression General Requirements.....	3372
903	A.9.3	Basic Regular Expressions.....	3373

Contents

904	A.9.3.1	BREs Matching a Single Character or Collating Element.....	3373
905			
906	A.9.3.2	BRE Ordinary Characters.....	3373
907	A.9.3.3	BRE Special Characters	3373
908	A.9.3.4	Periods in BREs	3373
909	A.9.3.5	RE Bracket Expression.....	3373
910	A.9.3.6	BREs Matching Multiple Characters.....	3375
911	A.9.3.7	BRE Precedence	3375
912	A.9.3.8	BRE Expression Anchoring.....	3375
913	A.9.4	Extended Regular Expressions.....	3376
914	A.9.4.1	EREs Matching a Single Character or Collating Element.....	3376
915			
916	A.9.4.2	ERE Ordinary Characters.....	3376
917	A.9.4.3	ERE Special Characters	3376
918	A.9.4.4	Periods in EREs	3376
919	A.9.4.5	ERE Bracket Expression.....	3377
920	A.9.4.6	EREs Matching Multiple Characters.....	3377
921	A.9.4.7	ERE Alternation.....	3377
922	A.9.4.8	ERE Precedence	3377
923	A.9.4.9	ERE Expression Anchoring.....	3377
924	A.9.5	Regular Expression Grammar.....	3377
925	A.9.5.1	BRE/ERE Grammar Lexical Conventions.....	3377
926	A.9.5.2	RE and Bracket Expression Grammar.....	3378
927	A.9.5.3	ERE Grammar.....	3378
928	A.10	Directory Structure and Devices.....	3378
929	A.10.1	Directory Structure and Files.....	3378
930	A.10.2	Output Devices and Terminal Types.....	3378
931	A.11	General Terminal Interface	3379
932	A.11.1	Interface Characteristics.....	3380
933	A.11.1.1	Opening a Terminal Device File.....	3380
934	A.11.1.2	Process Groups.....	3380
935	A.11.1.3	The Controlling Terminal.....	3381
936	A.11.1.4	Terminal Access Control	3381
937	A.11.1.5	Input Processing and Reading Data	3382
938	A.11.1.6	Canonical Mode Input Processing.....	3382
939	A.11.1.7	Non-Canonical Mode Input Processing.....	3382
940	A.11.1.8	Writing Data and Output Processing.....	3383
941	A.11.1.9	Special Characters.....	3383
942	A.11.1.10	Modem Disconnect.....	3383
943	A.11.1.11	Closing a Terminal Device File.....	3383
944	A.11.2	Parameters that Can be Set.....	3383
945	A.11.2.1	The termios Structure	3383
946	A.11.2.2	Input Modes.....	3383
947	A.11.2.3	Output Modes.....	3384
948	A.11.2.4	Control Modes.....	3384
949	A.11.2.5	Local Modes.....	3384
950	A.11.2.6	Special Control Characters.....	3384
951	A.12	Utility Conventions.....	3384
952	A.12.1	Utility Argument Syntax.....	3384
953	A.12.2	Utility Syntax Guidelines.....	3386
954	A.13	Headers.....	3388
955	A.13.1	Format of Entries.....	3388

956	A.13.2	Removed Headers in Issue 7	3389
957	Part B	System Interfaces.....	3391
958	Appendix B	Rationale for System Interfaces.....	3393
959	B.1	Introduction	3393
960	B.1.1	Change History	3393
961	B.1.2	Relationship to Other Formal Standards	3396
962	B.1.3	Format of Entries.....	3396
963	B.2	General Information	3397
964	B.2.1	Use and Implementation of Functions.....	3397
965	B.2.2	The Compilation Environment	3398
966	B.2.2.1	POSIX.1 Symbols.....	3398
967	B.2.2.2	The Name Space.....	3399
968	B.2.3	Error Numbers.....	3403
969	B.2.3.1	Additional Error Numbers	3406
970	B.2.4	Signal Concepts	3406
971	B.2.4.1	Signal Generation and Delivery	3408
972	B.2.4.2	Realtime Signal Generation and Delivery	3410
973	B.2.4.3	Signal Actions	3413
974	B.2.4.4	Signal Effects on Other Functions.....	3416
975	B.2.5	Standard I/O Streams	3416
976	B.2.5.1	Interaction of File Descriptors and Standard I/O Streams.....	3416
977	B.2.5.2	Stream Orientation and Encoding Rules	3416
978	B.2.6	STREAMS.....	3416
979	B.2.6.1	Accessing STREAMS	3417
980	B.2.7	XSI Interprocess Communication	3417
981	B.2.7.1	IPC General Information.....	3417
982	B.2.8	Realtime	3418
983	B.2.8.1	Realtime Signals	3423
984	B.2.8.2	Asynchronous I/O.....	3425
985	B.2.8.3	Memory Management.....	3427
986	B.2.8.4	Process Scheduling.....	3441
987	B.2.8.5	Clocks and Timers.....	3447
988	B.2.9	Threads	3463
989	B.2.9.1	Thread-Safety.....	3477
990	B.2.9.2	Thread IDs.....	3480
991	B.2.9.3	Thread Mutexes.....	3480
992	B.2.9.4	Thread Scheduling	3480
993	B.2.9.5	Thread Cancellation.....	3485
994	B.2.9.6	Thread Read-Write Locks.....	3489
995	B.2.9.7	Thread Interactions with Regular File Operations.....	3490
996	B.2.9.8	Use of Application-Managed Thread Stacks.....	3490
997	B.2.10	Sockets	3491
998	B.2.10.1	Address Families	3491
999	B.2.10.2	Addressing	3491
1000	B.2.10.3	Protocols	3491
1001	B.2.10.4	Routing	3491
1002	B.2.10.5	Interfaces	3491
1003	B.2.10.6	Socket Types.....	3491
1004			

Contents

1005	B.2.10.7	Socket I/O Mode.....	3491
1006	B.2.10.8	Socket Owner.....	3491
1007	B.2.10.9	Socket Queue Limits.....	3491
1008	B.2.10.10	Pending Error.....	3492
1009	B.2.10.11	Socket Receive Queue.....	3492
1010	B.2.10.12	Socket Out-of-Band Data State.....	3492
1011	B.2.10.13	Connection Indication Queue.....	3492
1012	B.2.10.14	Signals.....	3492
1013	B.2.10.15	Asynchronous Errors.....	3492
1014	B.2.10.16	Use of Options.....	3492
1015	B.2.10.17	Use of Sockets for Local UNIX Connections.....	3492
1016	B.2.10.18	Use of Sockets over Internet Protocols.....	3492
1017	B.2.10.19	Use of Sockets over Internet Protocols Based on IPv4.....	3492
1018	B.2.10.20	Use of Sockets over Internet Protocols Based on IPv6.....	3492
1019	B.2.11	Tracing.....	3492
1020	B.2.11.1	Objectives.....	3493
1021	B.2.11.2	Trace Model.....	3498
1022	B.2.11.3	Trace Programming Examples.....	3503
1023	B.2.11.4	Rationale on Trace for Debugging.....	3511
1024	B.2.11.5	Rationale on Trace Event Type Name Space.....	3511
1025	B.2.11.6	Rationale on Trace Events Type Filtering.....	3513
1026	B.2.11.7	Tracing, pthread API.....	3515
1027	B.2.11.8	Rationale on Triggering.....	3516
1028	B.2.11.9	Rationale on Timestamp Clock.....	3516
1029	B.2.11.10	Rationale on Different Overrun Conditions.....	3517
1030	B.2.12	Data Types.....	3517
1031	B.2.12.1	Defined Types.....	3517
1032	B.2.12.2	The char Type.....	3519
1033	B.2.12.3	Pointer Types.....	3519
1034	B.3	System Interfaces.....	3520
1035	B.3.1	System Interfaces Removed in this Version.....	3520
1036	B.3.1.1	bcmp.....	3520
1037	B.3.1.2	bcopy.....	3520
1038	B.3.1.3	bsd_signal.....	3520
1039	B.3.1.4	bzero.....	3520
1040	B.3.1.5	ecvt, fcvt, gcvt.....	3520
1041	B.3.1.6	ftime.....	3521
1042	B.3.1.7	getcontext, makecontext, swapcontext.....	3521
1043	B.3.1.8	gethostbyaddr, gethostbyname.....	3521
1044	B.3.1.9	getwd.....	3521
1045	B.3.1.10	h_errno.....	3521
1046	B.3.1.11	index.....	3521
1047	B.3.1.12	makecontext.....	3521
1048	B.3.1.13	mktemp.....	3521
1049	B.3.1.14	pthread_attr_getstackaddr, pthread_attr_setstackaddr.....	3521
1050	B.3.1.15	rindex.....	3522
1051	B.3.1.16	scalb.....	3522
1052	B.3.1.17	ualarm.....	3522
1053	B.3.1.18	usleep.....	3522
1054	B.3.1.19	vfork.....	3522
1055	B.3.1.20	wcswcs.....	3522
1056	B.3.2	Examples for Spawn.....	3522

1057	Part C	Shell and Utilities	3533
1058	Appendix C	Rationale for Shell and Utilities.....	3535
1059	C.1	Introduction	3535
1060	C.1.1	Change History	3535
1061	C.1.2	Relationship to Other Documents	3536
1062	C.1.2.1	System Interfaces.....	3536
1063	C.1.2.2	Concepts Derived from the ISO C Standard	3536
1064	C.1.3	Utility Limits.....	3537
1065	C.1.4	Grammar Conventions.....	3540
1066	C.1.5	Utility Description Defaults.....	3540
1067	C.1.6	Considerations for Utilities in Support of Files of Arbitrary Size.....	3543
1068			
1069	C.1.7	Built-In Utilities	3544
1070	C.2	Shell Command Language	3546
1071	C.2.1	Shell Introduction.....	3546
1072	C.2.2	Quoting.....	3546
1073	C.2.2.1	Escape Character (Backslash).....	3546
1074	C.2.2.2	Single-Quotes.....	3546
1075	C.2.2.3	Double-Quotes.....	3546
1076	C.2.3	Token Recognition.....	3548
1077	C.2.3.1	Alias Substitution.....	3548
1078	C.2.4	Reserved Words.....	3549
1079	C.2.5	Parameters and Variables.....	3549
1080	C.2.5.1	Positional Parameters	3549
1081	C.2.5.2	Special Parameters	3549
1082	C.2.5.3	Shell Variables.....	3550
1083	C.2.6	Word Expansions	3552
1084	C.2.6.1	Tilde Expansion	3552
1085	C.2.6.2	Parameter Expansion.....	3553
1086	C.2.6.3	Command Substitution	3553
1087	C.2.6.4	Arithmetic Expansion.....	3554
1088	C.2.6.5	Field Splitting	3557
1089	C.2.6.6	Pathname Expansion	3557
1090	C.2.6.7	Quote Removal.....	3557
1091	C.2.7	Redirection	3557
1092	C.2.7.1	Redirecting Input	3558
1093	C.2.7.2	Redirecting Output	3559
1094	C.2.7.3	Appending Redirected Output	3559
1095	C.2.7.4	Here-Document.....	3559
1096	C.2.7.5	Duplicating an Input File Descriptor	3559
1097	C.2.7.6	Duplicating an Output File Descriptor	3559
1098	C.2.7.7	Open File Descriptors for Reading and Writing.....	3559
1099	C.2.8	Exit Status and Errors	3559
1100	C.2.8.1	Consequences of Shell Errors	3559
1101	C.2.8.2	Exit Status for Commands	3559
1102	C.2.9	Shell Commands	3560
1103	C.2.9.1	Simple Commands.....	3560
1104	C.2.9.2	Pipelines	3562
1105	C.2.9.3	Lists	3563
1106	C.2.9.4	Compound Commands.....	3564
1107	C.2.9.5	Function Definition Command	3565

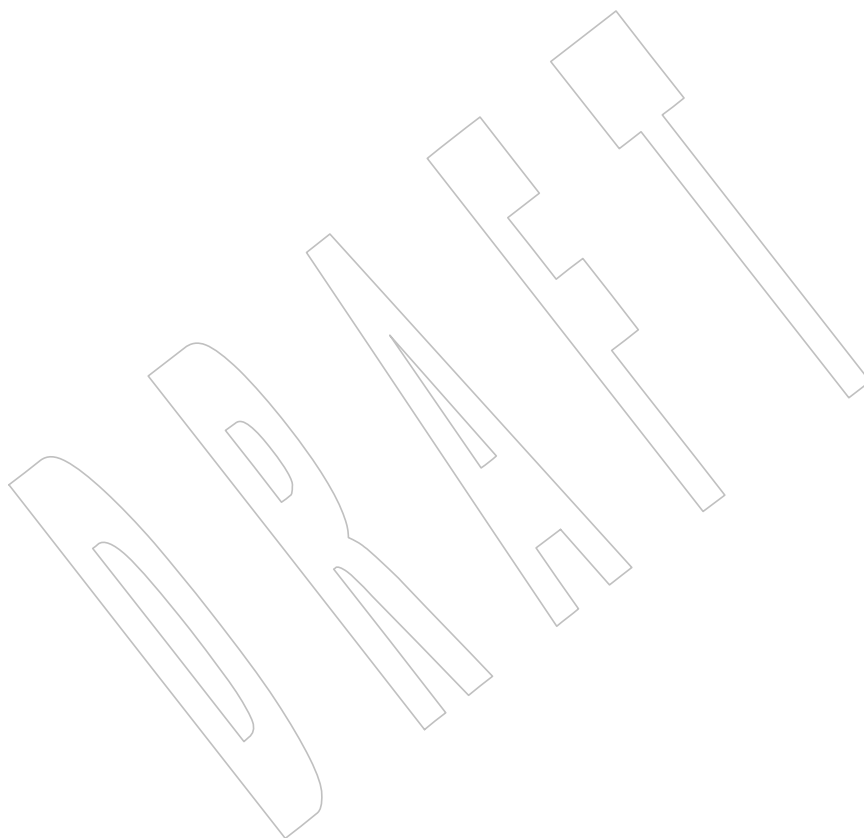
Contents

1108	C.2.10	Shell Grammar.....	3566
1109	C.2.10.1	Shell Grammar Lexical Conventions.....	3567
1110	C.2.10.2	Shell Grammar Rules.....	3567
1111	C.2.11	Signals and Error Handling.....	3568
1112	C.2.12	Shell Execution Environment.....	3568
1113	C.2.13	Pattern Matching Notation.....	3568
1114	C.2.13.1	Patterns Matching a Single Character.....	3568
1115	C.2.13.2	Patterns Matching Multiple Characters.....	3569
1116	C.2.13.3	Patterns Used for Filename Expansion.....	3569
1117	C.2.14	Special Built-In Utilities.....	3569
1118	C.3	Batch Environment Services and Utilities.....	3570
1119	C.3.1	Batch General Concepts.....	3573
1120	C.3.2	Batch Services.....	3575
1121	C.3.3	Common Behavior for Batch Environment Utilities.....	3576
1122	C.4	Utilities.....	3576
1123	Part D	Portability Considerations.....	3581
1124	Appendix D	Portability Considerations (Informative).....	3583
1125	D.1	User Requirements.....	3583
1126	D.1.1	Configuration Interrogation.....	3584
1127	D.1.2	Process Management.....	3584
1128	D.1.3	Access to Data.....	3584
1129	D.1.4	Access to the Environment.....	3584
1130	D.1.5	Access to Determinism and Performance Enhancements.....	3584
1131	D.1.6	Operating System-Dependent Profile.....	3584
1132	D.1.7	I/O Interaction.....	3585
1133	D.1.8	Internationalization Interaction.....	3585
1134	D.1.9	C-Language Extensions.....	3585
1135	D.1.10	Command Language.....	3585
1136	D.1.11	Interactive Facilities.....	3585
1137	D.1.12	Accomplish Multiple Tasks Simultaneously.....	3585
1138	D.1.13	Complex Data Manipulation.....	3585
1139	D.1.14	File Hierarchy Manipulation.....	3585
1140	D.1.15	Locale Configuration.....	3586
1141	D.1.16	Inter-User Communication.....	3586
1142	D.1.17	System Environment.....	3586
1143	D.1.18	Printing.....	3586
1144	D.1.19	Software Development.....	3586
1145	D.2	Portability Capabilities.....	3586
1146	D.2.1	Configuration Interrogation.....	3587
1147	D.2.2	Process Management.....	3587
1148	D.2.3	Access to Data.....	3588
1149	D.2.4	Access to the Environment.....	3589
1150	D.2.5	Bounded (Realtime) Response.....	3589
1151	D.2.6	Operating System-Dependent Profile.....	3590
1152	D.2.7	I/O Interaction.....	3590
1153	D.2.8	Internationalization Interaction.....	3590
1154	D.2.9	C-Language Extensions.....	3590
1155	D.2.10	Command Language.....	3590
1156	D.2.11	Interactive Facilities.....	3591

1157	D.2.12	Accomplish Multiple Tasks Simultaneously	3591
1158	D.2.13	Complex Data Manipulation	3592
1159	D.2.14	File Hierarchy Manipulation	3592
1160	D.2.15	Locale Configuration	3592
1161	D.2.16	Inter-User Communication	3592
1162	D.2.17	System Environment	3593
1163	D.2.18	Printing	3593
1164	D.2.19	Software Development	3593
1165	D.2.20	Future Growth	3593
1166	D.3	Profiling Considerations	3594
1167	D.3.1	Configuration Options	3594
1168	D.3.2	Configuration Options (Shell and Utilities)	3594
1169	D.3.3	Configurable Limits	3596
1170	D.3.4	Configuration Options (System Interfaces)	3596
1171	D.3.5	Configurable Limits	3601
1172	D.3.6	Optional Behavior	3603
1173	Part E	Subprofiling Considerations	3605
1174	Appendix E	Subprofiling Considerations (Informative)	3607
1175	E.1	Subprofiling Option Groups	3607
1176		Index	3613
1177	List of Figures		
1178	4-1	pax Format Archive Example	2937
1179	B-1	Example of a System with Typed Memory	3436
1180	B-2	Trace System Overview: for Offline Analysis	3498
1181	B-3	Trace System Overview: for Online Analysis	3499
1182	B-4	Trace System Overview: States of a Trace Stream	3501
1183	B-5	Trace Another Process	3511
1184	B-6	Trace Name Space Overview: With Third-Party Library	3512
1185	List of Tables		
1186	3-1	Job Control Job ID Formats	61
1187	5-1	Escape Sequences and Associated Actions	108
1188	6-1	Portable Character Set	111
1189	6-2	Control Character Set	116
1190	7-1	Valid Character Class Combinations	128
1191	10-1	Control Character Names	184
1192	2-1	Value of Level for Socket Options	501
1193	2-2	Socket-Level Options	501
1194	2-3	Trace Option: System Trace Events	515
1195	2-4	Trace and Trace Event Filter Options: System Trace Events	515
1196	2-5	Trace and Trace Log Options: System Trace Events	516
1197	2-6	Trace, Trace Log, and Trace Event Filter Options:	
1198		System Trace Events	516
1199	2-7	Trace Option: User Trace Event	517
1200	1-1	Actions when Creating a File that Already Exists	2229

Contents

1201	1-2	Selected ISO C Standard Operators and Control Flow	
1202		Keywords	2232
1203	1-3	Utility Limit Minimum Values	2233
1204	1-4	Symbolic Utility Limits	2234
1205	1-5	Regular Built-In Utilities	2244
1206	3-1	Batch Utilities.....	2319
1207	3-2	Environment Variable Summary	2323
1208	3-3	Next State Table.....	2325
1209	3-4	Results/Output Table.....	2326
1210	3-5	Batch Services Summary	2333
1211	4-1	Expressions in Decreasing Precedence in <i>awk</i>	2374
1212	4-2	Escape Sequences in <i>awk</i>	2379
1213	4-3	Operators in <i>bc</i>	2414
1214	4-4	Programming Environments: Type Sizes	2431
1215	4-5	Programming Environments: <i>c99</i> and <i>cc</i> Arguments	2432
1216	4-6	ASCII to EBCDIC Conversion.....	2520
1217	4-7	ASCII to IBM EBCDIC Conversion	2521
1218	4-8	File Utility Output Strings	2662
1219	4-9	Table Size Declarations in <i>lex</i>	2754
1220	4-10	Escape Sequences in <i>lex</i>	2756
1221	4-11	ERE Precedence in <i>lex</i>	2756
1222	4-12	Named Characters in <i>od</i>	2904
1223	4-13	ustar Header Block.....	2942
1224	4-14	ustar <i>mode</i> Field	2943
1225	4-15	Octet-Oriented <i>cpio</i> Archive Entry.....	2945
1226	4-16	Values for <i>cpio</i> <i>c_mode</i> Field	2946
1227	4-17	Variable Names and Default Headers in <i>ps</i>	2980
1228	4-18	Environment Variable Values (Utilities)	3029
1229	4-19	Control Character Names in <i>stty</i>	3111
1230	4-20	Circumflex Control Characters in <i>stty</i>	3111
1231	4-21	<i>uuencode</i> Base64 Values	3201
1232	4-22	Internal Limits in <i>yacc</i>	3302
1233	A-1	Historical Practice for Symbolic Links.....	3342



1234

Foreword

1235

Notes to Reviewers

1236

This section with side shading will not appear in the final copy. - Ed.

1237

This section will be completed in a later draft.

DRAFT

Introduction

1238

Note: This introduction is not part of POSIX.1-200x, Standard for Information Technology — Portable Operating System Interface (POSIX).

This draft standard was developed, and is maintained, by a joint working group of members of the IEEE Portable Applications Standards Committee, members of The Open Group, and members of ISO/IEC Joint Technical Committee 1. This joint working group is known as the Austin Group.¹

The Austin Group arose out of discussions amongst the parties which started in early 1998, leading to an initial meeting and formation of the group in September 1998. The purpose of the Austin Group is to develop and maintain the core open systems interfaces that are the POSIX[®] 1003.1 (and former 1003.2) standards, ISO/IEC 9945 Parts 1 to 4, and the core of the Single UNIX Specification.

The approach to specification development has been one of “write once, adopt everywhere”, with the deliverables being a set of specifications that carry the IEEE POSIX designation, The Open Group’s Technical Standard designation, and an ISO/IEC designation.

This unique development has combined both the industry-led efforts and the formal standardization activities into a single initiative, and included a wide spectrum of participants. The Austin Group continues as the maintenance body for this document.

Anyone wishing to participate in the Austin Group should contact the chair with their request. There are no fees for participation or membership. You may participate as an observer or as a contributor. You do not have to attend face-to-face meetings to participate; electronic participation is most welcome. For more information on the Austin Group and how to participate, see www.opengroup.org/austin.

Background

The developers of POSIX.1-200x represent a cross section of hardware manufacturers, vendors of operating systems and other software development tools, software designers, consultants, academics, authors, applications programmers, and others.

Conceptually, POSIX.1-200x describes a set of fundamental services needed for the efficient construction of application programs. Access to these services has been provided by defining an interface, using the C programming language, a command interpreter, and common utility programs that establish standard semantics and syntax. Since this interface enables application developers to write portable applications—it was developed with that goal in mind—it has been designated POSIX,² an acronym for Portable Operating System Interface.

Although originated to refer to the original IEEE Std 1003.1-1988, the name POSIX more correctly refers to a *family* of related standards: IEEE Std 1003.*n* and the parts of ISO/IEC 9945. In earlier editions of the IEEE standard, the term POSIX was used as a synonym for IEEE Std 1003.1-1988. A preferred term, POSIX.1, emerged. This maintained the advantages of readability of the symbol “POSIX” without being ambiguous with the POSIX family of

-
1. The Austin Group is named after the location of the inaugural meeting held at the IBM facility in Austin, Texas in September 1998.
 2. The name POSIX was suggested by Richard Stallman. It is expected to be pronounced *pahz-icks*, as in *positive*, not *poh-six*, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface.

XXX

Base Specifications, Issue 7 — Copyright © 2001-200x, IEEE and The Open Group. All rights reserved.

Introduction

1280 standards.

1281 **Audience**

1282 The intended audience for POSIX.1-200x is all persons concerned with an industry-wide
1283 standard operating system based on the UNIX system. This includes at least four groups of
1284 people:

- 1285 1. Persons buying hardware and software systems
- 1286 2. Persons managing companies that are deciding on future corporate computing directions
- 1287 3. Persons implementing operating systems, and especially
- 1288 4. Persons developing applications where portability is an objective

1289 **Purpose**

1290 Several principles guided the development of POSIX.1-200x:

- 1291 • Application-Oriented

1292 The basic goal was to promote portability of application programs across UNIX system
1293 environments by developing a clear, consistent, and unambiguous standard for the
1294 interface specification of a portable operating system based on the UNIX system
1295 documentation. POSIX.1-200x codifies the common, existing definition of the UNIX
1296 system.

- 1297 • Interface, Not Implementation

1298 POSIX.1-200x defines an interface, not an implementation. No distinction is made between
1299 library functions and system calls; both are referred to as functions. No details of the
1300 implementation of any function are given (although historical practice is sometimes
1301 indicated in the RATIONALE section). Symbolic names are given for constants (such as
1302 signals and error numbers) rather than numbers.

- 1303 • Source, Not Object, Portability

1304 POSIX.1-200x has been written so that a program written and translated for execution on
1305 one conforming implementation may also be translated for execution on another
1306 conforming implementation. POSIX.1-200x does not guarantee that executable (object or
1307 binary) code will execute under a different conforming implementation than that for which
1308 it was translated, even if the underlying hardware is identical.

- 1309 • The C Language

1310 The system interfaces and header definitions are written in terms of the standard C
1311 language as specified in the ISO C standard.

- 1312 • No Superuser, No System Administration

1313 There was no intention to specify all aspects of an operating system. System
1314 administration facilities and functions are excluded from this standard, and functions
1315 usable only by the superuser have not been included. Still, an implementation of the
1316 standard interface may also implement features not in POSIX.1-200x. POSIX.1-200x is also
1317 not concerned with hardware constraints or system maintenance.

- 1318 • Minimal Interface, Minimally Defined

1319 In keeping with the historical design principles of the UNIX system, the mandatory core
1320 facilities of POSIX.1-200x have been kept as minimal as possible. Additional capabilities
1321 have been added as optional extensions.

1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363

- Broadly Implementable

The developers of POSIX.1-200x endeavored to make all specified functions implementable across a wide range of existing and potential systems, including:

1. All of the current major systems that are ultimately derived from the original UNIX system code (Version 7 or later)
2. Compatible systems that are not derived from the original UNIX system code
3. Emulations hosted on entirely different operating systems
4. Networked systems
5. Distributed systems
6. Systems running on a broad range of hardware

No direct references to this goal appear in POSIX.1-200x, but some results of it are mentioned in the Rationale (Informative) volume.

- Minimal Changes to Historical Implementations

When the original version—IEEE Std 1003.1-1988—was published, there were no known historical implementations that did not have to change. However, there was a broad consensus on a set of functions, types, definitions, and concepts that formed an interface that was common to most historical implementations.

The adoption of the 1988 and 1990 IEEE system interface standards, the 1992 IEEE shell and utilities standard, the various Open Group (formerly X/Open) specifications, and IEEE Std 1003.1-2001 and its technical corrigenda have consolidated this consensus, and this version reflects the significantly increased level of consensus arrived at since the original versions. The authors of the original versions tried, as much as possible, to follow the principles below when creating new specifications:

1. By standardizing an interface like one in an historical implementation; for example, directories
2. By specifying an interface that is readily implementable in terms of, and backwards-compatible with, historical implementations, such as the extended *tar* format defined in the *pax* utility
3. By specifying an interface that, when added to an historical implementation, will not conflict with it; for example, the *sigaction()* function

POSIX.1-200x is specifically not a codification of a particular vendor's product.

It should be noted that implementations will have different kinds of extensions. Some will reflect "historical usage" and will be preserved for execution of pre-existing applications. These functions should be considered "obsolescent" and the standard functions used for new applications. Some extensions will represent functions beyond the scope of POSIX.1-200x. These need to be used with careful management to be able to adapt to future extensions of POSIX.1-200x and/or port to implementations that provide these services in a different manner.

- Minimal Changes to Existing Application Code

A goal of POSIX.1-200x was to minimize additional work for application developers. However, because every known historical implementation will have to change at least slightly to conform, some applications will have to change.

Introduction

1364 **POSIX.1-200x**

1365 POSIX.1-200x defines the Portable Operating System Interface (POSIX) requirements and
1366 consists of the following topics arranged as a series of volumes within the standard:

- 1367 • Base Definitions
- 1368 • System Interfaces
- 1369 • Shell and Utilities
- 1370 • Rationale (Informative)

1371 **Base Definitions**

1372 The Base Definitions volume provides common definitions for this standard, therefore readers
1373 should be familiar with it before using the other volumes.

1374 This volume is structured as follows:

- 1375 • [Chapter 1](#) is an introduction.
- 1376 • [Chapter 2](#) defines the conformance requirements.
- 1377 • [Chapter 3](#) defines general terms used.
- 1378 • [Chapter 4](#) describes general concepts used.
- 1379 • [Chapter 5](#) describes the notation used to specify file input and output formats in this
1380 volume and the Shell and Utilities volume.
- 1381 • [Chapter 6](#) describes the portable character set and the process of character set definition.
- 1382 • *locale* describes the syntax for defining internationalization locales as well as the POSIX
1383 locale provided on all systems.
- 1384 • [Chapter 8](#) describes the use of environment variables for internationalization and other
1385 purposes.
- 1386 • [Chapter 9](#) describes the syntax of pattern matching using regular expressions employed by
1387 many utilities and matched by the *regcomp()* and *regexexec()* functions.
- 1388 • [Chapter 10](#) describes files and devices found on all systems.
- 1389 • [Chapter 11](#) describes the asynchronous terminal interface for many of the functions in the
1390 System Interfaces volume and the *stty* utility in the Shell and Utilities volume.
- 1391 • [Chapter 12](#) describes the policies for command line argument construction and parsing.
- 1392 • [Chapter 13](#) defines the contents of headers which declare constants, macros, and data
1393 structures that are needed by programs using the services provided by the System
1394 Interfaces volume.

1395 Comprehensive references are available in the index.

1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436

System Interfaces

The System Interfaces volume describes the interfaces offered to application programs by POSIX-conformant systems. Readers are expected to be experienced C language programmers, and to be familiar with the Base Definitions volume.

This volume is structured as follows:

- [Chapter 1](#) explains the status of this volume and its relationship to other formal standards.
- [Chapter 2](#) contains important concepts, terms, and caveats relating to the rest of this volume.
- [Chapter 3](#) defines the functional interfaces to the POSIX-conformant system.

Comprehensive references are available in the index.

Shell and Utilities

The Shell and Utilities volume describes the commands and utilities offered to application programs on POSIX-conformant systems. Readers are expected to be familiar with the Base Definitions volume.

This volume is structured as follows:

- [Chapter 1](#) explains the status of this volume and its relationship to other formal standards. It also describes the defaults used by the utility descriptions in [Chapter 4](#).
- [Chapter 2](#) describes the command language used in POSIX-conformant systems.
- [Chapter 3](#) describes a set of services and utilities that are implemented on systems supporting the Batch Environment Services and Utilities option.
- [Chapter 4](#) consists of reference pages for all utilities available on POSIX-conformant systems.

Comprehensive references are available in the index.

Rationale (Informative)

This volume is being published to assist in the process of review. It contains historical information concerning the contents of this standard and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of POSIX.1-200x that may not be immediately apparent.

This volume is organized in parallel to the normative volumes of this standard, with a separate part for each of the three normative volumes.

Within this volume, the following terms are used:

base standard

The portions of POSIX.1-200x that are not optional, equivalent to the definitions of *classic* POSIX.1 and POSIX.2.

POSIX.0

Although this term is not used in the normative text of POSIX.1-200x, it is used in this volume to refer to IEEE Std 1003.0-1995.

POSIX.1b

Although this term is not used in the normative text of POSIX.1-200x, it is used in this volume to refer to the elements of the POSIX Realtime Extension amendment. (This was

Introduction

1437 earlier referred to as POSIX.4 during the standard development process.)

1438 **POSIX.1c**

1439 Although this term is not used in the normative text of POSIX.1-200x, it is used in this
1440 volume to refer to the POSIX Threads Extension amendment. (This was earlier referred to as
1441 POSIX.4a during the standard development process.)

1442 **standard developers**

1443 The individuals and companies in the development organizations responsible for
1444 POSIX.1-200x: the IEEE P1003.1 working groups, The Open Group Base working group,
1445 advised by the hundreds of individual technical experts who balloted the draft standards
1446 within the Austin Group, and the member bodies and technical experts of ISO/IEC
1447 JTC 1/SC22.

1448 **XSI option**

1449 The portions of POSIX.1-200x addressing the extension added for support of the Single
1450 UNIX Specification.

1451 **Typographical Conventions**

1452 The following typographical conventions are used throughout this standard. In the text, this
1453 standard is referred to as POSIX.1-200x, which is technically identical to The Open Group Base
1454 Specifications, Issue 7.

1455 The typographical conventions listed here are for ease of reading only. Editorial inconsistencies
1456 in the use of typography are unintentional and have no normative meaning in POSIX.1-200x.

Reference	Example	Notes
C-Language Data Structure	aiocb	
C-Language Data Structure Member	<i>aio_lio_opcode</i>	
C-Language Data Type	long	
C-Language External Variable	<i>errno</i>	
C-Language Function	<i>system()</i>	
C-Language Function Argument	<i>arg1</i>	
C-Language Function Family	<i>exec</i>	
C-Language Header	<sys/stat.h>	
C-Language Keyword	return	
C-Language Macro with Argument	<i>assert()</i>	
C-Language Macro with No Argument	INET_ADDRSTRLEN	
C-Language Preprocessing Directive	#define	
Commands within a Utility	a, c	
Conversion Specification, Specifier/Modifier Character	%A, g, E	1
Environment Variable	<i>PATH</i>	
Error Number	[EINTR]	
Example Output	Hello, World	
Filename	/tmp	
Literal Character	'c', '\r', '\'	2
Literal String	"abcde"	2
Optional Items in Utility Syntax	[]	
Parameter	<directory pathname>	
Special Character	<newline>	3
Symbolic Constant	_POSIX_VDISABLE	
Symbolic Limit, Configuration Value	{LINE_MAX}	4
Syntax	#include <sys/stat.h>	

1484
1485
1486
1487
1488
1489

Reference	Example	Notes
User Input and Example Code	echo Hello, World	5
Utility Name	awk	
Utility Operand	file_name	
Utility Option	-c	
Utility Option with Option-Argument	-w width	

1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505**Notes:**

1. Conversion specifications, specifier characters, and modifier characters are used primarily in date-related functions and utilities and the *fprintf* and *fscanf* formatting functions.
2. Unless otherwise noted, the quotes shall not be used as input or output. When used in a list item, the quotes are omitted. For literal characters, '\ ' (or any of the other sequences such as ' ' ') is the same as the C constant '\\\ ' (or '\ ' ').
3. The style selected for some of the special characters, such as <newline>, matches the form of the input given to the *localedef* utility. Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.
4. Names surrounded by braces represent symbolic limits or configuration values which may be declared in appropriate headers by means of the C **#define** construct.
5. Brackets shown in this font, "[]", are part of the syntax and do *not* indicate optional items. In syntax the '| ' symbol is used to separate alternatives, and ellipses ("...") are used to show that additional arguments are optional.

1506
1507
1508
1509
1510

Shading is used to identify extensions and options; see [Section 1.7.1](#) (on page 7).

Footnotes and notes within the body of the normative text are for information only (informative).

Informative sections (such as Rationale, Change History, Application Usage, and so on) are denoted by continuous shading bars in the margins.

1511
1512
1513
1514
1515

Ranges of values are indicated with parentheses or brackets as follows:

- (a,b) means the range of all values from a to b , including neither a nor b .
- $[a,b]$ means the range of all values from a to b , including a and b .
- $[a,b)$ means the range of all values from a to b , including a , but not b .
- $(a,b]$ means the range of all values from a to b , including b , but not a .

1516
1517
1518
1519
1520

Note: A symbolic limit beginning with POSIX is treated differently, depending on context. In a C-language header, the symbol `POSIXstring` (where *string* may contain underscores) is represented by the C identifier `_POSIXstring`, with a leading underscore required to prevent ISO C standard name space pollution. However, in other contexts, such as languages other than C, the leading underscore is not used because this requirement does not exist.

1521

Participants

1522

1523

POSIX.1-200x was prepared by the Austin Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society, The Open Group, and ISO/SC22.

1524

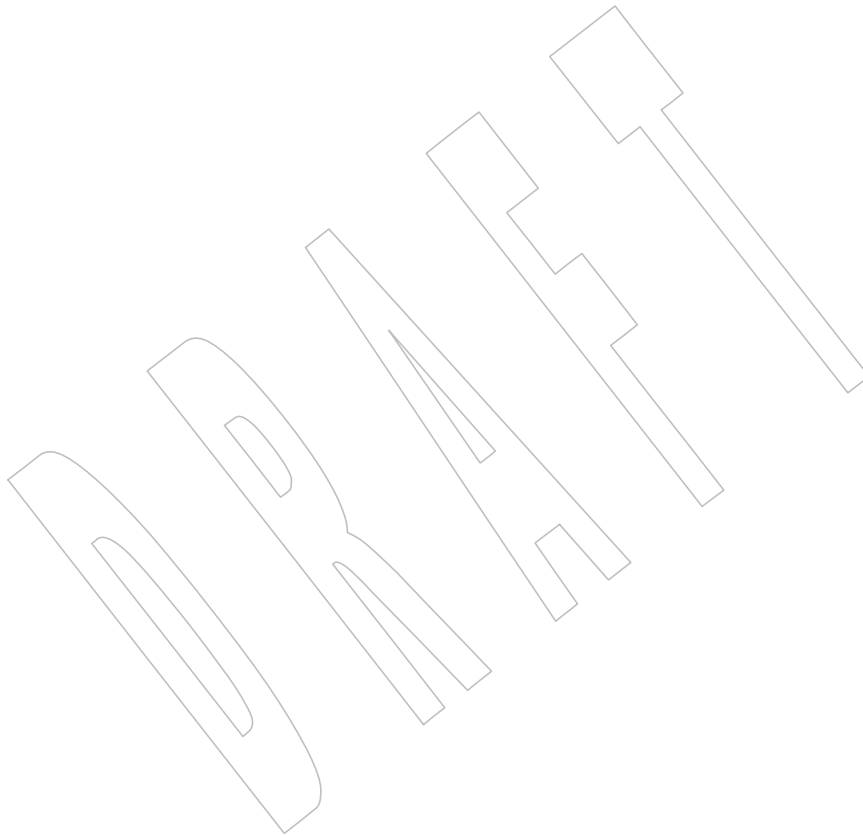
1525

1526

Notes to Reviewers

This section with side shading will not appear in the final copy. - Ed.

This section will be completed once the standard is approved.





Trademarks

1527

1528 The following information is given for the convenience of users of POSIX.1-200x and does not
1529 constitute an endorsement by the IEEE or The Open Group of these products. Equivalent
1530 products may be used if they can be shown to lead to the same results.

1531 There may be other products mentioned in the text that might be covered by trademark
1532 protection and readers are advised to verify them independently.

1533 1003.1™ is a trademark of the Institute of Electrical and Electronic Engineers, Inc.

1534 AIX® is a registered trademark of IBM Corporation.

1535 AT&T® is a registered trademark of AT&T in the U.S.A. and other countries.

1536 Boundaryless Information Flow™ and TOGAF™ are trademarks and Motif®, Making Standards
1537 Work®, OSF/1®, The Open Group®, UNIX®, and the “X” device are registered trademarks of
1538 The Open Group in the United States and other countries.

1539 BSD™ is a trademark of the University of California, Berkeley, U.S.A.

1540 Hewlett-Packard®, HP®, and HP-UX® are registered trademarks of Hewlett-Packard Company.

1541 IBM® is a registered trademark of International Business Machines Corporation.

1542 Linux® is a registered trademark of Linus Torvalds.

1543 POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

1544 Sun® and Sun Microsystems® are registered trademarks of Sun Microsystems, Inc.

1545 /usr/group® is a registered trademark of UniForum, the International Network of UNIX System
1546 Users.

Acknowledgements

1547

1548

1549

The contributions of the following organizations to the development of POSIX.1-200x are gratefully acknowledged:

1550

1551

- AT&T for permission to reproduce portions of its copyrighted System V Interface Definition (SVID) and material from the UNIX System V Release 2.0 documentation.

1552

1553

1554

- Hewlett-Packard Company, International Business Machines Corporation, Novell Inc., The Open Software Foundation, and Sun Microsystems Inc. for permission to reproduce portions of their copyrighted documentation

1555

- ISO/IEC JTC 1/SC 22/WG 14 C Language Committee

1556

- Red Hat Inc. for permission to reproduce portions of its copyrighted documentation

1557

1558

POSIX.1-200x was prepared by the Austin Group, a joint working group of the IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

Referenced Documents

1559

1560

Normative References

1561

Normative references for POSIX.1-200x are defined in [Section 1.3](#) (on page 4).

1562

Informative References

1563

The following documents are referenced in POSIX.1-200x:

1564

1984 /usr/group Standard

1565

/usr/group Standards Committee, Santa Clara, CA, UniForum 1984.

1566

Almasi and Gottlieb

1567

George S. Almasi and Allan Gottlieb, *Highly Parallel Computing*, The Benjamin/Cummings

1568

Publishing Company, Inc., 1989, ISBN: 0-8053-0177-1.

1569

ANSI C

1570

American National Standard for Information Systems: Standard X3.159-1989, Programming

1571

Language C.

1572

ANSI X3.226-1994

1573

American National Standard for Information Systems: Standard X3.226-1994, Programming

1574

Language Common LISP.

1575

Brawer

1576

Steven Brawer, *Introduction to Parallel Programming*, Academic Press, 1989,

1577

ISBN: 0-12-128470-0.

1578

DeRemer and Pennello Article

1579

DeRemer, Frank and Pennello, Thomas J., *Efficient Computation of LALR(1) Look-Ahead Sets*,

1580

SigPlan Notices, Volume 15, No. 8, August 1979.

1581

Draft ANSI X3J11.1

1582

IEEE Floating Point draft report of ANSI X3J11.1 (NCEG).

1583

FIPS 151-1

1584

Federal Information Procurement Standard (FIPS) 151-1. Portable Operating System

1585

Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language].

1586

FIPS 151-2

1587

Federal Information Procurement Standards (FIPS) 151-2, Portable Operating System

1588

Interface (POSIX)— Part 1: System Application Program Interface (API) [C Language].

1589

HP-UX Manual

1590

Hewlett-Packard HP-UX Release 9.0 Reference Manual, Third Edition, August 1992.

1591

IEC 60559: 1989

1592

IEC 60559: 1989, Binary Floating-Point Arithmetic for Microprocessor Systems (previously

1593

designated IEC 559: 1989).

1594

IEEE Standards Terms

1595

IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.

Referenced Documents

- 1596 IEEE Std 754-1985
1597 IEEE Std 754-1985 (Reaff 1990), IEEE Standard for Binary Floating-Point Arithmetic.
- 1598 IEEE Std 854-1987
1599 IEEE Std 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic.
- 1600 IEEE Std 1003.9-1992
1601 IEEE Std 1003.9-1992, IEEE Standard for Information Technology — POSIX FORTRAN 77
1602 Language Interfaces — Part 1: Binding for System Application Program Interface API.
- 1603 IETF RFC 791
1604 Internet Protocol, Version 4 (IPv4), September 1981 (available at:
1605 www.ietf.org/rfc/rfc0791.txt).
- 1606 IETF RFC 819
1607 The Domain Naming Convention for Internet User Applications, Z. Su, J. Postel, August
1608 1982 (available at: www.ietf.org/rfc/rfc0819.txt).
- 1609 IETF RFC 822
1610 Standard for the Format of ARPA Internet Text Messages, D.H. Crocker, August 1982
1611 (available at: www.ietf.org/rfc/rfc0822.txt).
- 1612 IETF RFC 919
1613 Broadcasting Internet Datagrams, J. Mogul, October 1984 (available at:
1614 www.ietf.org/rfc/rfc0919.txt).
- 1615 IETF RFC 920
1616 Domain Requirements, J. Postel, J. Reynolds, October 1984 (available at:
1617 www.ietf.org/rfc/rfc0920.txt).
- 1618 IETF RFC 921
1619 Domain Name System Implementation Schedule, J. Postel, October 1984 (available at:
1620 www.ietf.org/rfc/rfc0921.txt).
- 1621 IETF RFC 922
1622 Broadcasting Internet Datagrams in the Presence of Subnets, J. Mogul, October 1984
1623 (available at: www.ietf.org/rfc/rfc0922.txt).
- 1624 IETF RFC 1034
1625 Domain Names — Concepts and Facilities, P. Mockapetris, November 1987 (available at:
1626 www.ietf.org/rfc/rfc1034.txt).
- 1627 IETF RFC 1035
1628 Domain Names — Implementation and Specification, P. Mockapetris, November 1987
1629 (available at: www.ietf.org/rfc/rfc1035.txt).
- 1630 IETF RFC 1123
1631 Requirements for Internet Hosts — Application and Support, R. Braden, October 1989
1632 (available at: www.ietf.org/rfc/rfc1123.txt).
- 1633 IETF RFC 1886
1634 DNS Extensions to Support Internet Protocol, Version 6 (IPv6), C. Huitema, S. Thomson,
1635 December 1995 (available at: www.ietf.org/rfc/rfc1886.txt).
- 1636 IETF RFC 2045
1637 Multipurpose Internet Mail Extensions (MIME), Part 1: Format of Internet Message Bodies,
1638 N. Freed, N. Borenstein, November 1996 (available at: www.ietf.org/rfc/rfc2045.txt).

- 1639 IETF RFC 2181
 1640 Clarifications to the DNS Specification, R. Elz, R. Bush, July 1997 (available at:
 1641 www.ietf.org/rfc/rfc2181.txt).
- 1642 IETF RFC 2373
 1643 Internet Protocol, Version 6 (IPv6) Addressing Architecture, S. Deering, R. Hinden, July 1998
 1644 (available at: www.ietf.org/rfc/rfc2373.txt).
- 1645 IETF RFC 2460
 1646 Internet Protocol, Version 6 (IPv6), S. Deering, R. Hinden, December 1998 (available at:
 1647 www.ietf.org/rfc/rfc2460.txt).
- 1648 Internationalisation Guide
 1649 Guide, July 1993, Internationalisation Guide, Version 2 (ISBN: 1-859120-02-4, G304),
 1650 published by The Open Group.
- 1651 ISO C (1990)
 1652 ISO/IEC 9899:1990, Programming Languages — C, including Amendment 1:1995 (E), C
 1653 Integrity (Multibyte Support Extensions (MSE) for ISO C).
- 1654 ISO 2375:1985
 1655 ISO 2375:1985, Data Processing — Procedure for Registration of Escape Sequences.
- 1656 ISO 8652:1987
 1657 ISO 8652:1987, Programming Languages — Ada (technically identical to ANSI standard
 1658 1815A-1983).
- 1659 ISO/IEC 1539:1990
 1660 ISO/IEC 1539:1990, Information Technology — Programming Languages — Fortran
 1661 (technically identical to the ANSI X3.9-1978 standard [FORTRAN 77]).
- 1662 ISO/IEC 4873:1991
 1663 ISO/IEC 4873:1991, Information Technology — ISO 8-bit Code for Information Interchange
 1664 — Structure and Rules for Implementation.
- 1665 ISO/IEC 6429:1992
 1666 ISO/IEC 6429:1992, Information Technology — Control Functions for Coded Character
 1667 Sets.
- 1668 ISO/IEC 6937:1994
 1669 ISO/IEC 6937:1994, Information Technology — Coded Character Set for Text
 1670 Communication — Latin Alphabet.
- 1671 ISO/IEC 8802-3:1996
 1672 ISO/IEC 8802-3:1996, Information Technology — Telecommunications and Information
 1673 Exchange Between Systems — Local and Metropolitan Area Networks — Specific
 1674 Requirements — Part 3: Carrier Sense Multiple Access with Collision Detection
 1675 (CSMA/CD) Access Method and Physical Layer Specifications.
- 1676 ISO/IEC 8859
 1677 ISO/IEC 8859, Information Technology — 8-Bit Single-Byte Coded Graphic Character Sets:
 1678 Part 1: Latin Alphabet No. 1
 1679 Part 2: Latin Alphabet No. 2
 1680 Part 3: Latin Alphabet No. 3
 1681 Part 4: Latin Alphabet No. 4
 1682 Part 5: Latin/Cyrillic Alphabet
 1683 Part 6: Latin/Arabic Alphabet
 1684 Part 7: Latin/Greek Alphabet

Referenced Documents

- 1685 Part 8: Latin/Hebrew Alphabet
 1686 Part 9: Latin Alphabet No. 5
 1687 Part 10: Latin Alphabet No. 6
 1688 Part 11: Latin/Thai Alphabet
 1689 Part 13: Latin Alphabet No. 7
 1690 Part 14: Latin Alphabet No. 8
 1691 Part 15: Latin Alphabet No. 9
 1692 Part 16: Latin Alphabet No. 10
- 1693 ISO POSIX-1: 1996
 1694 ISO/IEC 9945-1:1996, Information Technology — Portable Operating System Interface
 1695 (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to
 1696 ANSI/IEEE Std 1003.1-1996). Incorporating ANSI/IEEE Stds 1003.1-1990, 1003.1b-1993,
 1697 1003.1c-1995, and 1003.1i-1995.
- 1698 ISO POSIX-2: 1993
 1699 ISO/IEC 9945-2:1993, Information Technology — Portable Operating System Interface
 1700 (POSIX) — Part 2: Shell and Utilities (identical to ANSI/IEEE Std 1003.2-1992, as amended
 1701 by ANSI/IEEE Std 1003.2a-1992).
- 1702 Issue 1
 1703 X/Open Portability Guide, July 1985 (ISBN: 0-444-87839-4).
- 1704 Issue 2
 1705 X/Open Portability Guide, January 1987:
 1706 • Volume 1: XVS Commands and Utilities (ISBN: 0-444-70174-5)
 1707 • Volume 2: XVS System Calls and Libraries (ISBN: 0-444-70175-3)
- 1708 Issue 3
 1709 X/Open Specification, 1988, 1989, February 1992:
 1710 • Commands and Utilities, Issue 3 (ISBN: 1-872630-36-7, C211); this specification was
 1711 formerly X/Open Portability Guide, Issue 3, Volume 1, January 1989, XSI Commands
 1712 and Utilities (ISBN: 0-13-685835-X, XO/XPG/89/002)
 1713 • System Interfaces and Headers, Issue 3 (ISBN: 1-872630-37-5, C212); this specification
 1714 was formerly X/Open Portability Guide, Issue 3, Volume 2, January 1989, XSI System
 1715 Interface and Headers (ISBN: 0-13-685843-0, XO/XPG/89/003)
 1716 • Curses Interface, Issue 3, contained in Supplementary Definitions, Issue 3
 1717 (ISBN: 1-872630-38-3, C213), Chapters 9 to 14 inclusive; this specification was formerly
 1718 X/Open Portability Guide, Issue 3, Volume 3, January 1989, XSI Supplementary
 1719 Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
 1720 • Headers Interface, Issue 3, contained in Supplementary Definitions, Issue 3
 1721 (ISBN: 1-872630-38-3, C213), Chapter 19, Cpio and Tar Headers; this specification was
 1722 formerly X/Open Portability Guide Issue 3, Volume 3, January 1989, XSI
 1723 Supplementary Definitions (ISBN: 0-13-685850-3, XO/XPG/89/004)
- 1724 Issue 4
 1725 CAE Specification, July 1992, published by The Open Group:
 1726 • System Interface Definitions (XBD), Issue 4 (ISBN: 1-872630-46-4, C204)
 1727 • Commands and Utilities (XCU), Issue 4 (ISBN: 1-872630-48-0, C203)

- 1728 • System Interfaces and Headers (XSH), Issue 4 (ISBN: 1-872630-47-2, C202)
- 1729 Issue 4, Version 2
- 1730 CAE Specification, August 1994, published by The Open Group:
- 1731 • System Interface Definitions (XBD), Issue 4, Version 2 (ISBN: 1-85912-036-9, C434)
- 1732 • Commands and Utilities (XCU), Issue 4, Version 2 (ISBN: 1-85912-034-2, C436)
- 1733 • System Interfaces and Headers (XSH), Issue 4, Version 2 (ISBN: 1-85912-037-7, C435)
- 1734 Issue 5
- 1735 Technical Standard, February 1997, published by The Open Group:
- 1736 • System Interface Definitions (XBD), Issue 5 (ISBN: 1-85912-186-1, C605)
- 1737 • Commands and Utilities (XCU), Issue 5 (ISBN: 1-85912-191-8, C604)
- 1738 • System Interfaces and Headers (XSH), Issue 5 (ISBN: 1-85912-181-0, C606)
- 1739 Issue 6
- 1740 Technical Standard, April 2004, published by The Open Group:
- 1741 • Base Definitions (XBD), Issue 6 (ISBN: 1-931624-43-7, C046)
- 1742 • System Interfaces (XSH), Issue 6 (ISBN: 1-931624-44-5, C047)
- 1743 • Shell and Utilities (XCU), Issue 6 (ISBN: 1-931624-45-3, C048)
- 1744 Knuth Article
- 1745 Knuth, Donald E., *On the Translation of Languages from Left to Right*, Information and Control,
- 1746 Volume 8, No. 6, October 1965.
- 1747 KornShell
- 1748 Bolsky, Morris I. and Korn, David G., *The New KornShell Command and Programming*
- 1749 *Language*, March 1995, Prentice Hall.
- 1750 MSE Working Draft
- 1751 Working draft of ISO/IEC 9899:1990/Add3:Draft, Addendum 3 — Multibyte Support
- 1752 Extensions (MSE) as documented in the ISO Working Paper SC22/WG14/N205 dated 31
- 1753 March 1992.
- 1754 POSIX.0:1995
- 1755 IEEE Std 1003.0-1995, IEEE Guide to the POSIX Open System Environment (OSE) (identical
- 1756 to ISO/IEC TR 14252).
- 1757 POSIX.1:1988
- 1758 IEEE Std 1003.1-1988, IEEE Standard for Information Technology — Portable Operating
- 1759 System Interface (POSIX) — Part 1: System Application Program Interface (API) [C
- 1760 Language].
- 1761 POSIX.1:1990
- 1762 IEEE Std 1003.1-1990, IEEE Standard for Information Technology — Portable Operating
- 1763 System Interface (POSIX) — Part 1: System Application Program Interface (API) [C
- 1764 Language].
- 1765 POSIX.1a
- 1766 P1003.1a, Standard for Information Technology — Portable Operating System Interface
- 1767 (POSIX) — Part 1: System Application Program Interface (API) — (C Language)
- 1768 Amendment.

Referenced Documents

- 1769 POSIX.1d: 1999
 1770 IEEE Std 1003.1d-1999, IEEE Standard for Information Technology — Portable Operating
 1771 System Interface (POSIX) — Part 1: System Application Program Interface (API) —
 1772 Amendment 4: Additional Realtime Extensions [C Language].
- 1773 POSIX.1g: 2000
 1774 IEEE Std 1003.1g-2000, IEEE Standard for Information Technology — Portable Operating
 1775 System Interface (POSIX) — Part 1: System Application Program Interface (API) —
 1776 Amendment 6: Protocol-Independent Interfaces (PII).
- 1777 POSIX.1j: 2000
 1778 IEEE Std 1003.1j-2000, IEEE Standard for Information Technology — Portable Operating
 1779 System Interface (POSIX) — Part 1: System Application Program Interface (API) —
 1780 Amendment 5: Advanced Realtime Extensions [C Language].
- 1781 POSIX.1q: 2000
 1782 IEEE Std 1003.1q-2000, IEEE Standard for Information Technology — Portable Operating
 1783 System Interface (POSIX) — Part 1: System Application Program Interface (API) —
 1784 Amendment 7: Tracing [C Language].
- 1785 POSIX.2b
 1786 P1003.2b, Standard for Information Technology — Portable Operating System Interface
 1787 (POSIX) — Part 2: Shell and Utilities — Amendment.
- 1788 POSIX.2d:-1994
 1789 IEEE Std 1003.2d-1994, IEEE Standard for Information Technology — Portable Operating
 1790 System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: Batch Environment.
- 1791 POSIX.13:-1998
 1792 IEEE Std 1003.13:1998, IEEE Standard for Information Technology — Standardized
 1793 Application Environment Profile (AEP) — POSIX Realtime Application Support.
- 1794 Sarwate Article
 1795 Sarwate, Dilip V., *Computation of Cyclic Redundancy Checks via Table Lookup*, Communications
 1796 of the ACM, Volume 30, No. 8, August 1988.
- 1797 Sprunt, Sha, and Lehoczky
 1798 Sprunt, B., Sha, L., and Lehoczky, J.P., *Aperiodic Task Scheduling for Hard Real-Time Systems*,
 1799 *The Journal of Real-Time Systems*, Volume 1, 1989, Pages 27-60.
- 1800 SVID, Issue 1
 1801 American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue
 1802 1; Morristown, NJ, UNIX Press, 1985.
- 1803 SVID, Issue 2
 1804 American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue
 1805 2; Morristown, NJ, UNIX Press, 1986.
- 1806 SVID, Issue 3
 1807 American Telephone and Telegraph Company, System V Interface Definition (SVID), Issue
 1808 3; Morristown, NJ, UNIX Press, 1989.
- 1809 The AWK Programming Language
 1810 Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK Programming*
 1811 *Language*, Reading, MA, Addison-Wesley 1988.
- 1812 UNIX Programmer's Manual
 1813 American Telephone and Telegraph Company, *UNIX Time-Sharing System: UNIX*
 1814 *Programmer's Manual*, 7th Edition, Murray Hill, NJ, Bell Telephone Laboratories, January

- 1815 1979.
- 1816 XNS, Issue 4
- 1817 CAE Specification, August 1994, Networking Services, Issue 4 (ISBN: 1-85912-049-0, C438),
- 1818 published by The Open Group.
- 1819 XNS, Issue 5
- 1820 CAE Specification, February 1997, Networking Services, Issue 5 (ISBN: 1-85912-165-9, C523),
- 1821 published by The Open Group.
- 1822 XNS, Issue 5.2
- 1823 Technical Standard, January 2000, Networking Services (XNS), Issue 5.2
- 1824 (ISBN: 1-85912-241-8, C808), published by The Open Group.
- 1825 X/Open Curses, Issue 4, Version 2
- 1826 CAE Specification, May 1996, X/Open Curses, Issue 4, Version 2 (ISBN: 1-85912-171-3,
- 1827 C610), published by The Open Group.
- 1828 Yacc
- 1829 *Yacc: Yet Another Compiler Compiler*, Stephen C. Johnson, 1978.

1830 Source Documents

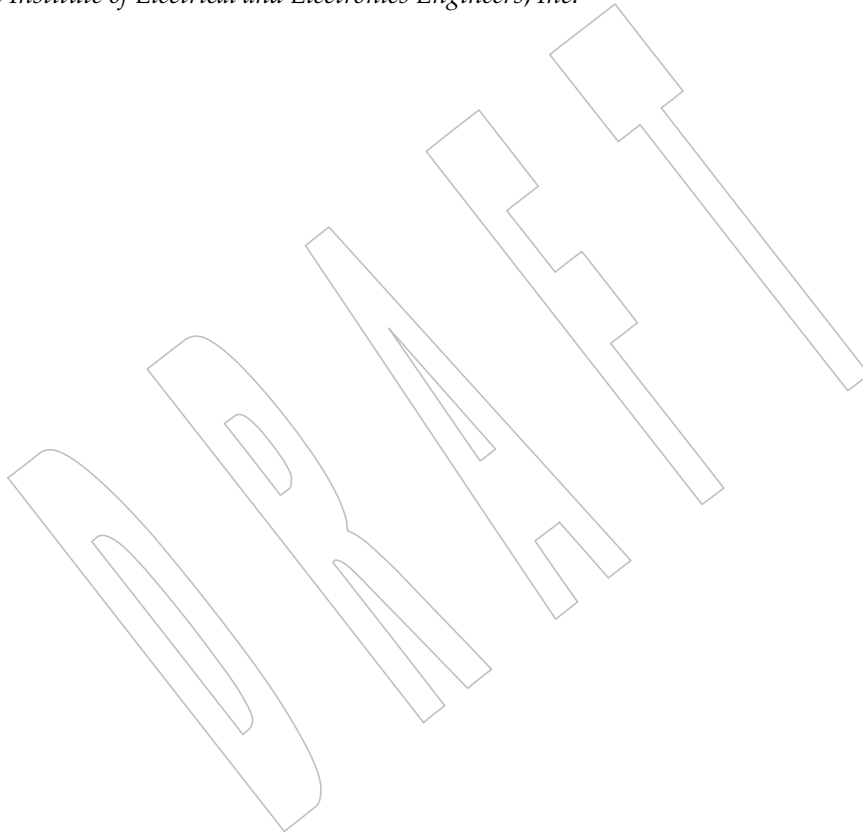
1831 Parts of the following documents were used to create the base documents for POSIX.1-200x:

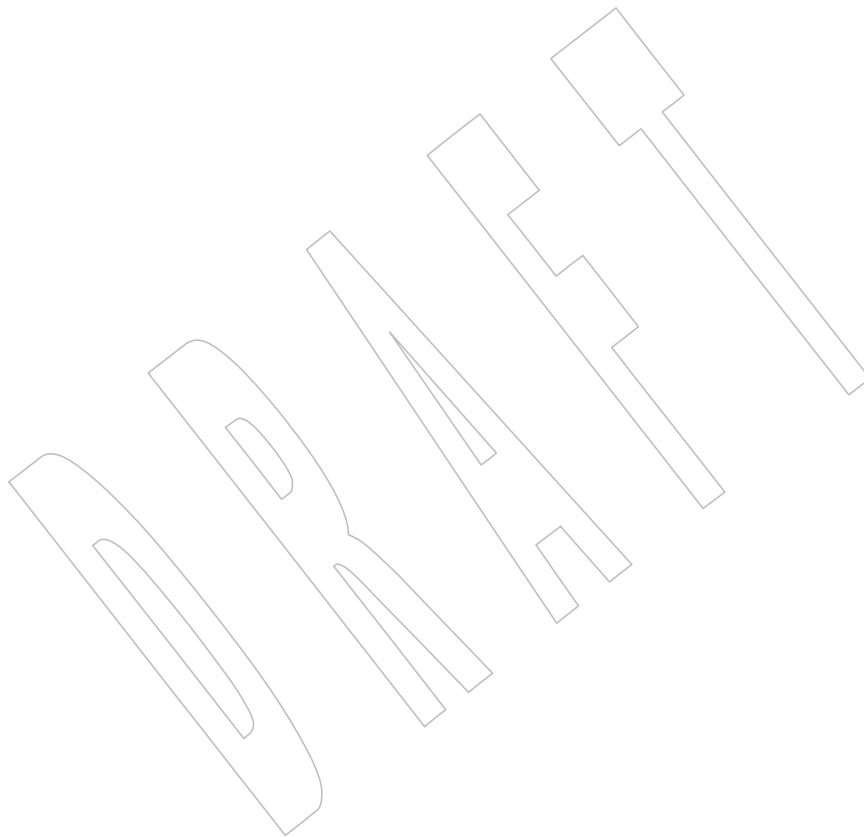
- 1832 AIX 3.2 Manual
- 1833 AIX Version 3.2 For RISC System/6000, Technical Reference: Base Operating System and
- 1834 Extensions, 1990, 1992 (Part No. SC23-2382-00).
- 1835 OSF/1
- 1836 OSF/1 Programmer's Reference, Release 1.2 (ISBN: 0-13-020579-6).
- 1837 OSF AES
- 1838 Application Environment Specification (AES) Operating System Programming Interfaces
- 1839 Volume, Revision A (ISBN: 0-13-043522-8).
- 1840 System V Release 2.0
- 1841 — UNIX System V Release 2.0 Programmer's Reference Manual (April 1984 - Issue 2).
- 1842 — UNIX System V Release 2.0 Programming Guide (April 1984 - Issue 2).
- 1843 System V Release 4.2
- 1844 Operating System API Reference, UNIX[®] SVR4.2 (1992) (ISBN: 0-13-017658-3).

Technical Standard

1 **Volume 1:**
2 **Base Definitions, Issue 7**

3 *The Open Group*
4 *The Institute of Electrical and Electronics Engineers, Inc.*





1.1 Scope

POSIX.1-200x defines a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs to support applications portability at the source code level. It is intended to be used by both application developers and system implementors.

POSIX.1-200x comprises four major components (each in an associated volume):

1. General terms, concepts, and interfaces common to all volumes of POSIX.1-200x, including utility conventions and C-language header definitions, are included in the Base Definitions volume of POSIX.1-200x.
2. Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume of POSIX.1-200x.
3. Definitions for a standard source code-level interface to command interpretation services (a “shell”) and common utility programs for application programs are included in the Shell and Utilities volume of POSIX.1-200x.
4. Extended rationale that did not fit well into the rest of the document structure, containing historical information concerning the contents of POSIX.1-200x and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume of POSIX.1-200x.

The following areas are outside of the scope of POSIX.1-200x:

- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-200x describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

The facilities provided in POSIX.1-200x are drawn from the following base documents:

- IEEE Std 1003.1, 2004 Edition (POSIX-1) (incorporating IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002, and IEEE Std 1003.1-2001/Cor 2-2004)

- 40 • The Open Group Technical Standard, 2006, Extended API Set Part 1
- 41 • The Open Group Technical Standard, 2006, Extended API Set Part 2
- 42 • The Open Group Technical Standard, 2006, Extended API Set Part 3
- 43 • The Open Group Technical Standard, 2006, Extended API Set Part 4
- 44 • ISO/IEC 9899:1999, Programming Languages — C (including ISO/IEC
- 45 9899:1999/Cor.1:2001(E) and ISO/IEC 9899:1999/Cor.2:2004(E))

46 Emphasis has been placed on standardizing existing practice for existing users, with changes
47 and additions limited to correcting deficiencies in the following areas:

- 48 • Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1,
49 and ISO/IEC defect reports against ISO/IEC 9945
- 50 • Issues raised in corrigenda for The Open Group Technical Standards and working group
51 resolutions from The Open Group
- 52 • Issues arising from ISO TR 24715:2006, Conflicts between POSIX and the LSB
- 53 • Changes to make the text self-consistent with the additional material merged
- 54 • Features, marked Legacy or obsolescent in the base documents, have been considered for
55 removal in this version
- 56 • A review and reorganization of the options within the standard
- 57 • Alignment with the ISO/IEC 9899:1999 standard, including Technical Corrigendum 2

58 1.2 Conformance

59 Conformance requirements for POSIX.1-200x are defined in [Chapter 2](#) (on page 15).

60 1.3 Normative References

61 The following standards contain provisions which, through references in POSIX.1-200x,
62 constitute provisions of POSIX.1-200x. At the time of publication, the editions indicated were
63 valid. All standards are subject to revision, and parties to agreements based on POSIX.1-200x are
64 encouraged to investigate the possibility of applying the most recent editions of the standards
65 listed below. Members of IEC and ISO maintain registers of currently valid International
66 Standards.

67 ANS X3.9-1978

68 (Reaffirmed 1989) American National Standard for Information Systems: Standard
69 X3.9-1978, Programming Language FORTRAN.¹

70 ISO/IEC 646:1991

71 ISO/IEC 646:1991, Information Processing — ISO 7-Bit Coded Character Set for
72 Information Interchange.²

73 1. ANSI documents can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY
74 10018, U.S.A.

75 2. ISO/IEC documents can be obtained from the ISO office: 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse

- 76 ISO 4217:2001
77 ISO 4217:2001, Codes for the Representation of Currencies and Funds.
- 78 ISO 8601:2000
79 ISO 8601:2000, Data Elements and Interchange Formats — Information Interchange —
80 Representation of Dates and Times.
- 81 ISO C (1999)
82 ISO/IEC 9899:1999, Programming Languages — C, including Technical Corrigendum 1 and
83 Technical Corrigendum 2.
- 84 ISO/IEC 10646-1:2000
85 ISO/IEC 10646-1:2000, Information Technology — Universal Multiple-Octet Coded
86 Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

87 1.4 Change History

88 Change history is described in the Rationale (Informative) volume of POSIX.1-200x, and in the
89 CHANGE HISTORY section of reference pages.

90 1.5 Terminology

91 For the purposes of POSIX.1-200x, the following terminology definitions apply:

92 **can**

93 Describes a permissible optional feature or behavior available to the user or application. The
94 feature or behavior is mandatory for an implementation that conforms to POSIX.1-200x. An
95 application can rely on the existence of the feature or behavior.

96 **implementation-defined**

97 Describes a value or behavior that is not defined by POSIX.1-200x but is selected by an
98 implementor. The value or behavior may vary among implementations that conform to
99 POSIX.1-200x. An application should not rely on the existence of the value or behavior. An
100 application that relies on such a value or behavior cannot be assured to be portable across
101 conforming implementations.

102 The implementor shall document such a value or behavior so that it can be used correctly
103 by an application.

104 **legacy**

105 Describes a feature or behavior that is being retained for compatibility with older
106 applications, but which has limitations which make it inappropriate for developing portable
107 applications. New applications should use alternative means of obtaining equivalent
108 functionality.

109 **may**

110 Describes a feature or behavior that is optional for an implementation that conforms to
111 POSIX.1-200x. An application should not rely on the existence of the feature or behavior. An
112 application that relies on such a feature or behavior cannot be assured to be portable across
113 conforming implementations.

114 To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

115 **shall**

116 For an implementation that conforms to POSIX.1-200x, describes a feature or behavior that
117 is mandatory. An application can rely on the existence of the feature or behavior.

118 For an application or user, describes a behavior that is mandatory. |

119 **should** |

120 For an implementation that conforms to POSIX.1-200x, describes a feature or behavior that |
121 is recommended but not mandatory. An application should not rely on the existence of the |
122 feature or behavior. An application that relies on such a feature or behavior cannot be |
123 assured to be portable across conforming implementations. |

124 For an application, describes a feature or behavior that is recommended programming |
125 practice for optimum portability. |

126 **undefined** |

127 Describes the nature of a value or behavior not defined by POSIX.1-200x which results from |
128 use of an invalid program construct or invalid data input. |

129 The value or behavior may vary among implementations that conform to POSIX.1-200x. An |
130 application should not rely on the existence or validity of the value or behavior. An |
131 application that relies on any particular value or behavior cannot be assured to be portable |
132 across conforming implementations. |

133 **unspecified** |

134 Describes the nature of a value or behavior not specified by POSIX.1-200x which results |
135 from use of a valid program construct or valid data input. |

136 The value or behavior may vary among implementations that conform to POSIX.1-200x. An |
137 application should not rely on the existence or validity of the value or behavior. An |
138 application that relies on any particular value or behavior cannot be assured to be portable |
139 across conforming implementations. |

140 1.6 Definitions and Concepts |

141 Definitions and concepts are defined in [Chapter 3](#) (on page 33) and [Chapter 4](#) (on page 95). |

142 1.7 Portability |

143 Some of the utilities in the Shell and Utilities volume of POSIX.1-200x and functions in the |
144 System Interfaces volume of POSIX.1-200x describe functionality that might not be fully portable |
145 to systems meeting the requirements for POSIX conformance (see [Chapter 2](#), on page 15). |

146 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in |
147 the margin identifies the nature of the option, extension, or warning (see [Section 1.7.1](#), on page |
148 7). For maximum portability, an application should avoid such functionality. |

149 Unless the primary task of a utility is to produce textual material on its standard output, |
150 application developers should not rely on the format or content of any such material that may be |
151 produced. Where the primary task *is* to provide such material, but the output format is |
152 incompletely specified, the description is marked with the OF margin code and shading. |
153 Application developers are warned not to expect that the output of such an interface on one |
154 system is any guide to its behavior on another system. |

1.7.1 Codes

The codes and their meanings are as follows. See also [Section 1.7.2](#) (on page 13).

ADV **Advisory Information**

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the ADV margin legend.

BE **Batch Environment Services and Utilities**

The functionality described is optional.

Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the BE margin legend.

CD **C-Language Development Utilities**

The functionality described is optional.

Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the CD margin legend.

CPT **Process CPU-Time Clocks**

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the CPT margin legend.

CX **Extension to the ISO C standard**

The functionality described is an extension to the ISO C standard. Application developers may make use of an extension as it is supported on all POSIX.1-200x-conforming systems.

With each function or header from the ISO C standard, a statement to the effect that “any conflict is unintentional” is included. That is intended to refer to a direct conflict. POSIX.1-200x acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors allowed to vary by the ISO C standard. Such limitations and other compatible differences are not considered conflicts, even if a CX mark is missing. The markings are for information only.

Where additional semantics apply to a function or header, the material is identified by use of the CX margin legend.

FD **FORTRAN Development Utilities**

The functionality described is optional.

Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the FD margin legend.

FR **FORTRAN Runtime Utilities**

The functionality described is optional.

Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the FR margin legend.

199	FSC	File Synchronization
200		The functionality described is optional. The functionality described is also an extension to the
201		ISO C standard.
202		Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section.
203		Where additional semantics apply to a function, the material is identified by use of the FSC
204		margin legend.
205	IP6	IPV6
206		The functionality described is optional. The functionality described is also an extension to the
207		ISO C standard.
208		Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section.
209		Where additional semantics apply to a function, the material is identified by use of the IP6
210		margin legend.
211	MC1	Non-Robust Mutex Priority Protection or Non-Robust Mutex Priority Inheritance or Robust
212		Mutex Priority Protection or Robust Mutex Priority Inheritance
213		The functionality described is optional. The functionality described is also an extension to the
214		ISO C standard.
215		This is a shorthand notation for combinations of multiple option codes.
216		Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section.
217		Where additional semantics apply to a function, the material is identified by use of the MC1
218		margin legend.
219		Refer to Section 1.7.2 (on page 13).
220	ML	Process Memory Locking
221		The functionality described is optional. The functionality described is also an extension to the
222		ISO C standard.
223		Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section.
224		Where additional semantics apply to a function, the material is identified by use of the ML
225		margin legend.
226	MLR	Range Memory Locking
227		The functionality described is optional. The functionality described is also an extension to the
228		ISO C standard.
229		Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section.
230		Where additional semantics apply to a function, the material is identified by use of the MLR
231		margin legend.
232	MON	Monotonic Clock
233		The functionality described is optional. The functionality described is also an extension to the
234		ISO C standard.
235		Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section.
236		Where additional semantics apply to a function, the material is identified by use of the MON
237		margin legend.
238	MSG	Message Passing
239		The functionality described is optional. The functionality described is also an extension to the
240		ISO C standard.
241		Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section.
242		Where additional semantics apply to a function, the material is identified by use of the MSG
243		margin legend.

244	MX	IEC 60559 Floating-Point
245		The functionality described is optional. The functionality described is also an extension to the
246		ISO C standard.
247		Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section.
248		Where additional semantics apply to a function, the material is identified by use of the MX
249		margin legend.
250	OB	Obsolescent
251		The functionality described may be removed in a future version of this volume of POSIX.1-200x.
252		Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use
253		obsolescent features.
254		Where applicable, the material is identified by use of the OB margin legend.
255	OF	Output Format Incompletely Specified
256		The functionality described is an XSI extension. The format of the output produced by the
257		utility is not fully specified. It is therefore not possible to post-process this output in a consistent
258		fashion. Typical problems include unknown length of strings and unspecified field delimiters.
259		Where applicable, the material is identified by use of the OF margin legend.
260	OH	Optional Header
261		In the SYNOPSIS section of some interfaces in the System Interfaces volume of POSIX.1-200x an
262		included header is marked as in the following example:
263	OH	<code>#include <sys/types.h></code>
264		<code>#include <grp.h></code>
265		<code>struct group *getgrnam(const char *name);</code>
266		The OH margin legend indicates that the marked header is not required on XSI-conformant
267		systems.
268	PIO	Prioritized Input and Output
269		The functionality described is optional. The functionality described is also an extension to the
270		ISO C standard.
271		Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section.
272		Where additional semantics apply to a function, the material is identified by use of the PIO
273		margin legend.
274	PS	Process Scheduling
275		The functionality described is optional. The functionality described is also an extension to the
276		ISO C standard.
277		Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section.
278		Where additional semantics apply to a function, the material is identified by use of the PS
279		margin legend.
280	RPI	Robust Mutex Priority Inheritance
281		The functionality described is optional. The functionality described is also an extension to the
282		ISO C standard.
283		Where applicable, functions are marked with the RPI margin legend in the SYNOPSIS section.
284		Where additional semantics apply to a function, the material is identified by use of the RPI
285		margin legend.
286	RPP	Robust Mutex Priority Protection
287		The functionality described is optional. The functionality described is also an extension to the
288		ISO C standard.

289		Where applicable, functions are marked with the RPP margin legend in the SYNOPSIS section.
290		Where additional semantics apply to a function, the material is identified by use of the RPP
291		margin legend.
292	RS	Raw Sockets
293		The functionality described is optional. The functionality described is also an extension to the
294		ISO C standard.
295		Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section.
296		Where additional semantics apply to a function, the material is identified by use of the RS
297		margin legend.
298	SD	Software Development Utilities
299		The functionality described is optional.
300		Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section.
301		Where additional semantics apply to a utility, the material is identified by use of the SD margin
302		legend.
303	SHM	Shared Memory Objects
304		The functionality described is optional. The functionality described is also an extension to the
305		ISO C standard.
306		Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section.
307		Where additional semantics apply to a function, the material is identified by use of the SHM
308		margin legend.
309	SIO	Synchronized Input and Output
310		The functionality described is optional. The functionality described is also an extension to the
311		ISO C standard.
312		Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section.
313		Where additional semantics apply to a function, the material is identified by use of the SIO
314		margin legend.
315	SPN	Spawn
316		The functionality described is optional. The functionality described is also an extension to the
317		ISO C standard.
318		Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section.
319		Where additional semantics apply to a function, the material is identified by use of the SPN
320		margin legend.
321	SS	Process Sporadic Server
322		The functionality described is optional. The functionality described is also an extension to the
323		ISO C standard.
324		Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section.
325		Where additional semantics apply to a function, the material is identified by use of the SS
326		margin legend.
327	TCT	Thread CPU-Time Clocks
328		The functionality described is optional. The functionality described is also an extension to the
329		ISO C standard.
330		Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section.
331		Where additional semantics apply to a function, the material is identified by use of the TCT
332		margin legend.

333	TEF	Trace Event Filter
334		The functionality described is optional. This functionality is dependent on support for the Trace
335		option. The functionality described is also an extension to the ISO C standard.
336		Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section.
337		Where additional semantics apply to a function, the material is identified by use of the TEF
338		margin legend.
339	TPI	Non-Robust Mutex Priority Inheritance
340		The functionality described is optional. The functionality described is also an extension to the
341		ISO C standard.
342		Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section.
343		Where additional semantics apply to a function, the material is identified by use of the TPI
344		margin legend.
345	TPP	Non-Robust Mutex Priority Protection
346		The functionality described is optional. The functionality described is also an extension to the
347		ISO C standard.
348		Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section.
349		Where additional semantics apply to a function, the material is identified by use of the TPP
350		margin legend.
351	TPS	Thread Execution Scheduling
352		The functionality described is optional. The functionality described is also an extension to the
353		ISO C standard.
354		Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section.
355		Where additional semantics apply to a function, the material is identified by use of the TPS
356		margin legend.
357	TRC	Trace
358		The functionality described is optional. The functionality described is also an extension to the
359		ISO C standard.
360		Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section.
361		Where additional semantics apply to a function, the material is identified by use of the TRC
362		margin legend.
363	TRI	Trace Inherit
364		The functionality described is optional. This functionality is dependent on support for the Trace
365		option. The functionality described is also an extension to the ISO C standard.
366		Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section.
367		Where additional semantics apply to a function, the material is identified by use of the TRI
368		margin legend.
369	TRL	Trace Log
370		The functionality described is optional. This functionality is dependent on support for the Trace
371		option. The functionality described is also an extension to the ISO C standard.
372		Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section.
373		Where additional semantics apply to a function, the material is identified by use of the TRL
374		margin legend.
375	TSA	Thread Stack Address Attribute
376		The functionality described is optional. The functionality described is also an extension to the
377		ISO C standard.

378		Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section.
379		Where additional semantics apply to a function, the material is identified by use of the TSA
380		margin legend.
381	TSH	Thread Process-Shared Synchronization
382		The functionality described is optional. The functionality described is also an extension to the
383		ISO C standard.
384		Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section.
385		Where additional semantics apply to a function, the material is identified by use of the TSH
386		margin legend.
387	TSP	Thread Sporadic Server
388		The functionality described is optional. The functionality described is also an extension to the
389		ISO C standard.
390		Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section.
391		Where additional semantics apply to a function, the material is identified by use of the TSP
392		margin legend.
393	TSS	Thread Stack Size Attribute
394		The functionality described is optional. The functionality described is also an extension to the
395		ISO C standard.
396		Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section.
397		Where additional semantics apply to a function, the material is identified by use of the TSS
398		margin legend.
399	TYM	Typed Memory Objects
400		The functionality described is optional. The functionality described is also an extension to the
401		ISO C standard.
402		Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section.
403		Where additional semantics apply to a function, the material is identified by use of the TYM
404		margin legend.
405	UP	User Portability Utilities
406		The functionality described is optional.
407		Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section.
408		Where additional semantics apply to a utility, the material is identified by use of the UP margin
409		legend.
410	UU	UUCP Utilities
411		The functionality described is optional. The functionality described is also an extension to the
412		ISO C standard.
413		Where applicable, functions are marked with the UU margin legend in the SYNOPSIS section.
414		Where additional semantics apply to a function, the material is identified by use of the UU
415		margin legend.
416	XSI	X/Open System Interfaces
417		The functionality described is part of the X/Open Systems Interfaces option. Functionality
418		marked XSI is an extension to the ISO C standard. Application developers may confidently
419		make use of such extensions on all systems supporting the X/Open System Interfaces option.
420		If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that
421		reference page is an extension. See Section 2.1.4 (on page 19).

422 XSR **XSI STREAMS**
 423 The functionality described is optional. The functionality described is also an extension to the
 424 ISO C standard.

425 Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section.
 426 Where additional semantics apply to a function, the material is identified by use of the XSR
 427 margin legend.

428 1.7.2 Margin Code Notation

429 Some of the functionality described in POSIX.1-200x depends on support of more than one
 430 option, or independently may depend on several options. The following notation for margin
 431 codes is used to denote the following cases.

432 A Feature Dependent on One or Two Options

433 In this case, margin codes have a <space> separator; for example:

434 SHM This feature requires support for only the Shared Memory Objects option.

435 SHM TYM This feature requires support for both the Shared Memory Objects option and the Typed
 436 Memory Objects option; that is, an application which uses this feature is portable only between
 437 implementations that provide both options.

438 A Feature Dependent on Either of the Options Denoted

439 In this case, margin codes have a ' | ' separator to denote the logical OR; for example:

440 SHM | TYM This feature is dependent on support for either the Shared Memory Objects option or the Typed
 441 Memory Objects option; that is, an application which uses this feature is portable between
 442 implementations that provide any (or all) of the options.

443 A Feature Dependent on More than Two Options

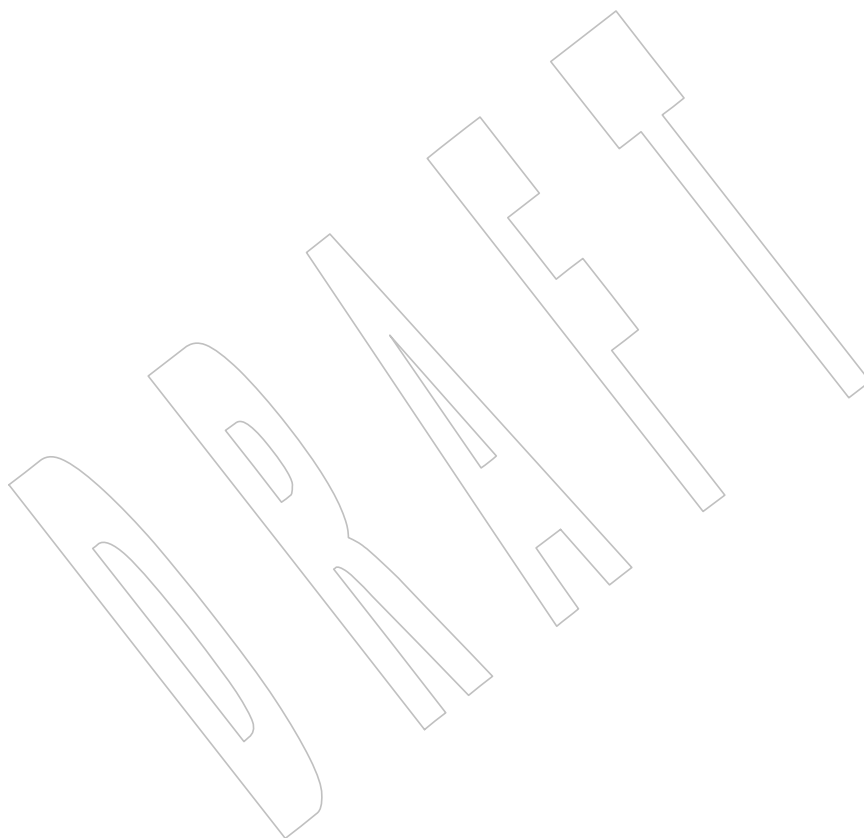
444 The following shorthand notations are used:

445 MC1 The MC1 margin code is shorthand for TPP | TPI | RPP | RPI. Features which are shaded with this
 446 margin code require support of either the Non-Robust Mutex Priority Protection option or the
 447 Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection option or
 448 the Robust Mutex Priority Inheritance option.

449 Large Sections Dependent on an Option

450 Where large sections of text are dependent on support for an option, a lead-in text block is
 451 provided and shaded accordingly; for example:

452 XSI This section describes extensions to support interprocess communication. The functionality
 453 described in this section shall be provided on implementations that support the XSI option (and
 454 the rest of this section is not further shaded).



2.1 Implementation Conformance

For the purposes of POSIX.1-200x, the implementation conformance requirements given in this section apply.

2.1.1 Requirements

A *conforming implementation* shall meet all of the following criteria:

1. The system shall support all utilities, functions, and facilities defined within POSIX.1-200x that are required for POSIX conformance (see [Section 2.1.3](#), on page 16). These interfaces shall support the functional behavior described herein.
2. The system may support the X/Open System Interfaces (XSI) option as described in [Section 2.1.4](#) (on page 19).
3. The system may support one or more options as described under [Section 2.1.5](#) (on page 20). When an implementation claims that an option is supported, all of its constituent parts shall be provided.
4. The system may provide additional utilities, functions, or facilities not required by POSIX.1-200x. Non-standard extensions of the utilities, functions, or facilities specified in POSIX.1-200x should be identified as such in the system documentation. Non-standard extensions, when used, may change the behavior of utilities, functions, or facilities defined by POSIX.1-200x. The conformance document shall define an environment in which an application can be run with the behavior specified by POSIX.1-200x. In no case shall such an environment require modification of a Strictly Conforming POSIX Application (see [Section 2.2.1](#), on page 28).

2.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to POSIX.1-200x. The conformance document shall have the same structure as POSIX.1-200x, with the information presented in the appropriate sections and subsections. Sections and subsections that consist solely of subordinate section titles, with no other information, are not required. The conformance document shall not contain information about extended facilities or capabilities outside the scope of POSIX.1-200x.

The conformance document shall contain a statement that indicates the full name, number, and date of the standard that applies. The conformance document may also list international software standards that are available for use by a Conforming POSIX Application. Applicable characteristics where documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the limit values found in the headers [<limits.h>](#) (on page 253) and [<unistd.h>](#) (on page 412), stating values, the conditions under which those values may change, and the limits of such variations, if any.

493 The conformance document shall describe the behavior of the implementation for all
 494 implementation-defined features defined in POSIX.1-200x. This requirement shall be met by
 495 listing these features and providing either a specific reference to the system documentation or
 496 providing full syntax and semantics of these features. When the value or behavior in the
 497 implementation is designed to be variable or customized on each instantiation of the system, the
 498 implementation provider shall document the nature and permissible ranges of this variation.

499 The conformance document may specify the behavior of the implementation for those features
 500 where POSIX.1-200x states that implementations may vary or where features are identified as
 501 undefined or unspecified.

502 The conformance document shall not contain documentation other than that specified in the
 503 preceding paragraphs except where such documentation is specifically allowed or required by
 504 other provisions of POSIX.1-200x.

505 The phrases “shall document” or “shall be documented” in POSIX.1-200x mean that
 506 documentation of the feature shall appear in the conformance document, as described
 507 previously, unless there is an explicit reference in the conformance document to show where the
 508 information can be found in the system documentation.

509 The system documentation should also contain the information found in the conformance
 510 document.

511 2.1.3 POSIX Conformance

512 A conforming implementation shall meet the following criteria for POSIX conformance.

513 2.1.3.1 POSIX System Interfaces

514 The following requirements apply to the system interfaces (functions and headers):

- 515 • The system shall support all the mandatory functions and headers defined in
 516 POSIX.1-200x, and shall set the symbolic constant `_POSIX_VERSION` to the value `200xxxL`.
- 517 • Although all implementations conforming to POSIX.1-200x support all the features
 518 described below, there may be system-dependent or file system-dependent configuration
 519 procedures that can remove or modify any or all of these features. Such configurations
 520 should not be made if strict compliance is required.

521 The following symbolic constants shall be defined with a value other than `-1`. If a constant
 522 is defined with the value zero, applications should use the `sysconf()`, `pathconf()`, or
 523 `fpathconf()` functions, or the `getconf` utility, to determine which features are present on the
 524 system at that time or for the particular pathname in question.

525 — `_POSIX_CHOWN_RESTRICTED`

526 The use of `chown()` is restricted to a process with appropriate privileges, and to
 527 changing the group ID of a file only to the effective group ID of the process or to one
 528 of its supplementary group IDs.

529 — `_POSIX_NO_TRUNC`

530 Pathname components longer than `{NAME_MAX}` generate an error.

- 531 • The following symbolic constants shall be defined by the implementation as follows:
 - 532 • Symbolic constants defined with the value `200xxxL`:

533 _POSIX_ASYNCHRONOUS_IO
 534 _POSIX_BARRIERS
 535 _POSIX_CLOCK_SELECTION
 536 _POSIX_MAPPED_FILES
 537 _POSIX_MEMORY_PROTECTION
 538 _POSIX_READER_WRITER_LOCKS
 539 _POSIX_REALTIME_SIGNALS
 540 _POSIX_SEMAPHORES
 541 _POSIX_SPIN_LOCKS
 542 _POSIX_THREAD_SAFE_FUNCTIONS
 543 _POSIX_THREADS
 544 _POSIX_TIMEOUTS
 545 _POSIX_TIMERS

- Symbolic constants defined with a value greater than zero:

547 _POSIX_JOB_CONTROL
 548 _POSIX_SAVED_IDS

- Symbolic constants defined with a value other than -1.

550 _POSIX_VDISABLE

551 **Note:** The symbols above represent historical options that are no longer allowed as options, but
 552 are retained here for backwards-compatibility of applications.

- The system may support one or more options (see [Section 2.1.6](#), on page 25) denoted by the following symbolic constants:

555 — _POSIX_ADVISORY_INFO
 556 — _POSIX_CPUTIME
 557 — _POSIX_FSYNC
 558 — _POSIX_IPV6
 559 — _POSIX_MEMLOCK
 560 — _POSIX_MEMLOCK_RANGE
 561 — _POSIX_MESSAGE_PASSING
 562 — _POSIX_MONOTONIC_CLOCK
 563 — _POSIX_PRIORITIZED_IO
 564 — _POSIX_PRIORITY_SCHEDULING
 565 — _POSIX_RAW_SOCKETS
 566 — _POSIX_SHARED_MEMORY_OBJECTS
 567 — _POSIX_SPAWN
 568 — _POSIX_SPORADIC_SERVER
 569 — _POSIX_SYNCHRONIZED_IO
 570 — _POSIX_THREAD_ATTR_STACKADDR

571 — _POSIX_THREAD_CPUTIME
 572 — _POSIX_THREAD_ATTR_STACKSIZE
 573 — _POSIX_THREAD_PRIO_INHERIT
 574 — _POSIX_THREAD_PRIO_PROTECT
 575 — _POSIX_THREAD_PRIORITY_SCHEDULING
 576 — _POSIX_THREAD_PROCESS_SHARED
 577 — _POSIX_THREAD_SPORADIC_SERVER
 578 — _POSIX_TRACE
 579 — _POSIX_TRACE_EVENT_FILTER
 580 — _POSIX_TRACE_INHERIT
 581 — _POSIX_TRACE_LOG
 582 — _POSIX_TYPED_MEMORY_OBJECTS
 583 — _XOPEN_CRYPT
 584 — _XOPEN_REALTIME
 585 — _XOPEN_REALTIME_THREADS
 586 — _XOPEN_STREAMS
 587 — _XOPEN_UNIX

588 If any of the symbolic constants `_POSIX_TRACE_EVENT_FILTER`, `_POSIX_TRACE_LOG`,
 589 or `_POSIX_TRACE_INHERIT` is defined to have a value other than `-1`, then the symbolic
 590 constant `_POSIX_TRACE` shall also be defined to have a value other than `-1`.

591 If the Advisory Information option is supported, there shall be at least one file system that
 592 supports the functionality.

593 2.1.3.2 *POSIX Shell and Utilities*

594 The following requirements apply to the shell and utilities:

- 595 • The system shall provide all the mandatory utilities in the Shell and Utilities volume of
 596 POSIX.1-200x with all the functional behavior described therein.
- 597 • The system shall support the Large File capabilities described in the Shell and Utilities
 598 volume of POSIX.1-200x.
- 599 • The system may support one or more options (see [Section 2.1.6](#), on page 25) denoted by the
 600 following symbolic constants. (The literal names below apply to the *getconf* utility.)

601 — POSIX2_C_DEV
 602 — POSIX2_CHAR_TERM
 603 — POSIX2_FORT_DEV
 604 — POSIX2_FORT_RUN
 605 — POSIX2_LOCALEDEF
 606 — POSIX2_PBS

- 607 — POSIX2_PBS_ACCOUNTING
- 608 — POSIX2_PBS_LOCATE
- 609 — POSIX2_PBS_MESSAGE
- 610 — POSIX2_PBS_TRACK
- 611 — POSIX2_SW_DEV
- 612 — POSIX2_UPE
- 613 — XOPEN_UNIX
- 614 — XOPEN_UUCP

615 Additional language bindings and development utility options may be provided in other related |
 616 standards or in a future version of this standard. In the former case, additional symbolic |
 617 constants of the same general form as shown in this subsection should be defined by the related |
 618 standard document and made available to the application without requiring POSIX.1-200x to be |
 619 updated.

620 2.1.4 XSI Conformance

621 XSI This section describes the criteria for implementations providing conformance to the X/Open
 622 System Interfaces (XSI) option (see [Section 3.442](#), on page 94). The functionality described in this
 623 section shall be provided on implementations that support the XSI option (and the rest of this
 624 section is not further shaded).

625 POSIX.1-200x describes utilities, functions, and facilities offered to application programs by the
 626 X/Open System Interfaces (XSI) option. An XSI-conforming implementation shall meet the
 627 criteria for POSIX conformance and the following requirements listed in this section.

628 XSI-conforming implementations shall set the symbolic constant `_XOPEN_UNIX` to a value
 629 other than `-1` and shall set the symbolic constant `_XOPEN_VERSION` to the value 700.

630 2.1.4.1 XSI System Interfaces

631 The following requirements apply to the system interfaces when the XSI option is supported:

- 632 • The system shall support all the functions and headers defined in POSIX.1-200x as part of
 633 the XSI option denoted by the XSI marking in the SYNOPSIS section, and any extensions
 634 marked with the XSI option marking (see [Section 1.7.1](#), on page 7) within the text.
- 635 • The system shall support the following options defined within POSIX.1-200x (see [Section](#) -
 636 [2.1.6](#), on page 25):
 - 637 — `_POSIX_FSYNC`
 - 638 — `_POSIX_THREAD_ATTR_STACKADDR`
 - 639 — `_POSIX_THREAD_ATTR_STACKSIZE`
 - 640 — `_POSIX_THREAD_PROCESS_SHARED`
- 641 • The system may support the following XSI Option Groups (see [Section 2.1.5.2](#), on page 21)
 642 defined within POSIX.1-200x:
 - 643 — Encryption
 - 644 — Realtime

- 645 — Advanced Realtime
- 646 — Realtime Threads
- 647 — Advanced Realtime Threads
- 648 — Tracing
- 649 — XSI STREAMS

650 2.1.4.2 XSI Shell and Utilities Conformance

651 The following requirements apply to the shell and utilities when the XSI option is supported:

- 652 • The system shall support all the utilities defined in the Shell and Utilities volume of
653 POSIX.1-200x as part of the XSI option denoted by the XSI marking in the SYNOPSIS
654 section, and any extensions marked with the XSI option marking (see [Section 1.7.1](#), on
655 page 7) within the text.
- 656 • The system shall support the User Portability Utilities option.
- 657 • The system shall support creation of locales (see [Chapter 7](#), on page 121).
- 658 • The C-language Development utility *c99* shall be supported.
- 659 • The XSI Development Utilities option may be supported. It consists of the following
660 software development utilities:

661	<i>admin</i>	<i>delta</i>	<i>rmdel</i>	<i>val</i>
662	<i>cflow</i>	<i>get</i>	<i>sact</i>	<i>what</i>
663	<i>ctags</i>	<i>nm</i>	<i>sccs</i>	
664	<i>cxref</i>	<i>prs</i>	<i>unget</i>	

665 2.1.5 Option Groups

666 An Option Group is a group of related functions or options defined within the System Interfaces
667 volume of POSIX.1-200x.

668 If an implementation supports an Option Group, then the system shall support the functional
669 behavior described herein.

670 If an implementation does not support an Option Group, then the system need not support the
671 functional behavior described herein.

672 2.1.5.1 Subprofiling Considerations

673 Profiling standards supporting functional requirements less than that required in POSIX.1-200x
674 may subset both mandatory and optional functionality required for POSIX Conformance (see
675 [Section 2.1.3](#), on page 16) or XSI Conformance (see [Section 2.1.4](#), on page 19). Such profiles shall
676 organize the subsets into Subprofiling Option Groups.

677 XRAT [Appendix E](#) (on page 3607) describes a representative set of such Subprofiling Option
678 Groups for use by profiles applicable to specialized realtime systems. POSIX.1-200x does not
679 require that the presence of Subprofiling Option Groups be testable at compile-time (as symbols
680 defined in any header) or at runtime (via *sysconf()* or *getconf()*).

681 A Subprofiling Option Group may provide basic system functionality that other Subprofiling
682 Option Groups and other options depend upon.³ If a profile of POSIX.1-200x does not require an
683 implementation to provide a Subprofiling Option Group that provides features utilized by a
684 required Subprofiling Option Group (or option),⁴ the profile shall specify⁵ all of the following:

- 685 • Restricted or altered behavior of interfaces defined in POSIX.1-200x that may differ on an
- 686 implementation of the profile
- 687 • Additional behaviors that may produce undefined or unspecified results
- 688 • Additional implementation-defined behavior that implementations shall be required to
- 689 document in the profile's conformance document

690 if any of the above is a result of the profile not requiring an interface required by POSIX.1-200x.

691 The following additional rules shall apply to all profiles of POSIX.1-200x:

- 692 • Any application that conforms to that profile shall also conform to POSIX.1-200x (that is, a
- 693 profile shall not require restricted, altered, or extended behaviors of an implementation of
- 694 POSIX.1-200x).
- 695 • Profiles are permitted to add additional requirements to the limits defined in `<limits.h>`
- 696 and `<stdint.h>`, subject to the following:

697 For the limits in `<limits.h>` and `<stdint.h>`:

- 698 — If the limit is specified as having a fixed value, it shall not be changed by a profile.
- 699 — If a limit is specified as having a minimum or maximum acceptable value, it may be
- 700 changed by a profile as follows:
 - 701 — A profile may increase a minimum acceptable value, but shall not make a
 - 702 minimum acceptable value smaller.
 - 703 — A profile may reduce a maximum acceptable value, but shall not make a
 - 704 maximum acceptable value larger.
- 705 • A profile shall not change a limit specified as having a minimum or maximum value into a
- 706 limit specified as having a fixed value.
- 707 • A profile shall not create new limits.
- 708 • Any implementation that conforms to POSIX.1-200x (including all options and extended
- 709 limits required by the profile) shall also conform to that profile.

710 2.1.5.2 XSI Option Groups

711 XSI This section describes Option Groups to support the definition of XSI conformance within the
 712 System Interfaces volume of POSIX.1-200x. The functionality described in this section shall be
 713 provided on implementations that support the XSI option and the appropriate Option Group
 714 (and the rest of this section is not further shaded).

715 The following Option Groups are defined.

-
- 716 3. As an example, the File System profiling option group provides underlying support for pathname resolution and file creation which are
 - 717 needed by any interface in POSIX.1-200x that parses a *path* argument. If a profile requires support for the Device Input and Output
 - 718 profiling option group but does not require support for the File System profiling option group, the profile must specify how pathname
 - 719 resolution is to behave in that profile, how the `O_CREAT` flag to `open()` is to be handled (and the use of the character 'a' in the *mode*
 - 720 argument of `open()` when a filename argument names a file that does not exist), and specify lots of other details.
 - 721 4. As an example, POSIX.1-200x requires that implementations claiming to support the Range Memory Locking option also support the
 - 722 Process Memory Locking option. A profile could require that the Range Memory Locking option had to be supplied without requiring that
 - 723 the Process Memory Locking option be supplied as long as the profile specifies everything an application developer or system implementor
 - 724 would have to know to build an application or implementation conforming to the profile.
 - 725 5. Note that the profile could just specify that any use of the features not specified by the profile would produce undefined or unspecified
 - 726 results.

727 **Encryption**

728 The Encryption Option Group is denoted by the symbolic constant `_XOPEN_CRYPT`. It includes
729 the following functions:

730 `crypt()`, `encrypt()`, `setkey()`

731 These functions are marked CRYPT.

732 Due to export restrictions on the decoding algorithm in some countries, implementations may
733 be restricted in making these functions available. All the functions in the Encryption Option
734 Group may therefore return [ENOSYS] or, alternatively, `encrypt()` shall return [ENOSYS] for the
735 decryption operation.

736 An implementation that claims conformance to this Option Group shall set `_XOPEN_CRYPT` to
737 a value other than `-1`.

738 **Realtime**

739 The Realtime Option Group is denoted by the symbolic constant `_XOPEN_REALTIME`.

740 This Option Group includes a set of realtime functions drawn from options within POSIX.1-200x
741 (see [Section 2.1.6](#), on page 25).

742 Where entire functions are included in the Option Group, the NAME section is marked with
743 REALTIME. Where additional semantics have been added to existing pages, the new material is
744 identified by use of the appropriate margin legend for the underlying option defined within
745 POSIX.1-200x.

746 An implementation that claims conformance to this Option Group shall set
747 `_XOPEN_REALTIME` to a value other than `-1`.

748 This Option Group consists of the set of the following options from within POSIX.1-200x (see
749 [Section 2.1.6](#), on page 25):

750 `_POSIX_FSYNC`
751 `_POSIX_MEMLOCK`
752 `_POSIX_MEMLOCK_RANGE`
753 `_POSIX_MESSAGE_PASSING`
754 `_POSIX_PRIORITIZED_IO`
755 `_POSIX_PRIORITY_SCHEDULING`
756 `_POSIX_SHARED_MEMORY_OBJECTS`
757 `_POSIX_SYNCHRONIZED_IO`

758 If the symbolic constant `_XOPEN_REALTIME` is defined to have a value other than `-1`, then the
759 following symbolic constants shall be defined by the implementation to have the value `200xxxL`:

760 `_POSIX_MEMLOCK`
761 `_POSIX_MEMLOCK_RANGE`
762 `_POSIX_MESSAGE_PASSING`
763 `_POSIX_PRIORITY_SCHEDULING`
764 `_POSIX_SHARED_MEMORY_OBJECTS`
765 `_POSIX_SYNCHRONIZED_IO`

766 The functionality associated with `_POSIX_FSYNC` shall always be supported on XSI-conformant
767 systems.

768 Support of `_POSIX_PRIORITIZED_IO` on XSI-conformant systems is optional. If
769 `_POSIX_PRIORITIZED_IO` is supported, then asynchronous I/O operations performed by

770 *aioread()*, *aiowrite()*, and *lio_listio()* shall be submitted at a priority equal to the scheduling
 771 priority equal to a base scheduling priority minus *aiocbp->aioreqprio*. If Thread Execution
 772 Scheduling is not supported, then the base scheduling priority is that of the calling process;
 773 otherwise, the base scheduling priority is that of the calling thread. The implementation shall
 774 also document for which files I/O prioritization is supported.

775 **Advanced Realtime**

776 An implementation that claims conformance to this Option Group shall also support the
 777 Realtime Option Group.

778 Where entire functions are included in the Option Group, the NAME section is marked with
 779 ADVANCED REALTIME. Where additional semantics have been added to existing pages, the
 780 new material is identified by use of the appropriate margin legend for the underlying option
 781 defined within POSIX.1-200x.

782 This Option Group consists of the set of the following options from within POSIX.1-200x (see
 783 [Section 2.1.6](#), on page 25):

```
784     _POSIX_ADVISORY_INFO
785     _POSIX_CPUTIME
786     _POSIX_MONOTONIC_CLOCK
787     _POSIX_SPAWN
788     _POSIX_SPORADIC_SERVER
789     _POSIX_TYPED_MEMORY_OBJECTS
```

790 If the implementation supports the Advanced Realtime Option Group, then the following
 791 symbolic constants shall be defined by the implementation to have the value 200xxxL:

```
792     _POSIX_ADVISORY_INFO
793     _POSIX_CPUTIME
794     _POSIX_MONOTONIC_CLOCK
795     _POSIX_SPAWN
796     _POSIX_SPORADIC_SERVER
797     _POSIX_TYPED_MEMORY_OBJECTS
```

798 If the symbolic constant `_POSIX_SPORADIC_SERVER` is defined, then the symbolic constant
 799 `_POSIX_PRIORITY_SCHEDULING` shall also be defined by the implementation to have the
 800 value 200xxxL.

801 **Realtime Threads**

802 The Realtime Threads Option Group is denoted by the symbolic constant
 803 `_XOPEN_REALTIME_THREADS`.

804 This Option Group consists of the set of the following options from within POSIX.1-200x (see
 805 [Section 2.1.6](#), on page 25):

```
806     _POSIX_THREAD_PRIO_INHERIT
807     _POSIX_THREAD_PRIO_PROTECT
808     _POSIX_THREAD_PRIORITY_SCHEDULING
```

809 Where applicable, whole pages are marked REALTIME THREADS, together with the
 810 appropriate option margin legend for the SYNOPSIS section (see [Section 1.7.1](#), on page 7).

811 An implementation that claims conformance to this Option Group shall set
 812 `_XOPEN_REALTIME_THREADS` to a value other than -1.

813 If the symbol `_XOPEN_REALTIME_THREADS` is defined to have a value other than `-1`, then the
 814 following options shall also be defined by the implementation to have the value `200xxxL`:

```
815     _POSIX_THREAD_PRIO_INHERIT
816     _POSIX_THREAD_PRIO_PROTECT
817     _POSIX_THREAD_PRIORITY_SCHEDULING
```

818 **Advanced Realtime Threads**

819 An implementation that claims conformance to this Option Group shall also support the
 820 Realtime Threads Option Group.

821 Where entire functions are included in the Option Group, the NAME section is marked with
 822 `ADVANCED_REALTIME_THREADS`. Where additional semantics have been added to existing
 823 pages, the new material is identified by use of the appropriate margin legend for the underlying
 824 option defined within `POSIX.1-200x`.

825 This Option Group consists of the set of the following options from within `POSIX.1-200x` (see
 826 [Section 2.1.6](#), on page 25):

```
827     _POSIX_THREAD_CPUTIME
828     _POSIX_THREAD_SPORADIC_SERVER
```

829 If the symbolic constant `_POSIX_THREAD_SPORADIC_SERVER` is defined to have the value
 830 `200xxxL`, then the symbolic constant `_POSIX_THREAD_PRIORITY_SCHEDULING` shall also be
 831 defined by the implementation to have the value `200xxxL`.

832 If the implementation supports the Advanced Realtime Threads Option Group, then the
 833 following symbolic constants shall be defined by the implementation to have the value `200xxxL`:

```
834     _POSIX_THREAD_CPUTIME
835     _POSIX_THREAD_SPORADIC_SERVER
```

836 **Tracing**

837 This Option Group includes a set of tracing functions drawn from options within `POSIX.1-200x`
 838 (see [Section 2.1.6](#), on page 25).

839 Where entire functions are included in the Option Group, the NAME section is marked with
 840 `TRACING`. Where additional semantics have been added to existing pages, the new material is
 841 identified by use of the appropriate margin legend for the underlying option defined within
 842 `POSIX.1-200x`.

843 This Option Group consists of the set of the following options from within `POSIX.1-200x` (see
 844 [Section 2.1.6](#), on page 25):

```
845     _POSIX_TRACE
846     _POSIX_TRACE_EVENT_FILTER
847     _POSIX_TRACE_LOG
848     _POSIX_TRACE_INHERIT
```

849 If the implementation supports the Tracing Option Group, then the following symbolic
 850 constants shall be defined by the implementation to have the value `200xxxL`:

851 _POSIX_TRACE
 852 _POSIX_TRACE_EVENT_FILTER
 853 _POSIX_TRACE_LOG
 854 _POSIX_TRACE_INHERIT

855 XSI STREAMS

856 OB XSR This section describes the XSI STREAMS Option Group, denoted by the symbolic constant
 857 _XOPEN_STREAMS. The functionality described in this section shall be provided on
 858 implementations that support the XSI STREAMS option (and the rest of this section is not
 859 further shaded).

860 This Option Group includes functionality related to STREAMS, a uniform mechanism for
 861 implementing networking services and other character-based I/O as described in XSH [Section](#)
 862 2.6 (on page 473).

863 It includes the following functions:

864	<i>fattach()</i>	<i>ioctl()</i>
865	<i>fdetach()</i>	<i>isastream()</i>
866	<i>getmsg()</i>	<i>putmsg()</i>
867	<i>getpmsg()</i>	<i>putpmsg()</i>

868 and the `<stropts.h>` header.

869 Where applicable, whole pages are marked STREAMS, together with the appropriate option
 870 margin legend for the SYNOPSIS section (see [Section 1.7.1](#), on page 7). Where additional
 871 semantics have been added to existing pages, the new material is identified by use of the
 872 appropriate margin legend for the underlying option defined within POSIX.1-200x.

873 An implementation that claims conformance to this Option Group shall set `_XOPEN_STREAMS`
 874 to a value other than `-1`.

875 2.1.6 Options

876 The symbolic constants defined in `<unistd.h>`, [Constants for Options and Option Groups](#) (on
 877 page 412) reflect implementation options for POSIX.1-200x. These symbols can be used by the
 878 application to determine which of three categories of support for optional facilities are provided
 879 by the implementation.

- 880 1. Option not supported for compilation.

881 The implementation advertises at compile time (by defining the constant in `<unistd.h>`
 882 with value `-1`, or by leaving it undefined) that the option is not supported for compilation
 883 and, at the time of compilation, is not supported for runtime use. In this case, the headers,
 884 data types, function interfaces, and utilities required only for the option need not be
 885 present. A later runtime check using the *fpathconf()*, *pathconf()*, or *sysconf* functions
 886 defined in the System Interfaces volume of POSIX.1-200x or the *getconf* utility defined in
 887 the Shell and Utilities volume of POSIX.1-200x can in some circumstances indicate that
 888 the option is supported at runtime. (For example, an old application binary might be run
 889 on a newer implementation to which support for the option has been added.)

- 890 2. Option always supported.

891 The implementation advertises at compile time (by defining the constant in `<unistd.h>`
 892 with a value greater than zero) that the option is supported both for compilation and for
 893 use at runtime. In this case, all headers, data types, function interfaces, and utilities
 894 required only for the option shall be available and shall operate as specified. Runtime

checks with *fpathconf()*, *pathconf()*, or *sysconf* shall indicate that the option is supported.

3. Option might or might not be supported at runtime.

The implementation advertises at compile time (by defining the constant in **<unistd.h>** with value zero) that the option is supported for compilation and might or might not be supported at runtime. In this case, the *fpathconf()*, *pathconf()*, or *sysconf()* functions defined in the System Interfaces volume of POSIX.1-200x or the *getconf* utility defined in the Shell and Utilities volume of POSIX.1-200x can be used to retrieve the value of each symbol on each specific implementation to determine whether the option is supported at runtime. All headers, data types, and function interfaces required to compile and execute applications which use the option at runtime (after checking at runtime that the option is supported) shall be provided, but if the option is not supported at runtime they need not operate as specified. Utilities or other facilities required only for the option, but not needed to compile and execute such applications, need not be present.

If an option is not supported for compilation, an application that attempts to use anything associated only with the option is considered to be requiring an extension. Unless explicitly specified otherwise, the behavior of functions associated with an option that is not supported at runtime is unspecified, and an application that uses such functions without first checking *fpathconf()*, *pathconf()*, or *sysconf* is considered to be requiring an extension.

Margin codes are defined for each option (see [Section 1.7.1](#), on page 7).

2.1.6.1 System Interfaces

Refer to **<unistd.h>**, [Constants for Options and Option Groups](#) (on page 412) for the list of options.

2.1.6.2 Shell and Utilities

Each of these symbols shall be considered valid names by the implementation. Refer to **<unistd.h>**, [Constants for Options and Option Groups](#) (on page 412).

The literal names shown below apply only to the *getconf* utility.

CD POSIX2_C_DEV

The system supports the C-Language Development Utilities option.

The utilities in the C-Language Development Utilities option are used for the development of C-language applications, including compilation or translation of C source code and complex program generators for simple lexical tasks and processing of context-free grammars.

The utilities listed below may be provided by a conforming system; however, any system claiming conformance to the C-Language Development Utilities option shall provide all of the utilities listed.

c99

lex

yacc

POSIX2_CHAR_TERM

The system supports the Terminal Characteristics option. This value need not be present on a system not supporting the User Portability Utilities option.

Where applicable, the dependency is noted within the description of the utility.

This option applies only to systems supporting the User Portability Utilities option. If

938 supported, then the system supports at least one terminal type capable of all operations
939 described in POSIX.1-200x; see [Section 10.2](#) (on page 184).

940 FD **POSIX2_FORT_DEV**

941 The system supports the FORTRAN Development Utilities option.

942 The *fort77* FORTRAN compiler is the only utility in the FORTRAN Development Utilities
943 option. This is used for the development of FORTRAN language applications, including
944 compilation or translation of FORTRAN source code.

945 The *fort77* utility may be provided by a conforming system; however, any system claiming
946 conformance to the FORTRAN Development Utilities option shall provide the *fort77* utility.

947 FR **POSIX2_FORT_RUN**

948 The system supports the FORTRAN Runtime Utilities option.

949 The *asa* utility is the only utility in the FORTRAN Runtime Utilities option.

950 The *asa* utility may be provided by a conforming system; however, any system claiming
951 conformance to the FORTRAN Runtime Utilities option shall provide the *asa* utility.

952 **POSIX2_LOCALEDEF**

953 The system supports the Locale Creation Utilities option.

954 If supported, the system supports the creation of locales as described in the *localedef* utility.

955 The *localedef* utility may be provided by a conforming system; however, any system
956 claiming conformance to the Locale Creation Utilities option shall provide the *localedef*
957 utility.

958 OB BE **POSIX2_PBS**

959 The system supports the Batch Environment Services and Utilities option (see XCU [Chapter](#)
960 [3](#), on page 2319).

961 **Note:** The Batch Environment Services and Utilities option is a combination of mandatory and
962 optional batch services and utilities. The `POSIX_PBS` symbolic constant implies the system
963 supports all the mandatory batch services and utilities.

964 **POSIX2_PBS_ACCOUNTING**

965 The system supports the Batch Accounting option.

966 **POSIX2_PBS_CHECKPOINT**

967 The system supports the Batch Checkpoint/Restart option.

968 **POSIX2_PBS_LOCATE**

969 The system supports the Locate Batch Job Request option.

970 **POSIX2_PBS_MESSAGE**

971 The system supports the Batch Job Message Request option.

972 **POSIX2_PBS_TRACK**

973 The system supports the Track Batch Job Request option.

974 SD **POSIX2_SW_DEV**

975 The system supports the Software Development Utilities option.

976 The utilities in the Software Development Utilities option are used for the development of
977 applications, including compilation or translation of source code, the creation and
978 maintenance of library archives, and the maintenance of groups of inter-dependent
979 programs.

980 The utilities listed below may be provided by the conforming system; however, any system
981 claiming conformance to the Software Development Utilities option shall provide all of the

982 utilities listed here.

983 *ar*
 984 *make*
 985 *nm*
 986 *strip*

987 UP POSIX2_UPE

988 The system supports the User Portability Utilities option.

989 The utilities in the User Portability Utilities option shall be implemented on all systems that
 990 claim conformance to this option, except for the *vi* utility which is noted as having features
 991 that cannot be implemented on all terminal types; if the POSIX2_CHAR_TERM option is
 992 supported, the system shall support all such features on at least one terminal type; see
 993 [Section 10.2](#) (on page 184).

994 The list of utilities in the User Portability Utilities option is as follows:

995 *bg fg talk*
 996 *ex jobs vi*
 997 *fc more*

998 XSI XOPEN_UNIX

999 The system supports the X/Open System Interfaces (XSI) option (see [Section 2.1.4](#), on page
 1000 19).

1001 UU XOPEN_UUCP

1002 The system supports the UUCP Utilities option.

1003 The list of utilities in the UUCP Utilities option is as follows:

1004 *uucp*
 1005 *uustat*
 1006 < *uux*

1007 2.2 Application Conformance

1008 For the purposes of POSIX.1-200x, the application conformance requirements given in this
 1009 section apply.

1010 All applications claiming conformance to POSIX.1-200x shall use only language-dependent
 1011 services for the C programming language described in [Section 2.3](#) (on page 30), shall use only
 1012 the utilities and facilities defined in the Shell and Utilities volume of POSIX.1-200x, and shall fall
 1013 within one of the following categories.

1014 2.2.1 Strictly Conforming POSIX Application

1015 A Strictly Conforming POSIX Application is an application that requires only the facilities
 1016 described in POSIX.1-200x. Such an application:

- 1017 1. Shall accept any implementation behavior that results from actions it takes in areas
 1018 described in POSIX.1-200x as *implementation-defined* or *unspecified*, or where POSIX.1-200x
 1019 indicates that implementations may vary
- 1020 2. Shall not perform any actions that are described as producing *undefined* results

- 1021 3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-200x,
 1022 but shall not rely on any value in the range being greater than the minimums listed or
 1023 being less than the maximums listed in POSIX.1-200x
- 1024 4. Shall not use facilities designated as *obsolescent*
- 1025 5. Is required to tolerate and permitted to adapt to the presence or absence of optional
 1026 facilities whose availability is indicated by [Section 2.1.3](#) (on page 16)
- 1027 6. For the C programming language, shall not produce any output dependent on any
 1028 behavior described in the ISO/IEC 9899:1999 standard as *unspecified*, *undefined*, or
 1029 *implementation-defined*, unless the System Interfaces volume of POSIX.1-200x specifies the
 1030 behavior
- 1031 7. For the C programming language, shall not exceed any minimum implementation limit
 1032 defined in the ISO/IEC 9899:1999 standard, unless the System Interfaces volume of
 1033 POSIX.1-200x specifies a higher minimum implementation limit
- 1034 8. For the C programming language, shall define `_POSIX_C_SOURCE` to be 200xxxL before
 1035 any header is included

1036 Within POSIX.1-200x, any restrictions placed upon a Conforming POSIX Application shall
 1037 restrict a Strictly Conforming POSIX Application.

1038 2.2.2 Conforming POSIX Application

1039 2.2.2.1 ISO/IEC Conforming POSIX Application

1040 An ISO/IEC Conforming POSIX Application is an application that uses only the facilities
 1041 described in POSIX.1-200x and approved Conforming Language bindings for any ISO or IEC
 1042 standard. Such an application shall include a statement of conformance that documents all
 1043 options and limit dependencies, and all other ISO or IEC standards used.

1044 2.2.2.2 <National Body> Conforming POSIX Application

1045 A <National Body> Conforming POSIX Application differs from an ISO/IEC Conforming
 1046 POSIX Application in that it also may use specific standards of a single ISO/IEC member body
 1047 referred to here as <National Body>. Such an application shall include a statement of
 1048 conformance that documents all options and limit dependencies, and all other <National Body>
 1049 standards used.

1050 2.2.3 Conforming POSIX Application Using Extensions

1051 A Conforming POSIX Application Using Extensions is an application that differs from a
 1052 Conforming POSIX Application only in that it uses non-standard facilities that are consistent
 1053 with POSIX.1-200x. Such an application shall fully document its requirements for these extended
 1054 facilities, in addition to the documentation required of a Conforming POSIX Application. A
 1055 Conforming POSIX Application Using Extensions shall be either an ISO/IEC Conforming
 1056 POSIX Application Using Extensions or a <National Body> Conforming POSIX Application
 1057 Using Extensions (see [Section 2.2.2.1](#) and [Section 2.2.2.2](#)).

2.2.4 Strictly Conforming XSI Application

A Strictly Conforming XSI Application is an application that requires only the facilities described in POSIX.1-200x. Such an application:

1. Shall accept any implementation behavior that results from actions it takes in areas described in POSIX.1-200x as *implementation-defined* or *unspecified*, or where POSIX.1-200x indicates that implementations may vary
2. Shall not perform any actions that are described as producing *undefined* results
3. For symbolic constants, shall accept any value in the range permitted by POSIX.1-200x, but shall not rely on any value in the range being greater than the minimums listed or being less than the maximums listed in POSIX.1-200x
4. Shall not use facilities designated as *obsolescent*
5. Is required to tolerate and permitted to adapt to the presence or absence of optional facilities whose availability is indicated by [Section 2.1.4](#) (on page 19)
6. For the C programming language, shall not produce any output dependent on any behavior described in the ISO C standard as *unspecified*, *undefined*, or *implementation-defined*, unless the System Interfaces volume of POSIX.1-200x specifies the behavior
7. For the C programming language, shall not exceed any minimum implementation limit defined in the ISO C standard, unless the System Interfaces volume of POSIX.1-200x specifies a higher minimum implementation limit
8. For the C programming language, shall define `_XOPEN_SOURCE` to be 700 before any header is included

Within POSIX.1-200x, any restrictions placed upon a Conforming POSIX Application shall restrict a Strictly Conforming XSI Application.

2.2.5 Conforming XSI Application Using Extensions

A Conforming XSI Application Using Extensions is an application that differs from a Strictly Conforming XSI Application only in that it uses non-standard facilities that are consistent with POSIX.1-200x. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Strictly Conforming XSI Application.

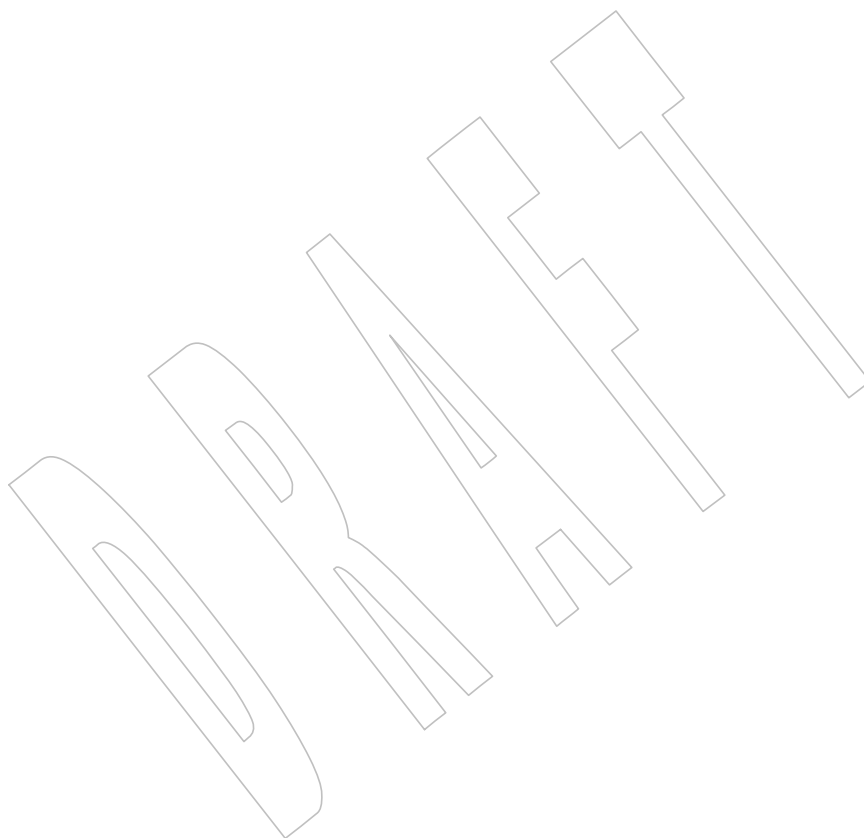
2.3 Language-Dependent Services for the C Programming Language

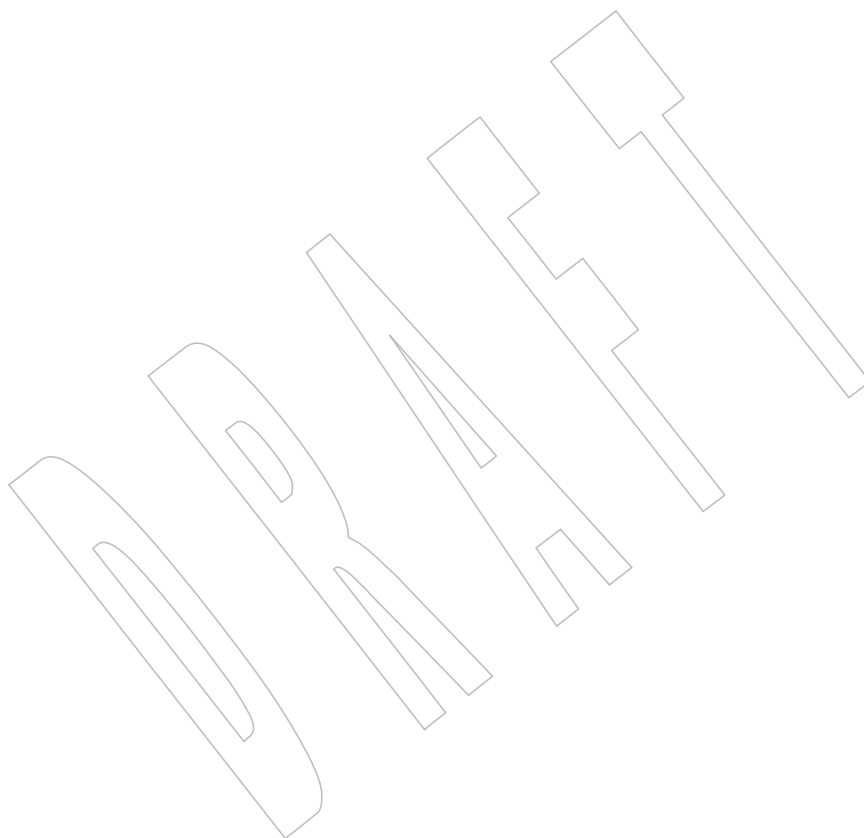
Implementors seeking to claim conformance using the ISO C standard shall claim POSIX conformance as described in [Section 2.1.3](#) (on page 16).

2.4 Other Language-Related Specifications

1089
1090 POSIX.1-200x is currently specified in terms of the shell command language and ISO C. Bindings
1091 to other programming languages are being developed.

1092 If conformance to POSIX.1-200x is claimed for implementation of any programming language,
1093 the implementation of that language shall support the use of external symbols distinct to at least
1094 31 bytes in length in the source program text. (That is, identifiers that differ at or before the
1095 thirty-first byte shall be distinct.) If a national or international standard governing a language
1096 defines a maximum length that is less than this value, the language-defined maximum shall be
1097 supported. External symbols that differ only by case shall be distinct when the character set in
1098 use distinguishes uppercase and lowercase characters and the language permits (or requires)
1099 uppercase and lowercase characters to be distinct in external symbols.





For the purposes of POSIX.1-200x, the following terms and definitions apply. The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition should be referenced for terms not defined in this section.

Note: No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

3.1 Abortive Release

An abrupt termination of a network connection that may result in the loss of data.

3.2 Absolute Pathname

A pathname beginning with a single or more than two slashes; see also [Section 3.265](#) (on page 70).

Note: Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

3.3 Access Mode

A particular form of access permitted to a file.

3.4 Additional File Access Control Mechanism

An implementation-defined mechanism that is layered upon the access control mechanisms defined here, but which do not grant permissions beyond those defined herein, although they may further restrict them.

Note: File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).

3.5 Address Space

The memory locations that can be referenced by a process or the threads of a process.

3.6 Advisory Information

An interface that advises the implementation on (portable) application behavior so that it can optimize the system.

1126 3.7 Affirmative Response

1127 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
1128 keyword **yesexpr**, matching an extended regular expression in the current locale.

1129 **Note:** The *LC_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 150).

1130 3.8 Alert

1131 To cause the user's terminal to give some audible or visual indication that an error or some other
1132 event has occurred. When the standard output is directed to a terminal device, the method for
1133 alerting the terminal user is unspecified. When the standard output is not directed to a terminal
1134 device, the alert is accomplished by writing the <alert> to standard output (unless the utility
1135 description indicates that the use of standard output produces undefined results in this case).

1136 3.9 Alert Character (<alert>)

1137 A character that in the output stream should cause a terminal to alert its user via a visual or
1138 audible notification. It is the character designated by '\a' in the C language. It is unspecified
1139 whether this character is the exact sequence transmitted to an output device by the system to
1140 accomplish the alert function.

1141 3.10 Alias Name

1142 In the shell command language, a word consisting solely of underscores, digits, and alphabets
1143 from the portable character set and any of the following characters: '!', '%', '&', '@', and '~'.

1144 Implementations may allow other characters within alias names as an extension.

1145 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 111).

1146 3.11 Alignment

1147 A requirement that objects of a particular type be located on storage boundaries with addresses
1148 that are particular multiples of a byte address.

1149 **Note:** See also the ISO C standard, Section B3.

1150 3.12 Alternate File Access Control Mechanism

1151 An implementation-defined mechanism that is independent of the access control mechanisms
1152 defined herein, and which if enabled on a file may either restrict or extend the permissions of a
1153 given user. POSIX.1-200x defines when such mechanisms can be enabled and when they are
1154 disabled.

1155 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).

1156 3.13 Alternate Signal Stack

1157 Memory associated with a thread, established upon request by the implementation for a thread,

1158 separate from the thread signal stack, in which signal handlers responding to signals sent to that
1159 thread may be executed.

1160 **3.14 Ancillary Data**

1161 Protocol-specific, local system-specific, or optional information. The information can be both
1162 local or end-to-end significant, header information, part of a data portion, protocol-specific, and
1163 implementation or system-specific.

1164 **3.15 Angle Brackets**

1165 The characters '`<`' (left-angle-bracket) and '`>`' (right-angle-bracket). When used in the phrase
1166 "enclosed in angle brackets", the symbol '`<`' immediately precedes the object to be enclosed,
1167 and '`>`' immediately follows it. When describing these characters in the portable character set,
1168 the names `<less-than-sign>` and `<greater-than-sign>` are used.

1169 **3.16 Application**

1170 A computer program that performs some desired function.

1171 **3.17 Application Address**

1172 Endpoint address of a specific application.

1173 **3.18 Application Program Interface (API)**

1174 The definition of syntax and semantics for providing computer system services.

1175 **3.19 Appropriate Privileges**

1176 An implementation-defined means of associating privileges with a process with regard to the
1177 function calls, function call options, and the commands that need special privileges. There may
1178 be zero or more such means. These means (or lack thereof) are described in the conformance
1179 document.

1180 **Note:** Function calls are defined in the System Interfaces volume of POSIX.1-200x, and commands are
1181 defined in the Shell and Utilities volume of POSIX.1-200x.

1182 3.20 Argument

1183 In the shell command language, a parameter passed to a utility as the equivalent of a single
1184 string in the *argv* array created by one of the *exec* functions. An argument is one of the options,
1185 option-arguments, or operands following the command name.

1186 **Note:** The Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 199) and XCU [Section](#)
1187 [2.9.1.1](#) (on page 2264).

1188 In the C language, an expression in a function call expression or a sequence of preprocessing
1189 tokens in a function-like macro invocation.

1190 3.21 Arm (a Timer)

1191 To start a timer measuring the passage of time, enabling notifying a process when the specified
1192 time or time interval has passed.

1193 3.22 Asterisk

1194 The character ' * '.

1195 3.23 Async-Cancel-Safe Function

1196 A function that may be safely invoked by an application while the asynchronous form of
1197 cancellation is enabled. No function is async-cancel-safe unless explicitly described as such.

1198 3.24 Asynchronous Events

1199 Events that occur independently of the execution of the application.

1200 3.25 Asynchronous Input and Output

1201 A functionality enhancement to allow an application process to queue data input and output
1202 commands with asynchronous notification of completion.

1203 3.26 Async-Signal-Safe Function

1204 A function that may be invoked, without restriction, from signal-catching functions. No function
1205 is async-signal-safe unless explicitly described as such.

1206 3.27 Asynchronously-Generated Signal

1207 A signal that is not attributable to a specific thread. Examples are signals sent via *kill()*, signals
1208 sent from the keyboard, and signals delivered to process groups. Being asynchronous is a
1209 property of how the signal was generated and not a property of the signal number. All signals
1210 may be generated asynchronously.

1211 **Note:** The *kill()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.

1212 3.28 Asynchronous I/O Completion

1213 For an asynchronous read or write operation, when a corresponding synchronous read or write
1214 would have completed and when any associated status fields have been updated.

1215 3.29 Asynchronous I/O Operation

1216 An I/O operation that does not of itself cause the thread requesting the I/O to be blocked from
1217 further use of the processor.

1218 This implies that the process and the I/O operation may be running concurrently.

1219 3.30 Authentication

1220 The process of validating a user or process to verify that the user or process is not a counterfeit.

1221 3.31 Authorization

1222 The process of verifying that a user or process has permission to use a resource in the manner
1223 requested.

1224 To ensure security, the user or process would also need to be authenticated before granting
1225 access.

1226 3.32 Background Job

1227 See *Background Process Group* in [Section 3.34](#).

1228 3.33 Background Process

1229 A process that is a member of a background process group.

1230 3.34 Background Process Group (or Background Job)

1231 Any process group, other than a foreground process group, that is a member of a session that
1232 has established a connection with a controlling terminal.

1233 **3.35 Backquote**
 1234 The character ' ` ', also known as a grave accent.

1235 **3.36 Backslash**
 1236 The character ' \ ', also known as a reverse solidus.

1237 **3.37 Backspace Character (<backspace>)**
 1238 A character that, in the output stream, should cause printing (or displaying) to occur one
 1239 column position previous to the position about to be printed. If the position about to be printed
 1240 is at the beginning of the current line, the behavior is unspecified. It is the character designated
 1241 by ' \b ' in the C language. It is unspecified whether this character is the exact sequence
 1242 transmitted to an output device by the system to accomplish the backspace function. The
 1243 <backspace> defined here is not necessarily the ERASE special character.
 1244 **Note:** Special Characters are defined in detail in [Section 11.1.9](#) (on page 189).

1245 **3.38 Barrier**
 1246 A synchronization object that allows multiple threads to synchronize at a particular point in
 1247 their execution.

1248 **3.39 Basename**
 1249 The final, or only, filename in a pathname.

1250 **3.40 Basic Regular Expression (BRE)**
 1251 A regular expression (see [Section 3.315](#), on page 77) used by the majority of utilities that select
 1252 strings from a set of character strings.
 1253 **Note:** Basic Regular Expressions are described in detail in [Section 9.3](#) (on page 169).

1254 **3.41 Batch Access List**
 1255 A list of user IDs and group IDs of those users and groups authorized to place batch jobs in a
 1256 batch queue.
 1257 A batch access list is associated with a batch queue. A batch server uses the batch access list of a
 1258 batch queue as one of the criteria in deciding to put a batch job in a batch queue.

1259 **3.42 Batch Administrator**
 1260 A user that is authorized to modify all the attributes of queues and jobs and to change the status
 1261 of a batch server.

1262 3.43 Batch Client

1263 A computational entity that utilizes batch services by making requests of batch servers.

1264 Batch clients often provide the means by which users access batch services, although a batch
1265 server may act as a batch client by virtue of making requests of another batch server.

1266 3.44 Batch Destination

1267 The batch server in a batch system to which a batch job should be sent for processing.

1268 Acceptance of a batch job at a batch destination is the responsibility of a receiving batch server.
1269 A batch destination may consist of a batch server-specific portion, a network-wide portion, or
1270 both. The batch server-specific portion is referred to as the "batch queue". The network-wide
1271 portion is referred to as a "batch server name".

1272 3.45 Batch Destination Identifier

1273 A string that identifies a specific batch destination.

1274 A string of characters in the portable character set used to specify a particular batch destination.

1275 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 111).

1276 3.46 Batch Directive

1277 A line from a file that is interpreted by the batch server. The line is usually in the form of a
1278 comment and is an additional means of passing options to the *qsub* utility.

1279 **Note:** The *qsub* utility is defined in detail in the Shell and Utilities volume of POSIX.1-200x.

1280 3.47 Batch Job

1281 A set of computational tasks for a computing system.

1282 Batch jobs are managed by batch servers.

1283 Once created, a batch job may be executing or pending execution. A batch job that is executing
1284 has an associated session leader (a process) that initiates and monitors the computational tasks
1285 of the batch job.

- 1286 **3.48 Batch Job Attribute**
- 1287 A named data type whose value affects the processing of a batch job.
- 1288 The values of the attributes of a batch job affect the processing of that job by the batch server that
- 1289 manages the batch job.
- 1290 **3.49 Batch Job Identifier**
- 1291 A unique name for a batch job. A name that is unique among all other batch job identifiers in a
- 1292 batch system and that identifies the batch server to which the batch job was originally
- 1293 submitted.
- 1294 **3.50 Batch Job Name**
- 1295 A label that is an attribute of a batch job. The batch job name is not necessarily unique.
- 1296 **3.51 Batch Job Owner**
- 1297 The *username@hostname* of the user submitting the batch job, where *username* is a user name (see
- 1298 also [Section 3.429](#), on page 92) and *hostname* is a network host name.
- 1299 **3.52 Batch Job Priority**
- 1300 A value specified by the user that may be used by an implementation to determine the order in
- 1301 which batch jobs are selected to be executed. Job priority has a numeric value in the range
- 1302 -1 024 to 1 023.
- 1303 **Note:** The batch job priority is not the execution priority (nice value) of the batch job.
- 1304 **3.53 Batch Job State**
- 1305 An attribute of a batch job which determines the types of requests that the batch server that
- 1306 manages the batch job can accept for the batch job. Valid states include QUEUED, RUNNING,
- 1307 HELD, WAITING, EXITING, and TRANSITING.
- 1308 **3.54 Batch Name Service**
- 1309 A service that assigns batch names that are unique within the batch name space, and that can
- 1310 translate a unique batch name into the location of the named batch entity.
- 1311 **3.55 Batch Name Space**
- 1312 The environment within which a batch name is known to be unique.

1313 **3.56 Batch Node**

1314 A host containing part or all of a batch system.

1315 A batch node is a host meeting at least one of the following conditions:

- 1316 • Capable of executing a batch client
- 1317 • Contains a routing batch queue
- 1318 • Contains an execution batch queue

1319 **3.57 Batch Operator**

1320 A user that is authorized to modify some, but not all, of the attributes of jobs and queues, and
1321 may change the status of the batch server.

1322 **3.58 Batch Queue**

1323 A manageable object that represents a set of batch jobs and is managed by a single batch server.

1324 **Note:** A set of batch jobs is called a batch queue largely for historical reasons. Jobs are selected from
1325 the batch queue for execution based on attributes such as priority, resource requirements, and
1326 hold conditions.

1327 See also XCU [Section 3.1.2](#) (on page 2320).

1328 **3.59 Batch Queue Attribute**

1329 A named data type whose value affects the processing of all batch jobs that are members of the
1330 batch queue.

1331 A batch queue has attributes that affect the processing of batch jobs that are members of the
1332 batch queue.

1333 **3.60 Batch Queue Position**

1334 The place, relative to other jobs in the batch queue, occupied by a particular job in a batch queue.
1335 This is defined in part by submission time and priority; see also [Section 3.61](#).

1336 **3.61 Batch Queue Priority**

1337 The maximum job priority allowed for any batch job in a given batch queue.

1338 The batch queue priority is set and may be changed by users with appropriate privilege. The
1339 priority is bounded in an implementation-defined manner.

1340 **3.62 Batch Rerunability**

1341 An attribute of a batch job indicating that it may be rerun after an abnormal termination from
1342 the beginning without affecting the validity of the results.

1343
1344
1345

1346

1348
1349
1350
1351

1352
1353
1354
1355

1356
1357
1358
1359
1360
1361

1362
1363
1364

1365
1366

3.63 Batch Restart

The action of resuming the processing of a batch job from the point of the last checkpoint. Typically, this is done if the batch job has been interrupted because of a system failure.

3.64 Batch Server

A computational entity that provides batch services.

3.65 Batch Server Name

A string of characters in the portable character set used to specify a particular server in a network.

Note: The Portable Character Set is defined in detail in [Section 6.1](#) (on page 111).

3.66 Batch Service

Computational and organizational services performed by a batch system on behalf of batch jobs. Batch services are of two types: requested and deferred.

Note: Batch Services are listed in XCU [Table 3-5](#) (on page 2333).

3.67 Batch Service Request

A solicitation of services from a batch client to a batch server.

A batch service request may entail the exchange of any number of messages between the batch client and the batch server.

When naming specific types of service requests, the term “request” is qualified by the type of request, as in *Queue Batch Job Request* and *Delete Batch Job Request*.

3.68 Batch Submission

The process by which a batch client requests that a batch server create a batch job via a *Queue Job Request* to perform a specified computational task.

3.69 Batch System

A collection of one or more batch servers.

- 1367 **3.70 Batch Target User**
1368 The name of a user on the batch destination batch server.
1369 The target user is the user name under whose account the batch job is to execute on the
1370 destination batch server.
- 1371 **3.71 Batch User**
1372 A user who is authorized to make use of batch services.
- 1373 **3.72 Bind**
1374 The process of assigning a network address to an endpoint.
- 1375 **3.73 Blank Character (<blank>)**
1376 One of the characters that belong to the **blank** character class as defined via the *LC_CTYPE*
1377 category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>.
- 1378 **3.74 Blank Line**
1379 A line consisting solely of zero or more <blank>s terminated by a <newline>; see also [Section](#)
1380 [3.144](#) (on page 53).
- 1381 **3.75 Blocked Process (or Thread)**
1382 A process (or thread) that is waiting for some condition (other than the availability of a
1383 processor) to be satisfied before it can continue execution.
- 1384 **3.76 Blocking**
1385 A property of an open file description that causes function calls associated with it to wait for the
1386 requested action to be performed before returning.
- 1387 **3.77 Block-Mode Terminal**
1388 A terminal device operating in a mode incapable of the character-at-a-time input and output
1389 operations described by some of the standard utilities.
1390 **Note:** Output Devices and Terminal Types are defined in detail in [Section 10.2](#) (on page 184).

1391 3.78 Block Special File

1392 A file that refers to a device. A block special file is normally distinguished from a character
1393 special file by providing access to the device in a manner such that the hardware characteristics
1394 of the device are not visible.

1395 3.79 Braces

1396 The characters ‘{’ (left brace) and ‘}’ (right brace), also known as curly braces. When used in
1397 the phrase “enclosed in (curly) braces” the symbol ‘{’ immediately precedes the object to be
1398 enclosed, and ‘}’ immediately follows it. When describing these characters in the portable
1399 character set, the names <left-brace> and <right-brace> are used.

1400 3.80 Brackets

1401 The characters ‘[’ (left-bracket) and ‘]’ (right-bracket), also known as square brackets. When
1402 used in the phrase “enclosed in (square) brackets” the symbol ‘[’ immediately precedes the
1403 object to be enclosed, and ‘]’ immediately follows it. When describing these characters in the
1404 portable character set, the names <left-square-bracket> and <right-square-bracket> are used.

1405 3.81 Broadcast

1406 The transfer of data from one endpoint to several endpoints, as described in RFC 919 and
1407 RFC 922.

1408 3.82 Built-In Utility (or Built-In)

1409 A utility implemented within a shell. The utilities referred to as special built-ins have special
1410 qualities. Unless qualified, the term “built-in” includes the special built-in utilities. Regular
1411 built-ins are not required to be actually built into the shell on the implementation, but they do
1412 have special command-search qualities.

1413 **Note:** Special Built-In Utilities are defined in detail in XCU [Section 2.14](#) (on page 2280). |

1414 Regular Built-In Utilities are defined in detail in XCU [Section 2.9.1.1](#) (on page 2264). |

1415 3.83 Byte

1416 An individually addressable unit of data storage that is exactly an octet, used to store a character
1417 or a portion of a character; see also [Section 3.86](#) (on page 45). A byte is composed of a
1418 contiguous sequence of 8 bits. The least significant bit is called the “low-order” bit; the most
1419 significant is called the “high-order” bit.

1420 **Note:** The definition of byte from the ISO C standard is broader than the above and might
1421 accommodate hardware architectures with different sized addressable units than octets.

1422 3.84 Byte Input/Output Functions

1423 The functions that perform byte-oriented input from streams or byte-oriented output to streams:
 1424 *fgetc()*, *fgets()*, *fprintf()*, *fputc()*, *fputs()*, *fread()*, *fscanf()*, *fwrite()*, *getc()*, *getchar()*, *gets()*, *perror()*,
 1425 *printf()*, *putc()*, *putchar()*, *puts()*, *scanf()*, *ungetc()*, *vfprintf()*, and *vprintf()*.

1426 **Note:** Functions are defined in detail in the System Interfaces volume of POSIX.1-200x.

1427 3.85 Carriage-Return Character (<carriage-return>)

1428 A character that in the output stream indicates that printing should start at the beginning of the
 1429 same physical line in which the <carriage-return> occurred. It is the character designated by
 1430 ‘\r’ in the C language. It is unspecified whether this character is the exact sequence
 1431 transmitted to an output device by the system to accomplish the movement to the beginning of
 1432 the line.

1433 3.86 Character

1434 A sequence of one or more bytes representing a single graphic symbol or control code.

1435 **Note:** This term corresponds to the ISO C standard term multi-byte character, where a single-byte
 1436 character is a special case of a multi-byte character. Unlike the usage in the ISO C standard,
 1437 *character* here has no necessary relationship with storage space, and *byte* is used when storage
 1438 space is discussed.

1439 See the definition of the portable character set in [Section 6.1](#) (on page 111) for a further
 1440 explanation of the graphical representations of (abstract) characters, as opposed to character
 1441 encodings.

1442 3.87 Character Array

1443 An array of elements of type `char`.

1444 3.88 Character Class

1445 A named set of characters sharing an attribute associated with the name of the class. The classes
 1446 and the characters that they contain are dependent on the value of the *LC_CTYPE* category in
 1447 the current locale.

1448 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 124).

1449 3.89 Character Set

1450 A finite set of different characters used for the representation, organization, or control of data.

1451

3.90 Character Special File

1452

A file that refers to a device. One specific type of character special file is a terminal device file.

1453

Note: The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 185).

1454

3.91 Character String

1455

A contiguous sequence of characters terminated by and including the first null byte.

1456

3.92 Child Process

1457

A new process created (by *fork()*, *posix_spawn()*, or *posix_spawnp()*) by a given process. A child process remains the child of the creating process as long as both processes continue to exist.

1458

1459

Note: The *fork()*, *posix_spawn()*, and *posix_spawnp()* functions are defined in detail in the System Interfaces volume of POSIX.1-200x.

1460

1461

3.93 Circumflex

1462

The character '^'.

1463

3.94 Clock

1464

A software or hardware object that can be used to measure the apparent or actual passage of time.

1465

1466

The current value of the time measured by a clock can be queried and, possibly, set to a value within the legal range of the clock.

1467

1468

3.95 Clock Jump

1469

The difference between two successive distinct values of a clock, as observed from the application via one of the “get time” operations.

1470

1471

3.96 Clock Tick

1472

An interval of time; an implementation-defined number of these occur each second. Clock ticks are one of the units that may be used to express a value found in type `clock_t`.

1473

1474

3.97 Coded Character Set

1475

A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its bit representation.

1476

3.98 Codeset

The result of applying rules that map a numeric code value to each element of a character set. An element of a character set may be related to more than one numeric code value but the reverse is not true. However, for state-dependent encodings the relationship between numeric code values and elements of a character set may be further controlled by state information. The character set may contain fewer elements than the total number of possible numeric code values; that is, some code values may be unassigned.

Note: Character Encoding is defined in detail in [Section 6.2](#) (on page 114).

3.99 Collating Element

The smallest entity used to determine the logical ordering of character or wide-character strings; see also [Section 3.101](#). A collating element consists of either a single character, or two or more characters collating as a single entity. The value of the `LC_COLLATE` category in the current locale determines the current set of collating elements.

3.100 Collation

The logical ordering of character or wide-character strings according to defined precedence rules. These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting of multiple collating elements.

3.101 Collation Sequence

The relative order of collating elements as determined by the setting of the `LC_COLLATE` category in the current locale. The collation sequence is used for sorting and is determined from the collating weights assigned to each collating element. In the absence of weights, the collation sequence is the order in which collating elements are specified between `order_start` and `order_end` keywords in the `LC_COLLATE` category.

Multi-level sorting is accomplished by assigning elements one or more collation weights, up to the limit `{COLL_WEIGHTS_MAX}`. On each level, elements may be given the same weight (at the primary level, called an equivalence class; see also [Section 3.150](#), on page 54) or be omitted from the sequence. Strings that collate equally using the first assigned weight (primary ordering) are then compared using the next assigned weight (secondary ordering), and so on.

Note: `{COLL_WEIGHTS_MAX}` is defined in detail in [<limits.h>](#).

3.102 Column Position

A unit of horizontal measure related to characters in a line.

It is assumed that each character in a character set has an intrinsic column width independent of any output device. Each printable character in the portable character set has a column width of one. The standard utilities, when used as described in POSIX.1-200x, assume that all characters have integral column widths. The column width of a character is not necessarily related to the internal representation of the character (numbers of bits or bytes).

The column position of a character in a line is defined as one plus the sum of the column widths of the preceding characters in the line. Column positions are numbered starting from 1.

1515 3.103 Command

1516 A directive to the shell to perform a particular task.

1517 **Note:** Shell Commands are defined in detail in XCU [Section 2.9](#) (on page 2263).

1518 3.104 Command Language Interpreter

1519 An interface that interprets sequences of text input as commands. It may operate on an input
1520 stream or it may interactively prompt and read commands from a terminal. It is possible for
1521 applications to invoke utilities through a number of interfaces, which are collectively considered
1522 to act as command interpreters. The most obvious of these are the *sh* utility and the *system()*
1523 function, although *popen()* and the various forms of *exec* may also be considered to behave as
1524 interpreters.

1525 **Note:** The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-200x.

1526 The *system()*, *popen()*, and *exec* functions are defined in detail in the System Interfaces volume
1527 of POSIX.1-200x.

1528 3.105 Composite Graphic Symbol

1529 A graphic symbol consisting of a combination of two or more other graphic symbols in a single
1530 character position, such as a diacritical mark and a base character.

1531 3.106 Condition Variable

1532 A synchronization object which allows a thread to suspend execution, repeatedly, until some
1533 associated predicate becomes true. A thread whose execution is suspended on a condition
1534 variable is said to be blocked on the condition variable.

1535 3.107 Connected Socket

1536 A connection-mode socket for which a connection has been established, or a connectionless-
1537 mode socket for which a peer address has been set. See also [Section 3.108](#), [Section 3.109](#), [Section](#)
1538 [3.110](#), and [Section 3.348](#) (on page 81).

1539 3.108 Connection

1540 An association established between two or more endpoints for the transfer of data

1541 3.109 Connection Mode

1542 The transfer of data in the context of a connection; see also [Section 3.110](#).

1543 3.110 Connectionless Mode

1544 The transfer of data other than in the context of a connection; see also [Section 3.109](#) and [Section](#)

1545 [3.123](#) (on page 50).

1546 **3.111 Control Character**

1547 A character, other than a graphic character, that affects the recording, processing, transmission,
1548 or interpretation of text.

1549 **3.112 Control Operator**

1550 In the shell command language, a token that performs a control function. It is one of the
1551 following symbols:

1552 & && () ; ;; newline | ||

1553 The end-of-input indicator used internally by the shell is also considered a control operator.

1554 **Note:** Token Recognition is defined in detail in XCU [Section 2.3](#) (on page 2247).

1555 **3.113 Controlling Process**

1556 The session leader that established the connection to the controlling terminal. If the terminal
1557 subsequently ceases to be a controlling terminal for this session, the session leader ceases to be
1558 the controlling process.

1559 **3.114 Controlling Terminal**

1560 A terminal that is associated with a session. Each session may have at most one controlling
1561 terminal associated with it, and a controlling terminal is associated with exactly one session.
1562 Certain input sequences from the controlling terminal cause signals to be sent to all processes in
1563 the foreground process group associated with the controlling terminal.

1564 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 185).

1565 **3.115 Conversion Descriptor**

1566 A per-process unique value used to identify an open codeset conversion.

1567 **3.116 Core File**

1568 A file of unspecified format that may be generated when a process terminates abnormally.

1569 **3.117 CPU Time (Execution Time)**

1570 The time spent executing a process or thread, including the time spent executing system services
1571 on behalf of that process or thread. If the Threads option is supported, then the value of the
1572 CPU-time clock for a process is implementation-defined. With this definition the sum of all the
1573 execution times of all the threads in a process might not equal the process execution time, even
1574 in a single-threaded process, because implementations may differ in how they account for time
1575 during context switches or for other reasons.

- 1576 **3.118 CPU-Time Clock**
1577 A clock that measures the execution time of a particular process or thread.
- 1578 **3.119 CPU-Time Timer**
1579 A timer attached to a CPU-time clock.
- 1580 **3.120 Current Job**
1581 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities. There
1582 is at most one current job; see also [Section 3.202](#) (on page 61).
- 1583 **3.121 Current Working Directory**
1584 See *Working Directory* in [Section 3.439](#) (on page 94).
- 1585 **3.122 Cursor Position**
1586 The line and column position on the screen denoted by the terminal's cursor.
- 1587 **3.123 Datagram**
1588 A unit of data transferred from one endpoint to another in connectionless mode service.
- 1589 **3.124 Data Segment**
1590 Memory associated with a process, that can contain dynamically allocated data.
- 1591 **3.125 Deferred Batch Service**
1592 A service that is performed as a result of events that are asynchronous with respect to requests.
1593 **Note:** Once a batch job has been created, it is subject to deferred services.
- 1594 **3.126 Device**
1595 A computer peripheral or an object that appears to the application as such.
- 1596 **3.127 Device ID**
1597 A non-negative integer used to identify a device.

- 1598 **3.128 Directory**
1599 A file that contains directory entries. No two directory entries in the same directory have the
1600 same name.
- 1601 **3.129 Directory Entry (or Link)**
1602 An object that associates a filename with a file. Several directory entries can associate names
1603 with the same file.
- 1604 **3.130 Directory Stream**
1605 A sequence of all the directory entries in a particular directory. An open directory stream may be
1606 implemented using a file descriptor.
- 1607 **3.131 Disarm (a Timer)**
1608 To stop a timer from measuring the passage of time, disabling any future process notifications
1609 (until the timer is armed again).
- 1610 **3.132 Display**
1611 To output to the user's terminal. If the output is not directed to a terminal, the results are
1612 undefined.
- 1613 **3.133 Display Line**
1614 A line of text on a physical device or an emulation thereof. Such a line will have a maximum
1615 number of characters which can be presented.
1616 **Note:** This may also be written as "line on the display".
- 1617 **3.134 Dollar Sign**
1618 The character '\$'.

1619 **3.135 Dot**

1620 In the context of naming files, the filename consisting of a single dot character (' . ').

1621 **Note:** In the context of shell special built-in utilities, see *dot* in XCU [Section 2.14](#) (on page 2280).

1622 Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

1623 **3.136 Dot-Dot**

1624 The filename consisting solely of two dot characters (" . . ").

1625 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

1626 **3.137 Double-Quote**

1627 The character ' " ', also known as quotation-mark.

1628 **Note:** The “double” adjective in this term refers to the two strokes in the character glyph.
1629 POSIX.1-200x never uses the term “double-quote” to refer to two apostrophes or quotation
1630 marks.

1631 **3.138 Downshifting**

1632 The conversion of an uppercase character that has a single-character lowercase representation
1633 into this lowercase representation.

1634 **3.139 Driver**

1635 A module that controls data transferred to and received from devices.

1636 **Note:** Drivers are traditionally written to be a part of the system implementation, although they are
1637 frequently written separately from the writing of the implementation. A driver may contain
1638 processor-specific code, and therefore be non-portable.

1639 **3.140 Effective Group ID**

1640 An attribute of a process that is used in determining various permissions, including file access
1641 permissions; see also [Section 3.187](#) (on page 59).

1642 **3.141 Effective User ID**

1643 An attribute of a process that is used in determining various permissions, including file access
1644 permissions; see also [Section 3.428](#) (on page 92).

1645 3.142 Eight-Bit Transparency

1646 The ability of a software component to process 8-bit characters without modifying or utilizing
1647 any part of the character in a way that is inconsistent with the rules of the current coded
1648 character set.

1649 3.143 Empty Directory

1650 A directory that contains, at most, directory entries for dot and dot-dot, and has exactly one link |
1651 to it (other than its own dot entry, if one exists), in dot-dot. No other links to the directory may |
1652 exist. It is unspecified whether an implementation can ever consider the root directory to be |
1653 empty.

1654 3.144 Empty Line

1655 A line consisting of only a <newline>; see also [Section 3.74](#) (on page 43).

1656 3.145 Empty String (or Null String)

1657 A string whose first byte is a null byte.

1658 3.146 Empty Wide-Character String

1659 A wide-character string whose first element is a null wide-character code.

1660 3.147 Encoding Rule

1661 The rules used to convert between wide-character codes and multi-byte character codes.

1662 **Note:** Stream Orientation and Encoding Rules are defined in detail in XSH [Section 2.5.2](#) (on page 472). |

1663 3.148 Entire Regular Expression

1664 The concatenated set of one or more basic regular expressions or extended regular expressions
1665 that make up the pattern specified for string selection.

1666 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 167).

1667 3.149 Epoch

1668 The time zero hours, zero minutes, zero seconds, on January 1, 1970 Coordinated Universal Time
1669 (UTC).

1670 **Note:** See also *Seconds Since the Epoch* defined in [Section 4.15](#) (on page 100). |

1671 3.150 Equivalence Class

1672 A set of collating elements with the same primary collation weight.

1673 Elements in an equivalence class are typically elements that naturally group together, such as all
1674 accented letters based on the same base letter.

1675 The collation order of elements within an equivalence class is determined by the weights
1676 assigned on any subsequent levels after the primary weight.

1677 3.151 Era

1678 A locale-specific method for counting and displaying years.

1679 **Note:** The *LC_TIME* category is defined in detail in [Section 7.3.5](#) (on page 144).

1680 3.152 Event Management

1681 The mechanism that enables applications to register for and be made aware of external events
1682 such as data becoming available for reading.

1683 3.153 Executable File

1684 A regular file acceptable as a new process image file by the equivalent of the *exec* family of
1685 functions, and thus usable as one form of a utility. The standard utilities described as compilers
1686 can produce executable files, but other unspecified methods of producing executable files may
1687 also be provided. The internal format of an executable file is unspecified, but a conforming
1688 application cannot assume an executable file is a text file.

1689 3.154 Execute

1690 To perform command search and execution actions, as defined in the Shell and Utilities volume
1691 of POSIX.1-200x; see also [Section 3.199](#) (on page 61).

1692 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2264). |

1693 3.155 Execution Time

1694 See *CPU Time* in [Section 3.117](#) (on page 49).

1695 3.156 Execution Time Monitoring

1696 A set of execution time monitoring primitives that allow online measuring of thread and process
1697 execution times.

1698 **3.157 Expand**

1699 In the shell command language, when not qualified, the act of applying word expansions.

1700 **Note:** Word Expansions are defined in detail in XCU [Section 2.6](#) (on page 2253). |1701 **3.158 Extended Regular Expression (ERE)**1702 A regular expression (see also [Section 3.315](#), on page 77) that is an alternative to the Basic
1703 Regular Expression using a more extensive syntax, occasionally used by some utilities.1704 **Note:** Extended Regular Expressions are described in detail in [Section 9.4](#) (on page 174).1705 **3.159 Extended Security Controls**1706 Implementation-defined security controls allowed by the file access permission and appropriate
1707 privilege (see also [Section 3.19](#), on page 35) mechanisms, through which an implementation can
1708 support different security policies from those described in POSIX.1-200x.1709 **Note:** See also *Extended Security Controls* defined in [Section 4.3](#) (on page 95). |1710 File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).1711 **3.160 Feature Test Macro**

1712 A macro used to determine whether a particular set of features is included from a header.

1713 **Note:** See also XSH [Section 2.2](#) (on page 448). |1714 **3.161 Field**1715 In the shell command language, a unit of text that is the result of parameter expansion,
1716 arithmetic expansion, command substitution, or field splitting. During command processing, the
1717 resulting fields are used as the command name and its arguments.1718 **Note:** Parameter Expansion is defined in detail in XCU [Section 2.6.2](#) (on page 2254). |1719 Arithmetic Expansion is defined in detail in XCU [Section 2.6.4](#) (on page 2257). |1720 Command Substitution is defined in detail in XCU [Section 2.6.3](#) (on page 2256). |1721 Field Splitting is defined in detail in XCU [Section 2.6.5](#) (on page 2258). |1722 For further information on command processing, see XCU [Section 2.9.1](#) (on page 2263). |1723 **3.162 FIFO Special File (or FIFO)**

1724 A type of file with the property that data written to such a file is read on a first-in-first-out basis.

1725 **Note:** Other characteristics of FIFOs are described in the System Interfaces volume of POSIX.1-200x,
1726 *lseek()*, *open()*, *read()*, and *write()*.

1727 **3.163 File**

1728 An object that can be written to, or read from, or both. A file has certain attributes, including
 1729 access permissions and type. File types include regular file, character special file, block special
 1730 file, FIFO special file, symbolic link, socket, and directory. Other types of files may be supported
 1731 by the implementation.

1732 **3.164 File Description**

1733 See *Open File Description* in [Section 3.252](#) (on page 67).

1734 **3.165 File Descriptor**

1735 A per-process unique, non-negative integer used to identify an open file for the purpose of file
 1736 access. The value of a file descriptor is from zero to {OPEN_MAX}. A process can have no more
 1737 than {OPEN_MAX} file descriptors open simultaneously. File descriptors may also be used to
 1738 implement message catalog descriptors and directory streams; see also [Section 3.252](#) (on page
 1739 67).

1740 **Note:** {OPEN_MAX} is defined in detail in [<limits.h>](#).

1741 **3.166 File Group Class**

1742 The property of a file indicating access permissions for a process related to the group
 1743 identification of a process. A process is in the file group class of a file if the process is not in the
 1744 file owner class and if the effective group ID or one of the supplementary group IDs of the
 1745 process matches the group ID associated with the file. Other members of the class may be
 1746 implementation-defined.

1747 **3.167 File Mode**

1748 An object containing the file mode bits and file type of a file.

1749 **Note:** File mode bits and file types are defined in detail in [<sys/stat.h>](#).

1750 **3.168 File Mode Bits**

1751 A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), set-group-ID-on-execution
 1752 bit (S_ISGID), and, on directories, the restricted deletion flag bit (S_ISVTX).

1753 **Note:** File Mode Bits are defined in detail in [<sys/stat.h>](#).

1754 **3.169 Filename**

1755 A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing
 1756 the name may be selected from the set of all character values excluding the slash character and
 1757 the null byte. The filenames dot and dot-dot have special meaning. A filename is sometimes
 1758 referred to as a "pathname component".

1759 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

1760 **3.170 File Offset**

1761 The byte position in the file where the next I/O operation begins. Each open file description
1762 associated with a regular file, block special file, or directory has a file offset. A character special
1763 file that does not refer to a terminal device may have a file offset. There is no file offset specified
1764 for a pipe or FIFO.

1765 **3.171 File Other Class**

1766 The property of a file indicating access permissions for a process related to the user and group
1767 identification of a process. A process is in the file other class of a file if the process is not in the
1768 file owner class or file group class.

1769 **3.172 File Owner Class**

1770 The property of a file indicating access permissions for a process related to the user
1771 identification of a process. A process is in the file owner class of a file if the effective user ID of
1772 the process matches the user ID of the file.

1773 **3.173 File Permission Bits**

1774 Information about a file that is used, along with other information, to determine whether a
1775 process has read, write, or execute/search permission to a file. The bits are divided into three
1776 parts: owner, group, and other. Each part is used with the corresponding file class of processes.
1777 These bits are contained in the file mode.

1778 **Note:** File modes are defined in detail in [<sys/stat.h>](#).

1779 File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).

1780 **3.174 File Serial Number**

1781 A per-file system unique identifier for a file.

1782 **3.175 File System**

1783 A collection of files and certain of their attributes. It provides a name space for file serial
1784 numbers referring to those files.

1785 **3.176 File Type**

1786 See *File* in [Section 3.163](#) (on page 56).

1787

3.177 Filter

1788

1789

1790

A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream.

1791

3.178 First Open (of a File)

1792

When a process opens a file that is not currently an open file within any process.

1793

3.179 Flow Control

1794

1795

The mechanism employed by a communications provider that constrains a sending entity to wait until the receiving entities can safely receive additional data without loss.

1796

3.180 Foreground Job

1797

See *Foreground Process Group* in [Section 3.182](#).

1798

3.181 Foreground Process

1799

A process that is a member of a foreground process group.

1800

3.182 Foreground Process Group (or Foreground Job)

1801

1802

1803

1804

A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has at most one process group of the session as the foreground process group of that controlling terminal.

1805

Note: The General Terminal Interface is defined in detail in [Chapter 11](#).

1806

3.183 Foreground Process Group ID

1807

The process group ID of the foreground process group.

1808 **3.184 Form-Feed Character (<form-feed>)**

1809 A character that in the output stream indicates that printing should start on the next page of an
 1810 output device. It is the character designated by ‘\f’ in the C language. If the <form-feed> is not
 1811 the first character of an output line, the result is unspecified. It is unspecified whether this
 1812 character is the exact sequence transmitted to an output device by the system to accomplish the
 1813 movement to the next page.

1814 **3.185 Graphic Character**

1815 A member of the **graph** character class of the current locale.

1816 **Note:** The **graph** character class is defined in detail in [Section 7.3.1](#) (on page 124).

1817 **3.186 Group Database**

1818 A system database that contains at least the following information for each group ID:

- 1819 • Group name
- 1820 • Numerical group ID
- 1821 • List of users allowed in the group

1822 The list of users allowed in the group is used by the *newgrp* utility.

1823 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-200x.

1824 **3.187 Group ID**

1825 A non-negative integer, which can be contained in an object of type **gid_t**, that is used to identify
 1826 a group of system users. Each system user is a member of at least one group. When the identity
 1827 of a group is associated with a process, a group ID value is referred to as a real group ID, an
 1828 effective group ID, one of the supplementary group IDs, or a saved set-group-ID.

1829 **3.188 Group Name**

1830 A string that is used to identify a group; see also [Section 3.186](#). To be portable across conforming
 1831 systems, the value is composed of characters from the portable filename character set. The
 1832 hyphen should not be used as the first character of a portable group name.

1833 **3.189 Hard Limit**

1834 A system resource limitation that may be reset to a lesser or greater limit by a privileged process.
 1835 A non-privileged process is restricted to only lowering its hard limit.

1836 **3.190 Hard Link**

1837 The relationship between two directory entries that represent the same file; see also [Section 3.129](#)
 1838 (on page 51). The result of an execution of the *ln* utility (without the *-s* option) or the *link()*
 1839 function. This term is contrasted against symbolic link; see also [Section 3.373](#) (on page 84).

1840

3.191 Home Directory

1841

The directory specified by the *HOME* environment variable.

1842

3.192 Host Byte Order

1843

The arrangement of bytes in any integer type when using a specific machine architecture.

1844

Note: Two common methods of byte ordering are big-endian and little-endian. Big-endian is a format for storage of binary data in which the most significant byte is placed first, with the rest in descending order. Little-endian is a format for storage or transmission of binary data in which the least significant byte is placed first, with the rest in ascending order. See also [Section 4.9](#) (on page 98).

1845

1846

1847

1848

1849

3.193 Incomplete Line

1850

A sequence of one or more non-`<newline>`s at the end of the file.

1851

3.194 Inf

1852

A value representing +infinity or a value representing -infinity that can be stored in a floating type. Not all systems support the Inf values.

1853

1854

3.195 Instrumented Application

1855

An application that contains at least one call to the trace point function *posix_trace_event()*. Each process of an instrumented application has a mapping of trace event names to trace event type identifiers. This mapping is used by the trace stream that is created for that process.

1856

1857

1858

3.196 Interactive Shell

1859

A processing mode of the shell that is suitable for direct user interaction.

1860

3.197 Internationalization

1861

The provision within a computer program of the capability of making itself adaptable to the requirements of different native languages, local customs, and coded character sets.

1862

1863

3.198 Interprocess Communication

1864

A functionality enhancement to add a high-performance, deterministic interprocess communication facility for local communication.

1865

1866 **3.199 Invoke**

1867 To perform command search and execution actions, except that searching for shell functions and
 1868 special built-in utilities is suppressed; see also [Section 3.154](#) (on page 54).

1869 **Note:** Command Search and Execution is defined in detail in XCU [Section 2.9.1.1](#) (on page 2264). |

1870 **3.200 Job**

1871 A set of processes, comprising a shell pipeline, and any processes descended from it, that are all
 1872 in the same process group.

1873 **Note:** See also XCU [Section 2.9.2](#) (on page 2265). |

1874 **3.201 Job Control**

1875 A facility that allows users selectively to stop (suspend) the execution of processes and continue
 1876 (resume) their execution at a later point. The user typically employs this facility via the
 1877 interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

1878 **3.202 Job Control Job ID**

1879 A handle that is used to refer to a job. The job control job ID can be any of the forms shown in
 1880 the following table:

1881 **Table 3-1** Job Control Job ID Formats

Job Control Job ID	Meaning
%%	Current job.
%+	Current job.
%-	Previous job.
%n	Job number <i>n</i> .
%string	Job whose command begins with <i>string</i> .
%?string	Job whose command contains <i>string</i> .

1890 **3.203 Last Close (of a File)**

1891 When a process closes a file, resulting in the file not being an open file within any process.

1892 **3.204 Line**

1893 A sequence of zero or more non-`<newline>`s plus a terminating `<newline>`.

1894 **3.205 Linger**

1895 A period of time before terminating a connection, to allow outstanding data to be transferred.

1896 **3.206 Link**
1897 See *Directory Entry* in [Section 3.129](#) (on page 51).

1898 **3.207 Link Count**
1899 The number of directory entries that refer to a particular file.

1900 **3.208 Local Customs**
1901 The conventions of a geographical area or territory for such things as date, time, and currency
1902 formats.

1903 **3.209 Local Interprocess Communication (Local IPC)**
1904 The transfer of data between processes in the same system.

1905 **3.210 Locale**
1906 The definition of the subset of a user's environment that depends on language and cultural
1907 conventions.
1908 **Note:** Locales are defined in detail in [Chapter 7](#) (on page 121).

1909 **3.211 Localization**
1910 The process of establishing information within a computer system specific to the operation of
1911 particular native languages, local customs, and coded character sets.

1912 **3.212 Login**
1913 The unspecified activity by which a user gains access to the system. Each login is associated
1914 with exactly one login name.

1915 **3.213 Login Name**
1916 A user name that is associated with a login.

1917 **3.214 Map**
1918 To create an association between a page-aligned range of the address space of a process and
1919 some memory object, such that a reference to an address in that range of the address space
1920 results in a reference to the associated memory object. The mapped memory object is not
1921 necessarily memory-resident.

1922 3.215 Marked Message

1923 A STREAMS message on which a certain flag is set. Marking a message gives the application
1924 protocol-specific information. An application can use *ioctl()* to determine whether a given
1925 message is marked.

1926 **Note:** The *ioctl()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.

1927 3.216 Matched

1928 A state applying to a sequence of zero or more characters when the characters in the sequence
1929 correspond to a sequence of characters defined by a basic regular expression or extended regular
1930 expression pattern.

1931 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 167).

1932 3.217 Memory Mapped Files

1933 A facility to allow applications to access files as part of the address space.

1934 3.218 Memory Object

1935 One of:

- 1936 • A file (see [Section 3.163](#), on page 56)
- 1937 • A shared memory object (see [Section 3.340](#), on page 80)
- 1938 • A typed memory object (see [Section 3.421](#), on page 91)

1939 When used in conjunction with *mmap()*, a memory object appears in the address space of the
1940 calling process.

1941 **Note:** The *mmap()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.

1942 3.219 Memory-Resident

1943 The process of managing the implementation in such a way as to provide an upper bound on
1944 memory access times.

1945 3.220 Message

1946 In the context of programmatic message passing, information that can be transferred between
1947 processes or threads by being added to and removed from a message queue. A message consists
1948 of a fixed-size message buffer.

1949 3.221 Message Catalog

1950 In the context of providing natural language messages to the user, a file or storage area
1951 containing program messages, command prompts, and responses to prompts for a particular
1952 native language, territory, and codeset.

1953 3.222 Message Catalog Descriptor

1954 In the context of providing natural language messages to the user, a per-process unique value
1955 used to identify an open message catalog. A message catalog descriptor may be implemented
1956 using a file descriptor.

1957 3.223 Message Queue

1958 In the context of programmatic message passing, an object to which messages can be added and
1959 removed. Messages may be removed in the order in which they were added or in priority order.

1960 3.224 Mode

1961 A collection of attributes that specifies a file's type and its access permissions.

1962 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).

1963 3.225 Monotonic Clock

1964 A realtime clock whose value cannot be set via `clock_settime()` and which cannot have negative
1965 clock jumps.

1966 3.226 Mount Point

1967 Either the system root directory or a directory for which the `st_dev` field of structure `stat` differs
1968 from that of its parent directory.

1969 **Note:** The `stat` structure is defined in detail in [<sys/stat.h>](#).

1970 3.227 Multi-Character Collating Element

1971 A sequence of two or more characters that collate as an entity. For example, in some coded
1972 character sets, an accented character is represented by a non-spacing accent, followed by the
1973 letter. Other examples are the Spanish elements *ch* and *ll*.

1974 3.228 Mutex

1975 A synchronization object used to allow multiple threads to serialize their access to shared data.
1976 The name derives from the capability it provides; namely, mutual-exclusion. The thread that has
1977 locked a mutex becomes its owner and remains the owner until that same thread unlocks the
1978 mutex.

- 1979 **3.229 Name**
- 1980 In the shell command language, a word consisting solely of underscores, digits, and alphabets
- 1981 from the portable character set. The first character of a name is not a digit.
- 1982 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 111).
- 1983 **3.230 Named STREAM**
- 1984 A STREAMS-based file descriptor that is attached to a name in the file system name space. All
- 1985 subsequent operations on the named STREAM act on the STREAM that was associated with the
- 1986 file descriptor until the name is disassociated from the STREAM.
- 1987 **3.231 NaN (Not a Number)**
- 1988 A set of values that may be stored in a floating type but that are neither Inf nor valid floating-
- 1989 point numbers. Not all systems support NaN values.
- 1990 **3.232 Native Language**
- 1991 A computer user's spoken or written language, such as American English, British English,
- 1992 Danish, Dutch, French, German, Italian, Japanese, Norwegian, or Swedish.
- 1993 **3.233 Negative Response**
- 1994 An input string that matches one of the responses acceptable to the *LC_MESSAGES* category
- 1995 keyword **noexpr**, matching an extended regular expression in the current locale.
- 1996 **Note:** The *LC_MESSAGES* category is defined in detail in [Section 7.3.6](#) (on page 150).
- 1997 **3.234 Network**
- 1998 A collection of interconnected hosts.
- 1999 **Note:** The term "network" in POSIX.1-200x is used to refer to the network of hosts. The term "batch
- 2000 system" is used to refer to the network of batch servers.
- 2001 **3.235 Network Address**
- 2002 A network-visible identifier used to designate specific endpoints in a network. Specific
- 2003 endpoints on host systems have addresses, and host systems may also have addresses.

2004 3.236 Network Byte Order

2005 The way of representing any integer type such that, when transmitted over a network via a
2006 network endpoint, the `int` type is transmitted as an appropriate number of octets with the most
2007 significant octet first, followed by any other octets in descending order of significance.

2008 **Note:** This order is more commonly known as big-endian ordering. See also [Section 4.9](#) (on page 98).

2009 3.237 Newline Character (<newline>)

2010 A character that in the output stream indicates that printing should start at the beginning of the
2011 next line. It is the character designated by '`\n`' in the C language. It is unspecified whether this
2012 character is the exact sequence transmitted to an output device by the system to accomplish the
2013 movement to the next line.

2014 3.238 Nice Value

2015 A number used as advice to the system to alter process scheduling. Numerically smaller values
2016 give a process additional preference when scheduling a process to run. Numerically larger
2017 values reduce the preference and make a process less likely to run. Typically, a process with a
2018 smaller nice value runs to completion more quickly than an equivalent process with a higher
2019 nice value. The symbol `{NZERO}` specifies the default nice value of the system.

2020 3.239 Non-Blocking

2021 A property of an open file description that causes function calls involving it to return without
2022 delay when it is detected that the requested action associated with the function call cannot be
2023 completed without unknown delay.

2024 **Note:** The exact semantics are dependent on the type of file associated with the open file description.
2025 For data reads from devices such as ttys and FIFOs, this property causes the read to return
2026 immediately when no data was available. Similarly, for writes, it causes the call to return
2027 immediately when the thread would otherwise be delayed in the write operation; for example,
2028 because no space was available. For networking, it causes functions not to await protocol events
2029 (for example, acknowledgements) to occur. See also [XSH Section 2.10.7](#) (on page 498).

2030 3.240 Non-Spacing Characters

2031 A character, such as a character representing a diacritical mark in the ISO/IEC 6937:2001
2032 standard coded character set, which is used in combination with other characters to form
2033 composite graphic symbols.

2034 3.241 NUL

2035 A character with all bits set to zero.

2036 3.242 Null Byte

2037 A byte with all bits set to zero.

2038
2039
2040
2041

3.243 Null Pointer

The value that is obtained by converting the number 0 into a pointer; for example, `(void *) 0`. The C language guarantees that this value does not match that of any legitimate pointer, so it is used by many functions that return pointers to indicate an error.

2042
2043

3.244 Null String

See *Empty String* in [Section 3.145](#) (on page 53).

2044
2045

3.245 Null Wide-Character Code

A wide-character code with all bits set to zero.

2046
2047

3.246 Number Sign

The character `'#'`, also known as hash sign.

2048
2049
2050
2051
2052

3.247 Object File

A regular file containing the output of a compiler, formatted as input to a linkage editor for linking with other object files into an executable form. The methods of linking are unspecified and may involve the dynamic linking of objects at runtime. The internal format of an object file is unspecified, but a conforming application cannot assume an object file is a text file.

2053
2054

3.248 Octet

Unit of data representation that consists of eight contiguous bits.

2055
2056
2057

3.249 Offset Maximum

An attribute of an open file description representing the largest value that can be used as a file offset.

2058
2059

3.250 Opaque Address

An address such that the entity making use of it requires no details about its contents or format.

2060
2061

3.251 Open File

A file that is currently associated with a file descriptor.

2062
2063

3.252 Open File Description

A record of how a process or group of processes is accessing a file. Each file descriptor refers to

2064 exactly one open file description, but an open file description can be referred to by more than
2065 one file descriptor. The file offset, file status, and file access modes are attributes of an open file
2066 description.

2067 **3.253 Operand**

2068 An argument to a command that is generally used as an object supplying information to a utility
2069 necessary to complete its processing. Operands generally follow the options in a command line.

2070 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 199).

2071 **3.254 Operator**

2072 In the shell command language, either a control operator or a redirection operator.

2073 **3.255 Option**

2074 An argument to a command that is generally used to specify changes in the utility's default
2075 behavior.

2076 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 199).

2077 **3.256 Option-Argument**

2078 A parameter that follows certain options. In some cases an option-argument is included within
2079 the same argument string as the option—in most cases it is the next argument.

2080 **Note:** Utility Argument Syntax is defined in detail in [Section 12.1](#) (on page 199).

2081 **3.257 Orientation**

2082 A stream has one of three orientations: unoriented, byte-oriented, or wide-oriented.

2083 **Note:** For further information, see XSH [Section 2.5.2](#) (on page 472).

2084 **3.258 Orphaned Process Group**

2085 A process group in which the parent of every member is either itself a member of the group or is
2086 not a member of the group's session.

2087 **3.259 Page**

2088 The granularity of process memory mapping or locking.

2089 Physical memory and memory objects can be mapped into the address space of a process on
2090 page boundaries and in integral multiples of pages. Process address space can be locked into
2091 memory (made memory-resident) on page boundaries and in integral multiples of pages.

2092 **3.260 Page Size**

2093 The size, in bytes, of the system unit of memory allocation, protection, and mapping. On
2094 systems that have segment rather than page-based memory architectures, the term “page”
2095 means a segment.

2096 **3.261 Parameter**

2097 In the shell command language, an entity that stores values. There are three types of parameters:
2098 variables (named parameters), positional parameters, and special parameters. Parameter
2099 expansion is accomplished by introducing a parameter with the ‘\$’ character.

2100 **Note:** See also XCU [Section 2.5](#) (on page 2249).

2101 In the C language, an object declared as part of a function declaration or definition that acquires
2102 a value on entry to the function, or an identifier following the macro name in a function-like
2103 macro definition.

2104 **3.262 Parent Directory**

2105 When discussing a given directory, the directory that both contains a directory entry for the
2106 given directory and is represented by the pathname dot-dot in the given directory.

2107 When discussing other types of files, a directory containing a directory entry for the file under
2108 discussion.

2109 This concept does not apply to dot and dot-dot.

2110 **3.263 Parent Process**

2111 The process which created (or inherited) the process under discussion.

2112 **3.264 Parent Process ID**

2113 An attribute of a new process identifying the parent of the process. The parent process ID of a
2114 process is the process ID of its creator, for the lifetime of the creator. After the creator’s lifetime
2115 has ended, the parent process ID is the process ID of an implementation-defined system process.

2116 3.265 Pathname

2117 A character string that is used to identify a file. In the context of POSIX.1-200x, a pathname
 2118 consists of, at most, {PATH_MAX} bytes, including the terminating null byte. It has an optional
 2119 beginning slash, followed by zero or more filenames separated by slashes. A pathname may
 2120 optionally contain one or more trailing slashes. Multiple successive slashes are considered to be
 2121 the same as one slash.

2122 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

2123 3.266 Pathname Component

2124 See *Filename* in [Section 3.169](#) (on page 56).

2125 3.267 Path Prefix

2126 A pathname, with an optional ending slash, that refers to a directory.

2127 3.268 Pattern

2128 A sequence of characters used either with regular expression notation or for pathname
 2129 expansion, as a means of selecting various character strings or pathnames, respectively.

2130 **Note:** Regular Expressions are defined in detail in [Chapter 9](#) (on page 167).

2131 See also XCU [Section 2.6.6](#) (on page 2259).

2132 The syntaxes of the two types of patterns are similar, but not identical; POSIX.1-200x always
 2133 indicates the type of pattern being referred to in the immediate context of the use of the term.

2134 3.269 Period

2135 The character '.'. The term "period" is contrasted with dot (see also [Section 3.135](#), on page 52),
 2136 which is used to describe a specific directory entry.

2137 3.270 Permissions

2138 Attributes of an object that determine the privilege necessary to access or manipulate the object.

2139 **Note:** File Access Permissions are defined in detail in [Section 4.4](#) (on page 96).

2140 3.271 Persistence

2141 A mode for semaphores, shared memory, and message queues requiring that the object and its
 2142 state (including data, if any) are preserved after the object is no longer referenced by any
 2143 process.

2144 Persistence of an object does not imply that the state of the object is maintained across a system
 2145 crash or a system reboot.

2146 **3.272 Pipe**

2147 An object identical to a FIFO which has no links in the file hierarchy. |

2148 **Note:** The *pipe()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.2149 **3.273 Polling**2150 A scheduling scheme whereby the local process periodically checks until the pre-specified
2151 events (for example, read, write) have occurred.2152 **3.274 Portable Character Set**2153 The collection of characters that are required to be present in all locales supported by
2154 conforming systems.2155 **Note:** The Portable Character Set is defined in detail in [Section 6.1](#) (on page 111).2156 This term is contrasted against the smaller portable filename character set; see also [Section 3.275](#).2157 **3.275 Portable Filename Character Set**

2158 The set of characters from which portable filenames are constructed.

2159 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
2160 a b c d e f g h i j k l m n o p q r s t u v w x y z
2161 0 1 2 3 4 5 6 7 8 9 . _ -

2162 The last three characters are the period, underscore, and hyphen characters, respectively.

2163 **3.276 Positional Parameter**2164 In the shell command language, a parameter denoted by a single digit or one or more digits in
2165 curly braces.2166 **Note:** For further information, see XCU [Section 2.5.1](#) (on page 2249). |2167 **3.277 Preallocation**

2168 The reservation of resources in a system for a particular use.

2169 Preallocation does not imply that the resources are immediately allocated to that use, but merely
2170 indicates that they are guaranteed to be available in bounded time when needed.

2171 **3.278 Preempted Process (or Thread)**

2172 A running thread whose execution is suspended due to another thread becoming runnable at a
2173 higher priority.

2174 **3.279 Previous Job**

2175 In the context of job control, the job that will be used as the default for the *fg* or *bg* utilities if the
2176 current job exits. There is at most one previous job; see also [Section 3.202](#) (on page 61).

2177 **3.280 Printable Character**

2178 One of the characters included in the **print** character classification of the *LC_CTYPE* category in
2179 the current locale.

2180 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 124).

2181 **3.281 Printable File**

2182 A text file consisting only of the characters included in the **print** and **space** character
2183 classifications of the *LC_CTYPE* category and the <backspace>, all in the current locale.

2184 **Note:** The *LC_CTYPE* category is defined in detail in [Section 7.3.1](#) (on page 124).

2185 **3.282 Priority**

2186 A non-negative integer associated with processes or threads whose value is constrained to a
2187 range defined by the applicable scheduling policy. Numerically higher values represent higher
2188 priorities.

2189 **3.283 Priority Band**

2190 The queuing order applied to normal priority STREAMS messages. High priority STREAMS
2191 messages are not grouped by priority bands. The only differentiation made by the STREAMS
2192 mechanism is between zero and non-zero bands, but specific protocol modules may differentiate
2193 between priority bands.

2194 **3.284 Priority Inversion**

2195 A condition in which a thread that is not voluntarily suspended (waiting for an event or time
2196 delay) is not running while a lower priority thread is running. Such blocking of the higher
2197 priority thread is often caused by contention for a shared resource.

2198 **3.285 Priority Scheduling**

2199 A performance and determinism improvement facility to allow applications to determine the
2200 order in which threads that are ready to run are granted access to processor resources.

2201 3.286 Priority-Based Scheduling

2202 Scheduling in which the selection of a running thread is determined by the priorities of the
2203 runnable processes or threads.

2204 3.287 Privilege

2205 See *Appropriate Privileges* in [Section 3.19](#) (on page 35).

2206 3.288 Process

2207 An address space with one or more threads executing within that address space, and the
2208 required system resources for those threads.

2209 **Note:** Many of the system resources defined by POSIX.1-200x are shared among all of the threads
2210 within a process. These include the process ID, the parent process ID, process group ID, session
2211 membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID,
2212 supplementary group IDs, current working directory, root directory, file mode creation mask,
2213 and file descriptors.

2214 3.289 Process Group

2215 A collection of processes that permits the signaling of related processes. Each process in the
2216 system is a member of a process group that is identified by a process group ID. A newly created
2217 process joins the process group of its creator.

2218 3.290 Process Group ID

2219 The unique positive integer identifier representing a process group during its lifetime.

2220 **Note:** See also *Process Group ID Reuse* defined in [Section 4.13](#) (on page 100).

2221 3.291 Process Group Leader

2222 A process whose process ID is the same as its process group ID.

2223 3.292 Process Group Lifetime

2224 A period of time that begins when a process group is created and ends when the last remaining
2225 process in the group leaves the group, due either to the end of the lifetime of the last process or
2226 to the last remaining process calling the *setsid()* or *setpgid()* functions.

2227 **Note:** The *setsid()* and *setpgid()* functions are defined in detail in the System Interfaces volume of
2228 POSIX.1-200x.

2229 **3.293 Process ID**

2230 The unique positive integer identifier representing a process during its lifetime.

2231 **Note:** See also *Process ID Reuse* defined in [Section 4.13](#) (on page 100). |2232 **3.294 Process Lifetime**

2233 The period of time that begins when a process is created and ends when its process ID is
 2234 returned to the system. After a process is created by *fork()*, *posix_spawn()*, or *posix_spawnnp()*, it is
 2235 considered active. At least one thread of control and address space exist until it terminates. It
 2236 then enters an inactive state where certain resources may be returned to the system, although
 2237 some resources, such as the process ID, are still in use. When another process executes a *wait()*,
 2238 *waitid()*, or *waitpid()* function for an inactive process, the remaining resources are returned to
 2239 the system. The last resource to be returned to the system is the process ID. At this time, the
 2240 lifetime of the process ends.

2241 **Note:** The *fork()*, *posix_spawn()*, *posix_spawnnp()*, *wait()*, *waitid()*, and *waitpid()* functions are defined in
 2242 detail in the System Interfaces volume of POSIX.1-200x.

2243 **3.295 Process Memory Locking**

2244 A performance improvement facility to bind application programs into the high-performance
 2245 random access memory of a computer system. This avoids potential latencies introduced by the
 2246 operating system in storing parts of a program that were not recently referenced on secondary
 2247 memory devices.

2248 **3.296 Process Termination**

2249 There are two kinds of process termination:

- 2250 1. Normal termination occurs by a return from *main()*, when requested with the *exit()*,
 2251 *_exit()*, or *_Exit()* functions; or when the last thread in the process terminates by
 2252 returning from its start function, by calling the *pthread_exit()* function, or through
 2253 cancellation.
- 2254 2. Abnormal termination occurs when requested by the *abort()* function or when some
 2255 signals are received.

2256 **Note:** The *_exit()*, *_Exit()*, *abort()*, and *exit()* functions are defined in detail in the System Interfaces
 2257 volume of POSIX.1-200x.

2258 **3.297 Process-To-Process Communication**

2259 The transfer of data between processes.

2260 **3.298 Process Virtual Time**

2261 The measurement of time in units elapsed by the system clock while a process is executing.

2262 3.299 Program

2263 A prepared sequence of instructions to the system to accomplish a defined task. The term
2264 “program” in POSIX.1-200x encompasses applications written in the Shell Command Language,
2265 complex utility input languages (for example, *awk*, *lex*, *sed*, and so on), and high-level languages.

2266 3.300 Protocol

2267 A set of semantic and syntactic rules for exchanging information.

2268 3.301 Pseudo-Terminal

2269 A facility that provides an interface that is identical to the terminal subsystem, except where
2270 noted otherwise in POSIX.1-200x. A pseudo-terminal is composed of two devices: the “master
2271 device” and a “slave device”. The slave device provides processes with an interface that is
2272 identical to the terminal interface, although there need not be hardware behind that interface.
2273 Anything written on the master device is presented to the slave as an input and anything
2274 written on the slave device is presented as an input on the master side.

2275 3.302 Radix Character

2276 The character that separates the integer part of a number from the fractional part.

2277 3.303 Read-Only File System

2278 A file system that has implementation-defined characteristics restricting modifications.

2279 **Note:** File Times Update is described in detail in [Section 4.8](#) (on page 97).

2280 3.304 Read-Write Lock

2281 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
2282 read-only access to data while allowing only one thread to have write access at any given time.
2283 They are typically used to protect data that is read-only more frequently than it is changed.

2284 Read-write locks can be used to synchronize threads in the current process and other processes if
2285 they are allocated in memory that is writable and shared among the cooperating processes and
2286 have been initialized for this behavior.

2287 3.305 Real Group ID

2288 The attribute of a process that, at the time of process creation, identifies the group of the user
2289 who created the process; see also [Section 3.187](#) (on page 59).

- 2290 **3.306 Real Time**
- 2291 Time measured as total units elapsed by the system clock without regard to which thread is
- 2292 executing.
- 2293 **3.307 Realtime Signal Extension**
- 2294 A determinism improvement facility to enable asynchronous signal notifications to an
- 2295 application to be queued without impacting compatibility with the existing signal functions.
- 2296 **3.308 Real User ID**
- 2297 The attribute of a process that, at the time of process creation, identifies the user who created the
- 2298 process; see also [Section 3.428](#) (on page 92).
- 2299 **3.309 Record**
- 2300 A collection of related data units or words which is treated as a unit.
- 2301 **3.310 Redirection**
- 2302 In the shell command language, a method of associating files with the input or output of
- 2303 commands.
- 2304 **Note:** For further information, see XCU [Section 2.7](#) (on page 2259).
- 2305 **3.311 Redirection Operator**
- 2306 In the shell command language, a token that performs a redirection function. It is one of the
- 2307 following symbols:
- 2308 < > >| << >> <& >& <<- <>
- 2309 **3.312 Reentrant Function**
- 2310 A function whose effect, when called by two or more threads, is guaranteed to be as if the
- 2311 threads each executed the function one after another in an undefined order, even if the actual
- 2312 execution is interleaved.
- 2313 **3.313 Referenced Shared Memory Object**
- 2314 A shared memory object that is open or has one or more mappings defined on it.
- 2315 **3.314 Refresh**
- 2316 To ensure that the information on the user's terminal screen is up-to-date.

2317 **3.315 Regular Expression**

2318 A pattern that selects specific strings from a set of character strings.

2319 **Note:** Regular Expressions are described in detail in [Chapter 9](#) (on page 167).

2320 **3.316 Region**

2321 In the context of the address space of a process, a sequence of addresses.

2322 In the context of a file, a sequence of offsets.

2323 **3.317 Regular File**

2324 A file that is a randomly accessible sequence of bytes, with no further structure imposed by the
2325 system.

2326 **3.318 Relative Pathname**

2327 A pathname not beginning with a slash.

2328 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

2329 **3.319 Relocatable File**

2330 A file holding code or data suitable for linking with other object files to create an executable or a
2331 shared object file.

2332 **3.320 Relocation**

2333 The process of connecting symbolic references with symbolic definitions. For example, when a
2334 program calls a function, the associated call instruction transfers control to the proper
2335 destination address at execution.

2336 **3.321 Requested Batch Service**

2337 A service that is either rejected or performed prior to a response from the service to the
2338 requester.

2339 **3.322 (Time) Resolution**

2340 The minimum time interval that a clock can measure or whose passage a timer can detect.

2341 3.323 Robust Mutex

2342 A mutex with the *robust* attribute set.

2343 **Note:** The *robust* attribute is defined in detail by the `pthread_mutexattr_getrobust()` function.

2344 3.324 Root Directory

2345 A directory, associated with a process, that is used in pathname resolution for pathnames that
2346 begin with a slash.

2347 3.325 Runnable Process (or Thread)

2348 A thread that is capable of being a running thread, but for which no processor is available.

2349 3.326 Running Process (or Thread)

2350 A thread currently executing on a processor. On multi-processor systems there may be more
2351 than one such thread in a system at a time.

2352 3.327 Saved Resource Limits

2353 An attribute of a process that provides some flexibility in the handling of unrepresentable
2354 resource limits, as described in the *exec* family of functions and `setrlimit()`.

2355 **Note:** The *exec* and `setrlimit()` functions are defined in detail in the System Interfaces volume of
2356 POSIX.1-200x.

2357 3.328 Saved Set-Group-ID

2358 An attribute of a process that allows some flexibility in the assignment of the effective group ID
2359 attribute, as described in the *exec* family of functions and `setgid()`.

2360 **Note:** The *exec* and `setgid()` functions are defined in detail in the System Interfaces volume of
2361 POSIX.1-200x.

2362 3.329 Saved Set-User-ID

2363 An attribute of a process that allows some flexibility in the assignment of the effective user ID
2364 attribute, as described in the *exec* family of functions and `setuid()`.

2365 **Note:** The *exec* and `setuid()` functions are defined in detail in the System Interfaces volume of
2366 POSIX.1-200x.

2367 3.330 Scheduling

2368 The application of a policy to select a runnable process or thread to become a running process or
2369 thread, or to alter one or more of the thread lists.

2370 3.331 Scheduling Allocation Domain

2371 The set of processors on which an individual thread can be scheduled at any given time.

2372 3.332 Scheduling Contention Scope

2373 A property of a thread that defines the set of threads against which that thread competes for
2374 resources.

2375 For example, in a scheduling decision, threads sharing scheduling contention scope compete for
2376 processor resources. In POSIX.1-200x, a thread has scheduling contention scope of either
2377 PTHREAD_SCOPE_SYSTEM or PTHREAD_SCOPE_PROCESS.

2378 3.333 Scheduling Policy

2379 A set of rules that is used to determine the order of execution of processes or threads to achieve
2380 some goal.

2381 **Note:** Scheduling Policy is defined in detail in [Section 4.14](#) (on page 100).

2382 3.334 Screen

2383 A rectangular region of columns and lines on a terminal display. A screen may be a portion of a
2384 physical display device or may occupy the entire physical area of the display device.

2385 3.335 Scroll

2386 To move the representation of data vertically or horizontally relative to the terminal screen.
2387 There are two types of scrolling:

- 2388 1. The cursor moves with the data.
- 2389 2. The cursor remains stationary while the data moves.

2390 3.336 Semaphore

2391 A minimum synchronization primitive to serve as a basis for more complex synchronization
2392 mechanisms to be defined by the application program.

2393 **Note:** Semaphores are defined in detail in [Section 4.16](#) (on page 101).

2394 3.337 Session

2395 A collection of process groups established for job control purposes. Each process group is a
2396 member of a session. A process is considered to be a member of the session of which its process
2397 group is a member. A newly created process joins the session of its creator. A process can alter
2398 its session membership; see *setsid()*. There can be multiple process groups in the same session.

2399 **Note:** The *setsid()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.

2400 3.338 Session Leader

2401 A process that has created a session.

2402 **Note:** For further information, see the *setsid()* function defined in the System Interfaces volume of
2403 POSIX.1-200x.

2404 3.339 Session Lifetime

2405 The period between when a session is created and the end of the lifetime of all the process
2406 groups that remain as members of the session.

2407 3.340 Shared Memory Object

2408 An object that represents memory that can be mapped concurrently into the address space of
2409 more than one process.

2410 3.341 Shell

2411 A program that interprets sequences of text input as commands. It may operate on an input
2412 stream or it may interactively prompt and read commands from a terminal.

2413 3.342 Shell, the

2414 The Shell Command Language Interpreter; a specific instance of a shell.

2415 **Note:** For further information, see the *sh* utility defined in the Shell and Utilities volume of
2416 POSIX.1-200x.

2417 3.343 Shell Script

2418 A file containing shell commands. If the file is made executable, it can be executed by specifying
2419 its name as a simple command. Execution of a shell script causes a shell to execute the
2420 commands within the script. Alternatively, a shell can be requested to execute the commands in
2421 a shell script by specifying the name of the shell script as the operand to the *sh* utility.

2422 **Note:** Simple Commands are defined in detail in XCU [Section 2.9.1](#) (on page 2263).

2423 The *sh* utility is defined in detail in the Shell and Utilities volume of POSIX.1-200x.

2424 3.344 Signal

2425 A mechanism by which a process or thread may be notified of, or affected by, an event occurring
2426 in the system. Examples of such events include hardware exceptions and specific actions by
2427 processes. The term signal is also used to refer to the event itself.

2428 3.345 Signal Stack

2429 Memory established for a thread, in which signal handlers catching signals sent to that thread

2430 are executed.

2431 **3.346 Single-Quote**

2432 The character ' ' , also known as apostrophe.

2433 **3.347 Slash**

2434 The character ' / ' , also known as solidus.

2435 **3.348 Socket**

2436 A file of a particular type that is used as a communications endpoint for process-to-process
2437 communication as described in the System Interfaces volume of POSIX.1-200x.

2438 **3.349 Socket Address**

2439 An address associated with a socket or remote endpoint, including an address family identifier
2440 and addressing information specific to that address family. The address may include multiple
2441 parts, such as a network address associated with a host system and an identifier for a specific
2442 endpoint.

2443 **3.350 Soft Limit**

2444 A resource limitation established for each process that the process may set to any value less than
2445 or equal to the hard limit.

2446 **3.351 Source Code**

2447 When dealing with the Shell Command Language, input to the command language interpreter.
2448 The term "shell script" is synonymous with this meaning.

2449 When dealing with an ISO/IEC-conforming programming language, source code is input to a
2450 compiler conforming to that ISO/IEC standard.

2451 Source code also refers to the input statements prepared for the following standard utilities: *awk*,
2452 *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

2453 Source code can also refer to a collection of sources meeting any or all of these meanings.

2454 **Note:** The *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc* utilities are defined in detail in the Shell and
2455 Utilities volume of POSIX.1-200x.

2456

3.352 Space Character (<space>)

2457

2458

2459

The character defined in the portable character set as <space>. The <space> is a member of the **space** character class of the current locale, but represents the single character, and not all of the possible members of the class; see also [Section 3.434](#) (on page 93).

2460

3.353 Spawn

2461

2462

A process creation primitive useful for systems that have difficulty with *fork()* and as an efficient replacement for *fork()/exec*.

2463

3.354 Special Built-In

2464

See *Built-In Utility* in [Section 3.82](#) (on page 44).

2465

3.355 Special Parameter

2466

2467

2468

In the shell command language, a parameter named by a single character from the following list:

* @ # ? ! - \$ 0

Note: For further information, see XCU [Section 2.5.2](#) (on page 2250).

2469

3.356 Spin Lock

2470

A synchronization object used to allow multiple threads to serialize their access to shared data.

2471

3.357 Sporadic Server

2472

2473

A scheduling policy for threads and processes that reserves a certain amount of execution capacity for processing aperiodic events at a given priority level.

2474

3.358 Standard Error

2475

An output stream usually intended to be used for diagnostic messages.

2476

3.359 Standard Input

2477

An input stream usually intended to be used for primary data input.

2478

3.360 Standard Output

2479

An output stream usually intended to be used for primary data output.

2480 3.361 Standard Utilities

2481 The utilities described in the Shell and Utilities volume of POSIX.1-200x.

2482 3.362 Stream

2483 Appearing in lowercase, a stream is a file access object that allows access to an ordered sequence
2484 of characters, as described by the ISO C standard. Such objects can be created by the *fdopen()*,
2485 *fmemopen()*, *fopen()*, *open_memstream()*, or *popen()* functions, and are associated with a file
2486 descriptor. A stream provides the additional services of user-selectable buffering and formatted
2487 input and output; see also [Section 3.363](#).

2488 **Note:** For further information, see XSH [Section 2.5](#) (on page 469).

2489 The *fdopen()*, *fmemopen()*, *fopen()*, *open_memstream()*, and *popen()* functions are defined in detail
2490 in the System Interfaces volume of POSIX.1-200x.

2491 3.363 STREAM

2492 Appearing in uppercase, STREAM refers to a full-duplex connection between a process and an
2493 open device or pseudo-device. It optionally includes one or more intermediate processing
2494 modules that are interposed between the process end of the STREAM and the device driver (or
2495 pseudo-device driver) end of the STREAM; see also [Section 3.362](#).

2496 **Note:** For further information, see XSH [Section 2.6](#) (on page 473).

2497 3.364 STREAM End

2498 The STREAM end is the driver end of the STREAM and is also known as the downstream end of
2499 the STREAM.

2500 3.365 STREAM Head

2501 The STREAM head is the beginning of the STREAM and is at the boundary between the system
2502 and the application process. This is also known as the upstream end of the STREAM.

2503 3.366 STREAMS Multiplexor

2504 A driver with multiple STREAMS connected to it. Multiplexing with STREAMS connected
2505 above is referred to as N-to-1, or “upper multiplexing”. Multiplexing with STREAMS connected
2506 below is referred to as 1-to-N or “lower multiplexing”.

2507 3.367 String

2508 A contiguous sequence of bytes terminated by and including the first null byte.

2509 3.368 Subshell

2510 A shell execution environment, distinguished from the main or current shell execution
2511 environment.

2512 **Note:** For further information, see XCU [Section 2.12](#) (on page 2277). |

2513 3.369 Successfully Transferred

2514 For a write operation to a regular file, when the system ensures that all data written is readable
2515 on any subsequent open of the file (even one that follows a system or power failure) in the
2516 absence of a failure of the physical storage medium.

2517 For a read operation, when an image of the data on the physical storage medium is available to
2518 the requesting process.

2519 3.370 Supplementary Group ID

2520 An attribute of a process used in determining file access permissions. A process has up to
2521 {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The
2522 supplementary group IDs of a process are set to the supplementary group IDs of the parent
2523 process when the process is created.

2524 3.371 Suspended Job

2525 A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the
2526 process group to stop. A suspended job is a background job, but a background job is not
2527 necessarily a suspended job.

2528 3.372 Symbolic Constant

2529 An object-like macro defined with a constant value. +

2530 Unless stated otherwise, the following shall apply to every symbolic constant: +

- 2531 • It expands to a compile-time constant expression with an integer type. +
- 2532 • It may be defined as another type of constant—e.g., an enumeration constant—as well as +
2533 being a macro. +
- 2534 • It need not be usable in `#if` preprocessing directives. +

2535 3.373 Symbolic Link

2536 A type of file with the property that when the file is encountered during pathname resolution, a
2537 string stored by the file is used to modify the pathname resolution. The stored string has a
2538 length of {SYMLINK_MAX} bytes or fewer.

2539 **Note:** Pathname Resolution is defined in detail in [Section 4.12](#) (on page 99).

2540 **3.374 Synchronized Input and Output**

2541 A determinism and robustness improvement mechanism to enhance the data input and output
2542 mechanisms, so that an application can ensure that the data being manipulated is physically
2543 present on secondary mass storage devices.

2544 **3.375 Synchronized I/O Completion**

2545 The state of an I/O operation that has either been successfully transferred or diagnosed as
2546 unsuccessful.

2547 **3.376 Synchronized I/O Data Integrity Completion**

2548 For read, when the operation has been completed or diagnosed if unsuccessful. The read is
2549 complete only when an image of the data has been successfully transferred to the requesting
2550 process. If there were any pending write requests affecting the data to be read at the time that
2551 the synchronized read operation was requested, these write requests are successfully transferred
2552 prior to reading the data.

2553 For write, when the operation has been completed or diagnosed if unsuccessful. The write is
2554 complete only when the data specified in the write request is successfully transferred and all file
2555 system information required to retrieve the data is successfully transferred.

2556 File attributes that are not necessary for data retrieval (access time, modification time, status
2557 change time) need not be successfully transferred prior to returning to the calling process.

2558 **3.377 Synchronized I/O File Integrity Completion**

2559 Identical to a synchronized I/O data integrity completion with the addition that all file
2560 attributes relative to the I/O operation (including access time, modification time, status change
2561 time) are successfully transferred prior to returning to the calling process.

2562 **3.378 Synchronized I/O Operation**

2563 An I/O operation performed on a file that provides the application assurance of the integrity of
2564 its data and files.

2565 **3.379 Synchronous I/O Operation**

2566 An I/O operation that causes the thread requesting the I/O to be blocked from further use of the
2567 processor until that I/O operation completes.

2568 **Note:** A synchronous I/O operation does not imply synchronized I/O data integrity completion or
2569 synchronized I/O file integrity completion.

2570 **3.380 Synchronously-Generated Signal**

2571 A signal that is attributable to a specific thread.

2572 For example, a thread executing an illegal instruction or touching invalid memory causes a

2573 synchronously-generated signal. Being synchronous is a property of how the signal was
2574 generated and not a property of the signal number.

2575 3.381 System

2576 An implementation of POSIX.1-200x.

2577 3.382 System Boot

2578 An unspecified sequence of events that may result in the loss of transitory data; that is, data that
2579 is not saved in permanent storage. For example, message queues, shared memory, semaphores,
2580 and processes.

2581 3.383 System Clock

2582 A clock with at least one second resolution that contains seconds since the Epoch.

2583 3.384 System Console

2584 A device that receives messages sent by the *syslog()* function, and the *fntmsg()* function when
2585 the MM_CONSOLE flag is set.

2586 **Note:** The *syslog()* and *fntmsg()* functions are defined in detail in the System Interfaces volume of
2587 POSIX.1-200x.

2588 3.385 System Crash

2589 An interval initiated by an unspecified circumstance that causes all processes (possibly other +
2590 than special system processes) to be terminated in an undefined manner, after which any +
2591 changes to the state and contents of files created or written to by an application prior to the +
2592 interval are undefined, except as required elsewhere in POSIX.1-200x.

2593 3.386 System Databases

2594 An implementation provides two system databases: the “group database” (see also [Section](#)
2595 [3.186](#), on page 59) and the “user database” (see also [Section 3.427](#), on page 92).

2596 3.387 System Documentation

2597 All documentation provided with an implementation except for the conformance document.
2598 Electronically distributed documents for an implementation are considered part of the system
2599 documentation.

2600 3.388 System Process

2601 An object other than a process executing an application, that is provided by the system and has a

2602 process ID.

2603 **3.389 System Reboot**

2604 See *System Boot* defined in [Section 3.382](#) (on page 86).

2605 **3.390 System Trace Event**

2606 A trace event that is generated by the implementation, in response either to a system-initiated
 2607 action or to an application-requested action, except for a call to *posix_trace_event()*. When
 2608 supported by the implementation, a system-initiated action generates a process-independent
 2609 system trace event and an application-requested action generates a process-dependent system
 2610 trace event. For a system trace event not defined by POSIX.1-200x, the associated trace event
 2611 type identifier is derived from the implementation-defined name for this trace event, and the
 2612 associated data is of implementation-defined content and length.

2613 **3.391 System-Wide**

2614 Pertaining to events occurring in all processes existing in an implementation at a given point in
 2615 time.

2616 **3.392 Tab Character (<tab>)**

2617 A character that in the output stream indicates that printing or displaying should start at the
 2618 next horizontal tabulation position on the current line. It is the character designated by '`\t`' in
 2619 the C language. If the current position is at or past the last defined horizontal tabulation
 2620 position, the behavior is unspecified. It is unspecified whether this character is the exact
 2621 sequence transmitted to an output device by the system to accomplish the tabulation.

2622 **3.393 Terminal (or Terminal Device)**

2623 A character special file that obeys the specifications of the general terminal interface.

2624 **Note:** The General Terminal Interface is defined in detail in [Chapter 11](#) (on page 185).

2625 **3.394 Text Column**

2626 A roughly rectangular block of characters capable of being laid out side-by-side next to other
 2627 text columns on an output page or terminal screen. The widths of text columns are measured in
 2628 column positions.

2629 3.395 Text File

2630 A file that contains characters organized into one or more lines. The lines do not contain NUL
 2631 characters and none can exceed {LINE_MAX} bytes in length, including the <newline>.
 2632 Although POSIX.1-200x does not distinguish between text files and binary files (see the ISO C
 2633 standard), many utilities only produce predictable or meaningful output when operating on text
 2634 files. The standard utilities that have such restrictions always specify “text files” in their STDIN
 2635 or INPUT FILES sections.

2636 3.396 Thread

2637 A single flow of control within a process. Each thread has its own thread ID, scheduling priority
 2638 and policy, *errno* value, thread-specific key/value bindings, and the required system resources to
 2639 support a flow of control. Anything whose address may be determined by a thread, including
 2640 but not limited to static variables, storage obtained via *malloc()*, directly addressable storage
 2641 obtained through implementation-defined functions, and automatic variables, are accessible to
 2642 all threads in the same process.

2643 **Note:** The *malloc()* function is defined in detail in the System Interfaces volume of POSIX.1-200x.

2644 3.397 Thread ID

2645 Each thread in a process is uniquely identified during its lifetime by a value of type **pthread_t**
 2646 called a thread ID.

2647 3.398 Thread List

2648 An ordered set of runnable threads that all have the same ordinal value for their priority.

2649 The ordering of threads on the list is determined by a scheduling policy or policies. The set of
 2650 thread lists includes all runnable threads in the system.

2651 3.399 Thread-Safe

2652 A function that may be safely invoked concurrently by multiple threads. Each function defined
 2653 in the System Interfaces volume of POSIX.1-200x is thread-safe unless explicitly stated
 2654 otherwise. Examples are any “pure” function, a function which holds a mutex locked while it is
 2655 accessing static storage, or objects shared among threads.

2656 3.400 Thread-Specific Data Key

2657 A process global handle of type **pthread_key_t** which is used for naming thread-specific data.

2658 Although the same key value may be used by different threads, the values bound to the key by
 2659 *pthread_setspecific()* and accessed by *pthread_getspecific()* are maintained on a per-thread basis
 2660 and persist for the life of the calling thread.

2661 **Note:** The *pthread_getspecific()* and *pthread_setspecific()* functions are defined in detail in the System
 2662 Interfaces volume of POSIX.1-200x.

- 2663 **3.401 Tilde**
2664 The character '~'.
- 2665 **3.402 Timeouts**
2666 A method of limiting the length of time an interface will block; see also [Section 3.75](#) (on page 43).
- 2667 **3.403 Timer**
2668 A mechanism that can notify a thread when the time as measured by a particular clock has
2669 reached or passed a specified value, or when a specified amount of time has passed.
- 2670 **3.404 Timer Overrun**
2671 A condition that occurs each time a timer, for which there is already an expiration signal queued
2672 to the process, expires.
- 2673 **3.405 Token**
2674 In the shell command language, a sequence of characters that the shell considers as a single unit
2675 when reading input. A token is either an operator or a word.
2676 **Note:** The rules for reading input are defined in detail in XCU [Section 2.3](#) (on page 2247).
- 2677 **3.406 Trace Analyzer Process**
2678 A process that extracts trace events from a trace stream to retrieve information about the
2679 behavior of an application.
- 2680 **3.407 Trace Controller Process**
2681 A process that creates a trace stream for tracing a process.
- 2682 **3.408 Trace Event**
2683 A data object that represents an action executed by the system, and that is recorded in a trace
2684 stream.
- 2685 **3.409 Trace Event Type**
2686 A data object type that defines a class of trace event.

- 2687 **3.410 Trace Event Type Mapping**
2688 A one-to-one mapping between trace event types and trace event names.
- 2689 **3.411 Trace Filter**
2690 A filter that allows the trace controller process to specify those trace event types that are to be
2691 ignored; that is, not generated.
- 2692 **3.412 Trace Generation Version**
2693 A data object that is an implementation-defined character string, generated by the trace system
2694 and describing the origin and version of the trace system.
- 2695 **3.413 Trace Log**
2696 The flushed image of a trace stream, if the trace stream is created with a trace log.
- 2697 **3.414 Trace Point**
2698 An action that may cause a trace event to be generated.
- 2699 **3.415 Trace Stream**
2700 An opaque object that contains trace events plus internal data needed to interpret those trace
2701 events.
- 2702 **3.416 Trace Stream Identifier**
2703 A handle to manage tracing operations in a trace stream.
- 2704 **3.417 Trace System**
2705 A system that allows both system and user trace events to be generated into a trace stream.
2706 These trace events can be retrieved later.
- 2707 **3.418 Traced Process**
2708 A process for which at least one trace stream has been created. A traced process is also called a
2709 target process.
- 2710 **3.419 Tracing Status of a Trace Stream**
2711 A status that describes the state of an active trace stream. The tracing status of a trace stream can
2712 be retrieved from the trace stream attributes. An active trace stream can be in one of two states:

2713 running or suspended.

2714 **3.420 Typed Memory Name Space**

2715 A system-wide name space that contains the names of the typed memory objects present in the
2716 system. It is configurable for a given implementation.

2717 **3.421 Typed Memory Object**

2718 A combination of a typed memory pool and a typed memory port. The entire contents of the
2719 pool are accessible from the port. The typed memory object is identified through a name that
2720 belongs to the typed memory name space.

2721 **3.422 Typed Memory Pool**

2722 An extent of memory with the same operational characteristics. Typed memory pools may be
2723 contained within each other.

2724 **3.423 Typed Memory Port**

2725 A hardware access path to one or more typed memory pools.

2726 **3.424 Unbind**

2727 Remove the association between a network address and an endpoint.

2728 **3.425 Unit Data**

2729 See *Datagram* in [Section 3.123](#) (on page 50).

2730 **3.426 Upshifting**

2731 The conversion of a lowercase character that has a single-character uppercase representation into
2732 this uppercase representation.

2733 3.427 User Database

2734 A system database that contains at least the following information for each user ID:

- 2735 • User name
- 2736 • Numerical user ID
- 2737 • Initial numerical group ID
- 2738 • Initial working directory
- 2739 • Initial user program

2740 The initial numerical group ID is used by the *newgrp* utility. Any other circumstances under
2741 which the initial values are operative are implementation-defined.

2742 If the initial user program field is null, an implementation-defined program is used.

2743 If the initial working directory field is null, the interpretation of that field is implementation-
2744 defined.

2745 **Note:** The *newgrp* utility is defined in detail in the Shell and Utilities volume of POSIX.1-200x.

2746 3.428 User ID

2747 A non-negative integer that is used to identify a system user. When the identity of a user is
2748 associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or
2749 a saved set-user-ID.

2750 3.429 User Name

2751 A string that is used to identify a user; see also [Section 3.427](#). To be portable across systems
2752 conforming to POSIX.1-200x, the value is composed of characters from the portable filename
2753 character set. The hyphen should not be used as the first character of a portable user name.

2754 3.430 User Trace Event

2755 A trace event that is generated explicitly by the application as a result of a call to
2756 *posix_trace_event()*.

2757 3.431 Utility

2758 A program, excluding special built-in utilities provided as part of the Shell Command Language,
2759 that can be called by name from a shell to perform a specific task, or related set of tasks.

2760 **Note:** For further information on special built-in utilities, see XCU [Section 2.14](#) (on page 2280).

2761 3.432 Variable

2762 In the shell command language, a named parameter.

2763 **Note:** For further information, see XCU [Section 2.5](#) (on page 2249).

2764 **3.433 Vertical-Tab Character (<vertical-tab>)**

2765 A character that in the output stream indicates that printing should start at the next vertical
2766 tabulation position. It is the character designated by '`\v`' in the C language. If the current
2767 position is at or past the last defined vertical tabulation position, the behavior is unspecified. It is
2768 unspecified whether this character is the exact sequence transmitted to an output device by the
2769 system to accomplish the tabulation.

2770 **3.434 White Space**

2771 A sequence of one or more characters that belong to the **space** character class as defined via the
2772 `LC_CTYPE` category in the current locale.

2773 In the POSIX locale, white space consists of one or more `<blank>`s (`<space>`s and `<tab>`s),
2774 `<newline>`s, `<carriage-return>`s, `<form-feed>`s, and `<vertical-tab>`s.

2775 **3.435 Wide-Character Code (C Language)**

2776 An integer value corresponding to a single graphic symbol or control code.

2777 **Note:** C Language Wide-Character Codes are defined in detail in [Section 6.3](#) (on page 115).

2778 **3.436 Wide-Character Input/Output Functions**

2779 The functions that perform wide-oriented input from streams or wide-oriented output to
2780 streams: `fgetwc()`, `fgetws()`, `fputwc()`, `fputws()`, `fwprintf()`, `fwscanf()`, `getwc()`, `getwchar()`, `putwc()`,
2781 `putwchar()`, `ungetwc()`, `vwprintf()`, `vwscanf()`, `vwprintf()`, `vwscanf()`, `wprintf()`, and `wscanf()`.

2782 **Note:** These functions are defined in detail in the System Interfaces volume of POSIX.1-200x.

2783 **3.437 Wide-Character String**

2784 A contiguous sequence of wide-character codes terminated by and including the first null wide-
2785 character code.

2786 **3.438 Word**

2787 In the shell command language, a token other than an operator. In some cases a word is also a
 2788 portion of a word token: in the various forms of parameter expansion, such as $\${name-word}$,
 2789 and variable assignment, such as $name=word$, the word is the portion of the token depicted by
 2790 *word*. The concept of a word is no longer applicable following word expansions—only fields
 2791 remain.

2792 **Note:** For further information, see XCU [Section 2.6.2](#) (on page 2254) and [Section 2.6](#) (on page 2253).

2793 **3.439 Working Directory (or Current Working Directory)**

2794 A directory, associated with a process, that is used in pathname resolution for pathnames that do
 2795 not begin with a slash.

2796 **3.440 Worldwide Portability Interface**

2797 Functions for handling characters in a codeset-independent manner.

2798 **3.441 Write**

2799 To output characters to a file, such as standard output or standard error. Unless otherwise stated,
 2800 standard output is the default output destination for all uses of the term “write”; see the
 2801 distinction between display and write in [Section 3.132](#) (on page 51).

2802 **3.442 XSI**

2803 The X/Open System Interfaces (XSI) option is the core application programming interface for C
 2804 and *sh* programming for systems conforming to the Single UNIX Specification. This is a
 2805 superset of the mandatory requirements for conformance to POSIX.1-200x.

2806 **3.443 XSI-Conformant**

2807 A system which allows an application to be built using a set of services that are consistent across
 2808 all systems that conform to POSIX.1-200x and that support the XSI option.

2809 **Note:** See also [Chapter 2](#) (on page 15).

2810 **3.444 Zombie Process**

2811 A process that has terminated and that is deleted when its exit status has been reported to
 2812 another process which is waiting for that process to terminate.

2813 **3.445 ± 0**

2814 The algebraic sign provides additional information about any variable that has the value zero
 2815 when the representation allows the sign to be determined.

2816

2817

2818

For the purposes of POSIX.1-200x, the general concepts given in [Chapter 4](#) apply.

2819

2820

2821

Note: No shading to denote extensions or options occurs in this chapter. Where the terms and definitions given in this chapter are used elsewhere in text related to extensions and options, they are shaded as appropriate.

2822

4.1 Concurrent Execution

2823

2824

Functions that suspend the execution of the calling thread shall not cause the execution of other threads to be indefinitely suspended.

2825

4.2 Directory Protection

2826

2827

If a directory is writable and the mode bit `S_ISVTX` is set on the directory, a process may remove or rename files within that directory only if one or more of the following is true:

2828

- The effective user ID of the process is the same as that of the owner ID of the file.

2829

- The effective user ID of the process is the same as that of the owner ID of the directory.

2830

- The process has appropriate privileges.

2831

If the `S_ISVTX` bit is set on a non-directory file, the behavior is unspecified.

2832

4.3 Extended Security Controls

2833

2834

2835

2836

An implementation may provide implementation-defined extended security controls (see [Section 3.159](#), on page 55). These permit an implementation to provide security mechanisms to implement different security policies than those described in POSIX.1-200x. These mechanisms shall not alter or override the defined semantics of any of the interfaces in POSIX.1-200x.

4.4 File Access Permissions

The standard file access control mechanism uses the file permission bits, as described below.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission bits. An alternate file access control mechanism shall:

- Specify file permission bits for the file owner class, file group class, and file other class of that file, corresponding to the access permissions.
- Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the appropriate privilege.
- Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The disabling of the alternate mechanism need not disable any additional mechanisms supported by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access shall be determined as follows:

- If a process has the appropriate privilege:
 - If read, write, or directory search permission is requested, access shall be granted.
 - If execute permission is requested, access shall be granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access shall be denied.
- Otherwise:
 - The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.
 - Access shall be granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access shall be denied.

POSIX.1-200x does not provide a way to open a directory for searching. It is unspecified whether directory search permission is granted based on the file access modes of the directory's file descriptor or on the mode of the directory at the time the directory is searched.

4.5 File Hierarchy

Files in the system are organized in a hierarchical structure in which all of the non-terminal nodes are directories and all of the terminal nodes are any other type of file. Since multiple directory entries may refer to the same file, the hierarchy is properly described as a "directed graph".

2872 4.6 Filenames

2873 Uppercase and lowercase letters shall retain their unique identities between conforming
2874 implementations.

2875 4.7 Filename Portability

2876 For a filename to be portable across implementations conforming to POSIX.1-200x, it shall
2877 consist only of the portable filename character set as defined in [Section 3.275](#) (on page 71).

2878 Portable filenames shall not have the hyphen character as the first character since this may cause
2879 problems when filenames are passed as command line arguments.

2880 4.8 File Times Update

2881 Each file has three distinct associated timestamps: the time of last data access, the time of last
2882 data modification, and the time the file status last changed. These values are returned in the file
2883 characteristics structure `struct stat`, as described in [<sys/stat.h>](#) (on page 370).

2884 Each function or utility in POSIX.1-200x that reads or writes data or changes file status indicates
2885 which of the appropriate timestamps shall be “marked for update”. If an implementation of
2886 such a function or utility marks for update one of these timestamps in a place or time not
2887 specified by POSIX.1-200x, this shall be documented, except that any changes caused by
2888 pathname resolution need not be documented. For the other functions or utilities in
2889 POSIX.1-200x (those that are not explicitly required to read or write file data or change file
2890 status, but that in some implementations happen to do so), the effect is unspecified.

2891 An implementation may update timestamps that are marked for update immediately, or it may
2892 update such timestamps periodically. At the point in time when an update occurs, any marked
2893 timestamps shall be set to the current time and the update marks shall be cleared. All
2894 timestamps that are marked for update shall be updated when the file ceases to be open by any
2895 process or before a `stat()`, `fstat()`, `lstat()`, `fsync()`, `futimens()`, `utime()`, `utimensat()`, or `utimes()` is
2896 successfully performed on the file. Other times at which updates are done are unspecified.
2897 Marks for update, and updates themselves, shall not be done for files on read-only file systems;
2898 see [Section 3.303](#) (on page 75).

2899 The resolution of timestamps of files in a file system is implementation-defined, but shall be no
2900 coarser than one-second resolution. The three timestamps shall always have values that are
2901 supported by the file system. Whenever any of a file’s timestamps are to be set to a value V
2902 according to the rules of the preceding paragraphs of this section, the implementation shall
2903 immediately set the timestamp to the greatest value supported by the file system that is not
2904 greater than V .

4.9 Host and Network Byte Orders

When data is transmitted over the network, it is sent as a sequence of octets (8-bit unsigned values). If an entity (such as an address or a port number) can be larger than 8 bits, it needs to be stored in several octets. The convention is that all such values are stored with 8 bits in each octet, and with the first (lowest-addressed) octet holding the most-significant bits. This is called “network byte order”.

Network byte order may not be convenient for processing actual values. For this, it is more sensible for values to be stored as ordinary integers. This is known as “host byte order”. In host byte order:

- The most significant bit might not be stored in the first byte in address order.
- Bits might not be allocated to bytes in any obvious order at all.

8-bit values stored in `uint8_t` objects do not require conversion to or from host byte order, as they have the same representation. 16 and 32-bit values can be converted using the `htonl()`, `htons()`, `ntohl()`, and `ntohs()` functions. When reading data that is to be converted to host byte order, it should either be received directly into a `uint16_t` or `uint32_t` object or should be copied from an array of bytes using `memcpy()` or similar. Passing the data through other types could cause the byte order to be changed. Similar considerations apply when sending data.

4.10 Measurement of Execution Time

The mechanism used to measure execution time shall be implementation-defined. The implementation shall also define to whom the CPU time that is consumed by interrupt handlers and system services on behalf of the operating system will be charged. See [Section 3.117](#) (on page 49).

4.11 Memory Synchronization

Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it. Such access is restricted using functions that synchronize thread execution and also synchronize memory with respect to other threads. The following functions synchronize memory with respect to other threads:

<code>fork()</code>	<code>pthread_mutex_trylock()</code>	<code>pthread_rwlock_unlock()</code>
<code>pthread_barrier_wait()</code>	<code>pthread_mutex_unlock()</code>	<code>pthread_rwlock_wrlock()</code>
<code>pthread_cond_broadcast()</code>	<code>pthread_spin_lock()</code>	<code>sem_post()</code>
<code>pthread_cond_signal()</code>	<code>pthread_spin_trylock()</code>	<code>sem_timedwait()</code>
<code>pthread_cond_timedwait()</code>	<code>pthread_spin_unlock()</code>	<code>sem_trywait()</code>
<code>pthread_cond_wait()</code>	<code>pthread_rwlock_rdlock()</code>	<code>sem_wait()</code>
<code>pthread_create()</code>	<code>pthread_rwlock_timedrdlock()</code>	<code>semctl()</code>
<code>pthread_join()</code>	<code>pthread_rwlock_timedwrlock()</code>	<code>semop()</code>
<code>pthread_mutex_lock()</code>	<code>pthread_rwlock_tryrdlock()</code>	<code>wait()</code>
<code>pthread_mutex_timedlock()</code>	<code>pthread_rwlock_trywrlock()</code>	<code>waitpid()</code>

The `pthread_once()` function shall synchronize memory for the first call in each thread for a given `pthread_once_t` object.

The `pthread_mutex_lock()` function need not synchronize memory if the mutex type is `PTHREAD_MUTEX_RECURSIVE` and the calling thread already owns the mutex. The

2947 *pthread_mutex_unlock()* function need not synchronize memory if the mutex type is
2948 PTHREAD_MUTEX_RECURSIVE and the mutex has a lock count greater than one.

2949 Unless explicitly stated otherwise, if one of the above functions returns an error, it is unspecified
2950 whether the invocation causes memory to be synchronized.

2951 Applications may allow more than one thread of control to read a memory location
2952 simultaneously.

2953 4.12 Pathname Resolution

2954 Pathname resolution is performed for a process to resolve a pathname to a particular file in a file
2955 hierarchy. There may be multiple pathnames that resolve to the same file.

2956 Each filename in the pathname is located in the directory specified by its predecessor (for
2957 example, in the pathname fragment *a/b*, file *b* is located in directory *a*). Pathname resolution
2958 shall fail if this cannot be accomplished. If the pathname begins with a slash, the predecessor of
2959 the first filename in the pathname shall be taken to be the root directory of the process (such
2960 pathnames are referred to as “absolute pathnames”). If the pathname does not begin with a
2961 slash, the predecessor of the first filename of the pathname shall be taken to be either the current
2962 working directory of the process or for certain interfaces the directory identified by a file
2963 descriptor passed to the interface (such pathnames are referred to as “relative pathnames”).

2964 The interpretation of a pathname component is dependent on the value of {NAME_MAX} and
2965 _POSIX_NO_TRUNC associated with the path prefix of that component. If any pathname
2966 component is longer than {NAME_MAX}, the implementation shall consider this an error.

2967 A pathname that contains at least one non-slash character and that ends with one or more
2968 trailing slashes shall be resolved as if a single dot character (‘.’) were appended to the
2969 pathname.

2970 If a symbolic link is encountered during pathname resolution, the behavior shall depend on
2971 whether the pathname component is at the end of the pathname and on the function being
2972 performed. If all of the following are true, then pathname resolution is complete:

- 2973 1. This is the last pathname component of the pathname.
- 2974 2. The pathname has no trailing slash.
- 2975 3. The function is required to act on the symbolic link itself, or certain arguments direct that
2976 the function act on the symbolic link itself.

2977 In all other cases, the system shall prefix the remaining pathname, if any, with the contents of the
2978 symbolic link. If the combined length exceeds {PATH_MAX}, and the implementation considers
2979 this to be an error, *errno* shall be set to [ENAMETOOLONG] and an error indication shall be
2980 returned. Otherwise, the resolved pathname shall be the resolution of the pathname just created.
2981 If the resulting pathname does not begin with a slash, the predecessor of the first filename of the
2982 pathname is taken to be the directory containing the symbolic link.

2983 If the system detects a loop in the pathname resolution process, it shall set *errno* to [ELOOP] and
2984 return an error indication. The same may happen if during the resolution process more symbolic
2985 links were followed than the implementation allows. This implementation-defined limit shall
2986 not be smaller than {SYMLOOP_MAX}.

2987 The special filename dot shall refer to the directory specified by its predecessor. The special
2988 filename dot-dot shall refer to the parent directory of its predecessor directory. As a special case,
2989 in the root directory, dot-dot may refer to the root directory itself.

2990 A pathname consisting of a single slash shall resolve to the root directory of the process. A null

2991 pathname shall not be successfully resolved. A pathname that begins with two successive
 2992 slashes may be interpreted in an implementation-defined manner, although more than two
 2993 leading slashes shall be treated as a single slash.

2994 4.13 Process ID Reuse

2995 A process group ID shall not be reused by the system until the process group lifetime ends.

2996 A process ID shall not be reused by the system until the process lifetime ends. In addition, if
 2997 there exists a process group whose process group ID is equal to that process ID, the process ID
 2998 shall not be reused by the system until the process group lifetime ends. A process that is not a
 2999 system process shall not have a process ID of 1.

3000 4.14 Scheduling Policy

3001 A scheduling policy affects process or thread ordering:

- 3002 • When a process or thread is a running thread and it becomes a blocked thread
- 3003 • When a process or thread is a running thread and it becomes a preempted thread
- 3004 • When a process or thread is a blocked thread and it becomes a runnable thread
- 3005 • When a running thread calls a function that can change the priority or scheduling policy of
 3006 a process or thread
- 3007 • In other scheduling policy-defined circumstances

3008 Conforming implementations shall define the manner in which each of the scheduling policies
 3009 may modify the priorities or otherwise affect the ordering of processes or threads at each of the
 3010 occurrences listed above. Additionally, conforming implementations shall define in what other
 3011 circumstances and in what manner each scheduling policy may modify the priorities or affect
 3012 the ordering of processes or threads.

3013 4.15 Seconds Since the Epoch

3014 A value that approximates the number of seconds that have elapsed since the Epoch. A
 3015 Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*),
 3016 hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900
 3017 (*tm_year*)) is related to a time represented as seconds since the Epoch, according to the
 3018 expression below.

3019 If the year is <1970 or the value is negative, the relationship is undefined. If the year is ≥1970 and
 3020 the value is non-negative, the value is related to a Coordinated Universal Time name according
 3021 to the C-language expression, where *tm_sec*, *tm_min*, *tm_hour*, *tm_yday*, and *tm_year* are all
 3022 integer types:

3023
$$tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +$$
 3024
$$(tm_year-70)*31536000 + ((tm_year-69)/4)*86400 -$$
 3025
$$((tm_year-1)/100)*86400 + ((tm_year+299)/400)*86400$$

3026 The relationship between the actual time of day and the current value for seconds since the
 3027 Epoch is unspecified.

3028 How any changes to the value of seconds since the Epoch are made to align to a desired
 3029 relationship with the current actual time is implementation-defined. As represented in seconds

3030 since the Epoch, each and every day shall be accounted for by exactly 86 400 seconds.

3031 **Note:** The last three terms of the expression add in a day for each year that follows a leap year starting
 3032 with the first leap year since the Epoch. The first term adds a day every 4 years starting in 1973,
 3033 the second subtracts a day back out every 100 years starting in 2001, and the third adds a day
 3034 back in every 400 years starting in 2001. The divisions in the formula are integer divisions; that
 3035 is, the remainder is discarded leaving only the integer quotient.

3036 4.16 Semaphore

3037 A minimum synchronization primitive to serve as a basis for more complex synchronization
 3038 mechanisms to be defined by the application program.

3039 For the semaphores associated with the Semaphores option, a semaphore is represented as a
 3040 shareable resource that has a non-negative integer value. When the value is zero, there is a
 3041 (possibly empty) set of threads awaiting the availability of the semaphore.

3042 For the semaphores associated with the X/Open System Interfaces (XSI) option, a semaphore is
 3043 a positive integer (0 through 32767). The *semget()* function can be called to create a set or array of
 3044 semaphores. A semaphore set can contain one or more semaphores up to an implementation-
 3045 defined value.

3046 Semaphore Lock Operation

3047 An operation that is applied to a semaphore. If, prior to the operation, the value of the
 3048 semaphore is zero, the semaphore lock operation shall cause the calling thread to be blocked and
 3049 added to the set of threads awaiting the semaphore; otherwise, the value shall be decremented.

3050 Semaphore Unlock Operation

3051 An operation that is applied to a semaphore. If, prior to the operation, there are any threads in
 3052 the set of threads awaiting the semaphore, then some thread from that set shall be removed from
 3053 the set and becomes unblocked; otherwise, the semaphore value shall be incremented.

3054 4.17 Thread-Safety

3055 Refer to XSH [Section 2.9](#) (on page 485).

3056 4.18 Tracing

3057 The trace system allows a traced process to have a selection of events created for it. Traces
 3058 consist of streams of trace event types.

3059 A trace event type is identified on the one hand by a trace event type name, also referenced as a
 3060 trace event name, and on the other hand by a trace event type identifier. A trace event name is a
 3061 human-readable string. A trace event type identifier is an opaque identifier used by the trace
 3062 system. There shall be a one-to-one relationship between trace event type identifiers and trace
 3063 event names for a given trace stream and also for a given traced process. The trace event type
 3064 identifier shall be generated automatically from a trace event name by the trace system either
 3065 when a trace controller process invokes *posix_trace_trid_eventid_open()* or when an instrumented
 3066 application process invokes *posix_trace_eventid_open()*. Trace event type identifiers are used to
 3067 filter trace event types, to allow interpretation of user data, and to identify the kind of trace
 3068 point that generated a trace event.

3069 Each trace event shall be of a particular trace event type, and associated with a trace event type
 3070 identifier. The execution of a trace point shall generate a trace event if a trace stream has been
 3071 created and started for the process that executed the trace point and if the corresponding trace
 3072 event type identifier is not ignored by filtering.

3073 A generated trace event shall be recorded in a trace stream, and optionally also in a trace log if a
 3074 trace log is associated with the trace stream, except that:

- 3075 • For a trace stream, if no resources are available for the event, the event is lost.
- 3076 • For a trace log, if no resources are available for the event, or a flush operation does not
 3077 succeed, the event is lost.

3078 A trace event recorded in an active trace stream may be retrieved by an application having the
 3079 appropriate privileges.

3080 A trace event recorded in a trace log may be retrieved by an application having the appropriate
 3081 privileges after opening the trace log as a pre-recorded trace stream, with the function
 3082 *posix_trace_open()*.

3083 When a trace event is reported it is possible to retrieve the following:

- 3084 • A trace event type identifier
- 3085 • A timestamp
- 3086 • The process ID of the traced process, if the trace event is process-dependent
- 3087 • Any optional trace event data including its length
- 3088 • If the Threads option is supported, the thread ID, if the trace event is process-dependent
- 3089 • The program address at which the trace point was invoked

3090 Trace events may be mapped from trace event types to trace event names. One such mapping
 3091 shall be associated with each trace stream. An active trace stream is associated with a traced
 3092 process, and also with its children if the Trace Inherit option is supported and also the
 3093 inheritance policy is set to `_POSIX_TRACE_INHERIT`. Therefore each traced process has a
 3094 mapping of the trace event names to trace event type identifiers that have been defined for that
 3095 process.

3096 Traces can be recorded into either trace streams or trace logs.

3097 The implementation and format of a trace stream are unspecified. A trace stream need not be
 3098 and generally is not persistent. A trace stream may be either active or pre-recorded:

- 3099 • An active trace stream is a trace stream that has been created and has not yet been shut
 3100 down. It can be of one of the two following classes:
 - 3101 1. An active trace stream without a trace log that was created with the
 3102 *posix_trace_create()* function
 - 3103 2. If the Trace Log option is supported, an active trace stream with a trace log that was
 3104 created with the *posix_trace_create_withlog()* function
- 3105 • A pre-recorded trace stream is a trace stream that was opened from a trace log object using
 3106 the *posix_trace_open()* function.

3107 An active trace stream can loop. This behavior means that when the resources allocated by the
 3108 trace system for the trace stream are exhausted, the trace system reuses the resources associated
 3109 with the oldest recorded trace events to record new trace events.

3110 If the Trace Log option is supported, an active trace stream with a trace log can be flushed. This

3111 operation causes the trace system to write trace events from the trace stream to the associated
3112 trace log, following the defined policies or using an explicit function call. After this operation,
3113 the trace system may reuse the resources associated with the flushed trace events.

3114 An active trace stream with or without a trace log can be cleared. This operation shall cause all
3115 the resources associated with this trace stream to be reinitialized. The trace stream shall behave
3116 as if it was returning from its creation, except that the mapping of trace event type identifiers to
3117 trace event names shall not be cleared. If a trace log was associated with this trace stream, the
3118 trace log shall also be reinitialized.

3119 A trace log shall be recorded when the *posix_trace_shutdown()* operation is invoked or during
3120 tracing, depending on the tracing strategy which is defined by a log policy. After the trace
3121 stream has been shut down, the trace information can be retrieved from the associated trace log
3122 using the same interface used to retrieve information from an active trace stream.

3123 For a traced process, if the Trace Inherit option is supported and the trace stream's inheritance
3124 attribute is `_POSIX_TRACE_INHERIT`, the initial targeted traced process shall be traced together
3125 with all of its future children. The *posix_pid* member of each trace event in a trace stream shall be
3126 the process ID of the traced process.

3127 Each trace point may be an implementation-defined action such as a context switch, or an
3128 application-programmed action such as a call to a specific operating system service (for
3129 example, *fork()*) or a call to *posix_trace_event()*.

3130 Trace points may be filtered. The operation of the filter is to filter out (ignore) selected trace
3131 events. By default, no trace events are filtered.

3132 The results of the tracing operations can be analyzed and monitored by a trace controller process
3133 or a trace analyzer process.

3134 Only the trace controller process has control of the trace stream it has created. The control of the
3135 operation of a trace stream is done using its corresponding trace stream identifier. The trace
3136 controller process is able to:

- 3137 • Initialize the attributes of a trace stream
- 3138 • Create the trace stream
- 3139 • Start and stop tracing
- 3140 • Know the mapping of the traced process
- 3141 • If the Trace Event Filter option is supported, filter the type of trace events to be recorded
- 3142 • Shut the trace stream down

3143 A traced process may also be a trace controller process. Only the trace controller process can
3144 control its trace stream(s). A trace stream created by a trace controller process shall be shut down
3145 if its controller process terminates or executes another file.

3146 A trace controller process may also be a trace analyzer process. Trace analysis can be done
3147 concurrently with the traced process or can be done off-line, in the same or in a different
3148 platform.

3149 4.19 Treatment of Error Conditions for Mathematical Functions

3150 For all the functions in the `<math.h>` header, an application wishing to check for error situations
 3151 should set `errno` to 0 and call `feclearexcept(FE_ALL_EXCEPT)` before calling the function. On
 3152 return, if `errno` is non-zero or `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW |`
 3153 `FE_UNDERFLOW)` is non-zero, an error has occurred.

3154 The following error conditions are defined for all functions in the `<math.h>` header.

3155 4.19.1 Domain Error

3156 A “domain error” shall occur if an input argument is outside the domain over which the
 3157 mathematical function is defined. The description of each function lists any required domain
 3158 errors; an implementation may define additional domain errors, provided that such errors are
 3159 consistent with the mathematical definition of the function.

3160 On a domain error, the function shall return an implementation-defined value; if the integer
 3161 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to [EDOM]; if
 3162 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “invalid”
 3163 floating-point exception shall be raised.

3164 4.19.2 Pole Error

3165 A “pole error” occurs if the mathematical result of the function is an exact infinity (for example,
 3166 `log(0.0)`).

3167 On a pole error, the function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or
 3168 `HUGE_VALL` according to the return type, with the same sign as the correct value of the
 3169 function; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall
 3170 be set to [ERANGE]; if the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-
 3171 zero, the “divide-by-zero” floating-point exception shall be raised.

3172 4.19.3 Range Error

3173 A “range error” shall occur if the finite mathematical result of the function cannot be
 3174 represented in an object of the specified type, due to extreme magnitude.

3175 4.19.3.1 Result Overflows

3176 A floating result overflows if the magnitude of the mathematical result is finite but so large that
 3177 the mathematical result cannot be represented without extraordinary roundoff error in an object
 3178 of the specified type. If a floating result overflows and default rounding is in effect, then the
 3179 function shall return the value of the macro `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL`
 3180 according to the return type, with the same sign as the correct value of the function; if the integer
 3181 expression `(math_errhandling & MATH_ERRNO)` is non-zero, `errno` shall be set to [ERANGE]; if
 3182 the integer expression `(math_errhandling & MATH_ERREXCEPT)` is non-zero, the “overflow”
 3183 floating-point exception shall be raised.

3184 4.19.3.2 Result Underflows

3185 The result underflows if the magnitude of the mathematical result is so small that the
 3186 mathematical result cannot be represented, without extraordinary roundoff error, in an object of
 3187 the specified type. If the result underflows, the function shall return an implementation-defined
 3188 value whose magnitude is no greater than the smallest normalized positive number in the
 3189 specified type; if the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 3190 whether `errno` is set to [ERANGE] is implementation-defined; if the integer expression

3191 (math_errhandling & MATH_ERREXCEPT) is non-zero, whether the “underflow” floating-point
3192 exception is raised is implementation-defined.

3193 4.20 Treatment of NaN Arguments for the Mathematical Functions

3194 For functions called with a NaN argument, no errors shall occur and a NaN shall be returned,
3195 except where stated otherwise.

3196 If a function with one or more NaN arguments returns a NaN result, the result should be the
3197 same as one of the NaN arguments (after possible type conversion), except perhaps for the sign.

3198 On implementations that support the IEC 60559:1989 standard floating point, functions with
3199 signaling NaN argument(s) shall be treated as if the function were called with an argument that
3200 is a required domain error and shall return a quiet NaN result, except where stated otherwise.

3201 **Note:** The function might never see the signaling NaN, since it might trigger when the arguments are
3202 evaluated during the function call.

3203 On implementations that support the IEC 60559:1989 standard floating point, for those
3204 functions that do not have a documented domain error, the following shall apply:

3205 These functions shall fail if:

3206 Domain Error Any argument is a signaling NaN.

3207 Either, the integer expression (math_errhandling & MATH_ERRNO) is non-zero and *errno*
3208 shall be set to [EDOM], or the integer expression (math_errhandling &
3209 MATH_ERREXCEPT) is non-zero and the invalid floating-point exception shall be raised.

3210 4.21 Utility

3211 A utility program shall be either an executable file, such as might be produced by a compiler or
3212 linker system from computer source code, or a file of shell source code, directly interpreted by
3213 the shell. The program may have been produced by the user, provided by the system
3214 implementor, or acquired from an independent distributor.

3215 The system may implement certain utilities as shell functions (see XCU [Section 2.9.5](#), on page |
3216 2270) or built-in utilities, but only an application that is aware of the command search order (as |
3217 described in XCU [Section 2.9.1.1](#), on page 2264) or of performance characteristics can discern |
3218 differences between the behavior of such a function or built-in utility and that of an executable |
3219 file.

4.22 Variable Assignment

In the shell command language, a word consisting of the following parts:

varname=value

When used in a context where assignment is defined to occur and at no other time, the *value* (representing a word or field) shall be assigned as the value of the variable denoted by *varname*.

Note: For further information, see XCU [Section 2.9.1](#) (on page 2263).

The *varname* and *value* parts shall meet the requirements for a name and a word, respectively, except that they are delimited by the embedded unquoted equals-sign, in addition to other delimiters.

Note: Additional delimiters are described in XCU [Section 2.3](#) (on page 2247).

When a variable assignment is done, the variable shall be created if it did not already exist. If *value* is not specified, the variable shall be given a null value.

Note: An alternative form of variable assignment:

symbol=value

(where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces unspecified results. The form *symbol=value* is used by the KornShell *name[expression]=value* syntax.

File Format Notation

The STDIN, STDOUT, STDERR, INPUT FILES, and OUTPUT FILES sections of the utility descriptions use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the System Interfaces volume of POSIX.1-200x *printf()* function, as described in this chapter. When used in STDIN or INPUT FILES sections of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the System Interfaces volume of POSIX.1-200x *scanf()* function to read the input file.

The description of an individual record is as follows:

"<format>", [*<arg1>*, *<arg2>*, . . . , *<argn>*]

The *format* is a character string that contains three types of objects defined below:

1. *Characters* that are not "escape sequences" or "conversion specifications", as described below, shall be copied to the output.
2. *Escape Sequences* represent non-graphic characters.
3. *Conversion Specifications* specify the output format of each argument; see below.

The following characters have the following special meaning in the format string:

- ' ' (An empty character position.) Represents one or more <blank>s.
- Δ Represents exactly one <space>.

Table 5-1 lists escape sequences and associated actions on display devices capable of the action.

3257

Table 5-1 Escape Sequences and Associated Actions

3258

3259

3260

3261

3262

3263

3264

3265

3266

3267

3268

3269

3270

3271

3272

3273

Escape Sequence	Represents Character	Terminal Action
'\\'	backslash	Print the character '\\ '.
'\a'	alert	Attempt to alert the user through audible or visible notification.
'\b'	backspace	Move the printing position to one column before the current position, unless the current position is the start of a line.
'\f'	form-feed	Move the printing position to the initial printing position of the next logical page.
'\n'	newline	Move the printing position to the start of the next line.
'\r'	carriage-return	Move the printing position to the start of the current line.
'\t'	tab	Move the printing position to the next tab position on the current line. If there are no more tab positions remaining on the line, the behavior is undefined.
'\v'	vertical-tab	Move the printing position to the start of the next vertical tab position. If there are no more vertical tab positions left on the page, the behavior is undefined.

3274

3275

Each conversion specification is introduced by the percent-sign character ('%'). After the character '%', the following shall appear in sequence:

3276

3277

flags Zero or more *flags*, in any order, that modify the meaning of the conversion specification.

3278

3279

3280

3281

field width An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer bytes than the field width, it shall be padded on the left (or right, if the left-adjustment flag ('-'), described below, has been given) to the field width.

3282

3283

3284

3285

3286

3287

3288

precision Gives the minimum number of digits to appear for the *d*, *o*, *i*, *u*, *x*, or *X* conversion specifiers (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversion specifiers, the maximum number of significant digits for the *g* conversion specifier; or the maximum number of bytes to be written from a string in the *s* conversion specifier. The precision shall take the form of a period ('.') followed by a decimal digit string; a null digit string is treated as zero.

3289

3290

3291

conversion specifier characters
A conversion specifier character (see below) that indicates the type of conversion to be applied.

3292

The *flag* characters and their meanings are:

3293

3294

3295

3296

3297

3298

3299

3300

3301

3302

- The result of the conversion shall be left-justified within the field.

+ The result of a signed conversion shall always begin with a sign ('+' or '-').

<space> If the first character of a signed conversion is not a sign, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.

The value shall be converted to an alternative form. For *c*, *d*, *i*, *u*, and *s* conversion specifiers, the behavior is undefined. For the *o* conversion specifier, it shall increase the precision to force the first digit of the result to be a zero. For *x* or *X* conversion specifiers, a non-zero result has 0x or 0X prefixed to it, respectively. For *e*, *E*, *f*, *g*, and *G* conversion specifiers, the result shall always contain a radix

File Format Notation

3303		character, even if no digits follow the radix character. For <code>g</code> and <code>G</code> conversion
3304		specifiers, trailing zeros shall not be removed from the result as they usually are.
3305	0	For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , <code>X</code> , <code>e</code> , <code>E</code> , <code>f</code> , <code>g</code> , and <code>G</code> conversion specifiers, leading zeros (following
3306		any indication of sign or base) shall be used to pad to the field width; no space
3307		padding is performed. If the <code>'0'</code> and <code>'-'</code> flags both appear, the <code>'0'</code> flag shall be
3308		ignored. For <code>d</code> , <code>i</code> , <code>o</code> , <code>u</code> , <code>x</code> , and <code>X</code> conversion specifiers, if a precision is specified, the
3309		<code>'0'</code> flag shall be ignored. For other conversion specifiers, the behavior is
3310		undefined.
3311		Each conversion specifier character shall result in fetching zero or more arguments. The results
3312		are undefined if there are insufficient arguments for the format. If the format is exhausted while
3313		arguments remain, the excess arguments shall be ignored.
3314		The conversion specifiers and their meanings are:
3315	<code>d,i,o,u,x,X</code>	The integer argument shall be written as signed decimal (<code>d</code> or <code>i</code>), unsigned octal
3316		(<code>o</code>), unsigned decimal (<code>u</code>), or unsigned hexadecimal notation (<code>x</code> and <code>X</code>). The <code>d</code> and
3317		<code>i</code> specifiers shall convert to signed decimal in the style <code>"[-]dddd"</code> . The <code>x</code>
3318		conversion specifier shall use the numbers and letters <code>"0123456789abcdef"</code> and
3319		the <code>X</code> conversion specifier shall use the numbers and letters
3320		<code>"0123456789ABCDEF"</code> . The <i>precision</i> component of the argument shall specify
3321		the minimum number of digits to appear. If the value being converted can be
3322		represented in fewer digits than the specified minimum, it shall be expanded with
3323		leading zeros. The default precision shall be 1. The result of converting a zero
3324		value with a precision of 0 shall be no characters. If both the field width and
3325		precision are omitted, the implementation may precede, follow, or precede and
3326		follow numeric arguments of types <code>d</code> , <code>i</code> , and <code>u</code> with <code><blank></code> s; arguments of type <code>o</code>
3327		(octal) may be preceded with leading zeros.
3328	<code>f</code>	The floating-point number argument shall be written in decimal notation in the
3329		style <code>[-]ddd.ddd</code> , where the number of digits after the radix character (shown here
3330		as a decimal point) shall be equal to the <i>precision</i> specification. The <code>LC_NUMERIC</code>
3331		locale category shall determine the radix character to use in this format. If the
3332		<i>precision</i> is omitted from the argument, six digits shall be written after the radix
3333		character; if the <i>precision</i> is explicitly 0, no radix character shall appear.
3334	<code>e,E</code>	The floating-point number argument shall be written in the style <code>[-]d.ddde±dd</code> (the
3335		symbol <code>'±'</code> indicates either a plus or minus sign), where there is one digit before
3336		the radix character (shown here as a decimal point) and the number of digits after
3337		it is equal to the precision. The <code>LC_NUMERIC</code> locale category shall determine the
3338		radix character to use in this format. When the precision is missing, six digits shall
3339		be written after the radix character; if the precision is 0, no radix character shall
3340		appear. The <code>E</code> conversion specifier shall produce a number with <code>E</code> instead of <code>e</code>
3341		introducing the exponent. The exponent shall always contain at least two digits.
3342		However, if the value to be written requires an exponent greater than two digits,
3343		additional exponent digits shall be written as necessary.
3344	<code>g,G</code>	The floating-point number argument shall be written in style <code>f</code> or <code>e</code> (or in style <code>F</code>
3345		or <code>E</code> in the case of a <code>G</code> conversion specifier), with the precision specifying the
3346		number of significant digits. The style used depends on the value converted: style
3347		<code>e</code> (or <code>E</code>) shall be used only if the exponent resulting from the conversion is less
3348		than <code>-4</code> or greater than or equal to the precision. Trailing zeros are removed from
3349		the result. A radix character shall appear only if it is followed by a digit.

- 3350 c The integer argument shall be converted to an **unsigned char** and the resulting
3351 byte shall be written.
- 3352 s The argument shall be taken to be a string and bytes from the string shall be
3353 written until the end of the string or the number of bytes indicated by the *precision*
3354 specification of the argument is reached. If the precision is omitted from the
3355 argument, it shall be taken to be infinite, so all bytes up to the end of the string
3356 shall be written.
- 3357 % Write a '%' character; no argument is converted.
- 3358 In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of
3359 a conversion is wider than the field width, the field is simply expanded to contain the
3360 conversion result. The term "field width" should not be confused with the term "precision"
3361 used in the description of %s.

3362 Examples

3363 To represent the output of a program that prints a date and time in the form Sunday, July 3,
3364 10:02, where *weekday* and *month* are strings:

3365 "%s, Δ%sΔ%d, Δ%d: %.2d\n" <weekday>, <month>, <day>, <hour>, <min>

3366 To show 'π' written to 5 decimal places:

3367 "piΔ=Δ%.5f\n", <value of π>

3368 To show an input file format consisting of five colon-separated fields:

3369 "%s:%s:%s:%s:%s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

6.1 Portable Character Set

Conforming implementations shall support one or more coded character sets. Each supported locale shall include the *portable character set*, which is the set of symbolic names for characters in Table 6-1. This is used to describe characters within the text of POSIX.1-200x. The first eight entries in Table 6-1 are defined in the ISO/IEC 6429:1992 standard and the rest of the characters are defined in the ISO/IEC 10646-1:2000 standard.

Table 6-1 Portable Character Set

Symbolic Name	Glyph	UCS	Description
<NUL>		<U0000>	NULL (NUL)
<alert>		<U0007>	BELL (BEL)
<backspace>		<U0008>	BACKSPACE (BS)
<tab>		<U0009>	CHARACTER TABULATION (HT)
<carriage-return>		<U000D>	CARRIAGE RETURN (CR)
<newline>		<U000A>	LINE FEED (LF)
<vertical-tab>		<U000B>	LINE TABULATION (VT)
<form-feed>		<U000C>	FORM FEED (FF)
<space>		<U0020>	SPACE
<exclamation-mark>	!	<U0021>	EXCLAMATION MARK
<quotation-mark>	"	<U0022>	QUOTATION MARK
<number-sign>	#	<U0023>	NUMBER SIGN
<dollar-sign>	\$	<U0024>	DOLLAR SIGN
<percent-sign>	%	<U0025>	PERCENT SIGN
<ampersand>	&	<U0026>	AMPERSAND
<apostrophe>	'	<U0027>	APOSTROPHE
<left-parenthesis>	(<U0028>	LEFT PARENTHESIS
<right-parenthesis>)	<U0029>	RIGHT PARENTHESIS
<asterisk>	*	<U002A>	ASTERISK
<plus-sign>	+	<U002B>	PLUS SIGN
<comma>	,	<U002C>	COMMA
<hyphen-minus>	-	<U002D>	HYPHEN-MINUS
<hyphen>	-	<U002D>	HYPHEN-MINUS
<full-stop>	.	<U002E>	FULL STOP
<period>	.	<U002E>	FULL STOP
<slash>	/	<U002F>	SOLIDUS
<solidus>	/	<U002F>	SOLIDUS
<zero>	0	<U0030>	DIGIT ZERO
<one>	1	<U0031>	DIGIT ONE
<two>	2	<U0032>	DIGIT TWO
<three>	3	<U0033>	DIGIT THREE

	Symbolic Name	Glyph	UCS	Description
3411				
3412	<four>	4	<U0034>	DIGIT FOUR
3413	<five>	5	<U0035>	DIGIT FIVE
3414	<six>	6	<U0036>	DIGIT SIX
3415	<seven>	7	<U0037>	DIGIT SEVEN
3416	<eight>	8	<U0038>	DIGIT EIGHT
3417	<nine>	9	<U0039>	DIGIT NINE
3418	<colon>	:	<U003A>	COLON
3419	<semicolon>	;	<U003B>	SEMICOLON
3420	<less-than-sign>	<	<U003C>	LESS-THAN SIGN
3421	<equals-sign>	=	<U003D>	EQUALS SIGN
3422	<greater-than-sign>	>	<U003E>	GREATER-THAN SIGN
3423	<question-mark>	?	<U003F>	QUESTION MARK
3424	<commercial-at>	@	<U0040>	COMMERCIAL AT
3425	<A>	A	<U0041>	LATIN CAPITAL LETTER A
3426		B	<U0042>	LATIN CAPITAL LETTER B
3427	<C>	C	<U0043>	LATIN CAPITAL LETTER C
3428	<D>	D	<U0044>	LATIN CAPITAL LETTER D
3429	<E>	E	<U0045>	LATIN CAPITAL LETTER E
3430	<F>	F	<U0046>	LATIN CAPITAL LETTER F
3431	<G>	G	<U0047>	LATIN CAPITAL LETTER G
3432	<H>	H	<U0048>	LATIN CAPITAL LETTER H
3433	<I>	I	<U0049>	LATIN CAPITAL LETTER I
3434	<J>	J	<U004A>	LATIN CAPITAL LETTER J
3435	<K>	K	<U004B>	LATIN CAPITAL LETTER K
3436	<L>	L	<U004C>	LATIN CAPITAL LETTER L
3437	<M>	M	<U004D>	LATIN CAPITAL LETTER M
3438	<N>	N	<U004E>	LATIN CAPITAL LETTER N
3439	<O>	O	<U004F>	LATIN CAPITAL LETTER O
3440	<P>	P	<U0050>	LATIN CAPITAL LETTER P
3441	<Q>	Q	<U0051>	LATIN CAPITAL LETTER Q
3442	<R>	R	<U0052>	LATIN CAPITAL LETTER R
3443	<S>	S	<U0053>	LATIN CAPITAL LETTER S
3444	<T>	T	<U0054>	LATIN CAPITAL LETTER T
3445	<U>	U	<U0055>	LATIN CAPITAL LETTER U
3446	<V>	V	<U0056>	LATIN CAPITAL LETTER V
3447	<W>	W	<U0057>	LATIN CAPITAL LETTER W
3448	<X>	X	<U0058>	LATIN CAPITAL LETTER X
3449	<Y>	Y	<U0059>	LATIN CAPITAL LETTER Y
3450	<Z>	Z	<U005A>	LATIN CAPITAL LETTER Z
3451	<left-square-bracket>	[<U005B>	LEFT SQUARE BRACKET
3452	<backslash>	\	<U005C>	REVERSE SOLIDUS
3453	<reverse-solidus>	\	<U005C>	REVERSE SOLIDUS
3454	<right-square-bracket>]	<U005D>	RIGHT SQUARE BRACKET
3455	<circumflex-accent>	^	<U005E>	CIRCUMFLEX ACCENT
3456	<circumflex>	^	<U005E>	CIRCUMFLEX ACCENT
3457	<low-line>	_	<U005F>	LOW LINE
3458	<underscore>	_	<U005F>	LOW LINE
3459	<grave-accent>	`	<U0060>	GRAVE ACCENT
3460	<a>	a	<U0061>	LATIN SMALL LETTER A
3461		b	<U0062>	LATIN SMALL LETTER B
3462	<c>	c	<U0063>	LATIN SMALL LETTER C

	Symbolic Name	Glyph	UCS	Description
3463				
3464	<d>	d	<U0064>	LATIN SMALL LETTER D
3465	<e>	e	<U0065>	LATIN SMALL LETTER E
3466	<f>	f	<U0066>	LATIN SMALL LETTER F
3467	<g>	g	<U0067>	LATIN SMALL LETTER G
3468	<h>	h	<U0068>	LATIN SMALL LETTER H
3469	<i>	i	<U0069>	LATIN SMALL LETTER I
3470	<j>	j	<U006A>	LATIN SMALL LETTER J
3471	<k>	k	<U006B>	LATIN SMALL LETTER K
3472	<l>	l	<U006C>	LATIN SMALL LETTER L
3473	<m>	m	<U006D>	LATIN SMALL LETTER M
3474	<n>	n	<U006E>	LATIN SMALL LETTER N
3475	<o>	o	<U006F>	LATIN SMALL LETTER O
3476	<p>	p	<U0070>	LATIN SMALL LETTER P
3477	<q>	q	<U0071>	LATIN SMALL LETTER Q
3478	<r>	r	<U0072>	LATIN SMALL LETTER R
3479	<s>	s	<U0073>	LATIN SMALL LETTER S
3480	<t>	t	<U0074>	LATIN SMALL LETTER T
3481	<u>	u	<U0075>	LATIN SMALL LETTER U
3482	<v>	v	<U0076>	LATIN SMALL LETTER V
3483	<w>	w	<U0077>	LATIN SMALL LETTER W
3484	<x>	x	<U0078>	LATIN SMALL LETTER X
3485	<y>	y	<U0079>	LATIN SMALL LETTER Y
3486	<z>	z	<U007A>	LATIN SMALL LETTER Z
3487	<left-brace>	{	<U007B>	LEFT CURLY BRACKET
3488	<left-curly-bracket>	{	<U007B>	LEFT CURLY BRACKET
3489	<vertical-line>		<U007C>	VERTICAL LINE
3490	<right-brace>	}	<U007D>	RIGHT CURLY BRACKET
3491	<right-curly-bracket>	}	<U007D>	RIGHT CURLY BRACKET
3492	<tilde>	~	<U007E>	TILDE

3493 POSIX.1-200x uses character names other than the above, but only in an informative way; for
 3494 example, in examples to illustrate the use of characters beyond the portable character set with
 3495 the facilities of POSIX.1-200x.

3496 **Table 6-1** (on page 111) defines the characters in the portable character set and the corresponding
 3497 symbolic character names used to identify each character in a character set description file. The
 3498 table contains more than one symbolic character name for characters whose traditional name
 3499 differs from the chosen name. Characters defined in **Table 6-2** (on page 116) may also be used in
 3500 character set description files.

3501 POSIX.1-200x places only the following requirements on the encoded values of the characters in
 3502 the portable character set:

- 3503 • If the encoded values associated with each member of the portable character set are not
 3504 invariant across all locales supported by the implementation, if an application accesses any
 3505 pair of locales where the character encodings differ, or accesses data from an application
 3506 running in a locale which has different encodings from the application's current locale, the
 3507 results are unspecified.
- 3508 • The encoded values associated with the digits 0 to 9 shall be such that the value of each
 3509 character after 0 shall be one greater than the value of the previous character.
- 3510 • A null character, NUL, which has all bits set to zero, shall be in the set of characters.

- 3511 • The encoded values associated with the members of the portable character set are each
3512 represented in a single byte. Moreover, if the value is stored in an object of C-language
3513 type **char**, it is guaranteed to be positive (except the NUL, which is always zero).

3514 Conforming implementations shall support certain character and character set attributes, as
3515 defined in [Section 7.2](#) (on page 122).

3516 6.2 Character Encoding

3517 The POSIX locale contains the characters in [Table 6-1](#) (on page 111), which have the properties
3518 listed in [Section 7.3.1](#) (on page 124). In other locales, the presence, meaning, and representation
3519 of any additional characters are locale-specific.

3520 In locales other than the POSIX locale, a character may have a state-dependent encoding. There
3521 are two types of these encodings:

- 3522 • A single-shift encoding (where each character not in the initial shift state is preceded by a
3523 shift code) can be defined if each shift-code and character sequence is considered a multi-
3524 byte character. This is done using the concatenated-constant format in a character set
3525 description file, as described in [Section 6.4](#) (on page 115). If the implementation supports a
3526 character encoding of this type, all of the standard utilities in the Shell and Utilities volume
3527 of POSIX.1-200x shall support it. Use of a single-shift encoding with any of the functions in
3528 the System Interfaces volume of POSIX.1-200x that do not specifically mention the effects
3529 of state-dependent encoding is implementation-defined.
- 3530 • A locking-shift encoding (where the state of the character is determined by a shift code
3531 that may affect more than the single character following it) cannot be defined with the
3532 current character set description file format. Use of a locking-shift encoding with any of
3533 the standard utilities in the Shell and Utilities volume of POSIX.1-200x or with any of the
3534 functions in the System Interfaces volume of POSIX.1-200x that do not specifically mention
3535 the effects of state-dependent encoding is implementation-defined.

3536 While in the initial shift state, all characters in the portable character set shall retain their usual
3537 interpretation and shall not alter the shift state. The interpretation for subsequent bytes in the
3538 sequence shall be a function of the current shift state. A byte with all bits zero shall be
3539 interpreted as the null character independent of shift state. Such a byte shall not occur as part of
3540 any other character.

3541 The maximum allowable number of bytes in a character in the current locale shall be indicated
3542 by {MB_CUR_MAX}, defined in the `<stdlib.h>` header and by the `<mb_cur_max>` value in a
3543 character set description file; see [Section 6.4](#) (on page 115). The implementation's maximum
3544 number of bytes in a character shall be defined by the C-language macro {MB_LEN_MAX}.

6.3 C Language Wide-Character Codes

In the shell, the standard utilities are written so that the encodings of characters are described by the locale's `LC_CTYPE` definition (see [Section 7.3.1](#), on page 124) and there is no differentiation between characters consisting of single octets (8-bit bytes) or multiple bytes. However, in the C language, a differentiation is made. To ease the handling of variable length characters, the C language has introduced the concept of wide-character codes.

All wide-character codes in a given process consist of an equal number of bits. This is in contrast to characters, which can consist of a variable number of bytes. The byte or byte sequence that represents a character can also be represented as a wide-character code. Wide-character codes thus provide a uniform size for manipulating text data. A wide-character code having all bits zero is the null wide-character code (see [Section 3.245](#), on page 67), and terminates wide-character strings (see [Section 3.435](#), on page 93). The wide-character value for each member of the portable character set shall equal its value when used as the lone character in an integer character constant. Wide-character codes for other characters are locale and implementation-defined. State shift bytes shall not have a wide-character code representation. POSIX.1-200x provides no means of defining a wide-character codeset.

6.4 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in POSIX.1-200x as *charmap* files. It is implementation-defined whether or not users or applications can provide additional character set description files.

POSIX.1-200x does not require that multiple character sets or codesets be supported. Although multiple charmap files are supported, it is the responsibility of the implementation to provide the file or files; if only one is provided, only that one is accessible using the *localedef* utility's `-f` option.

Each character set description file, except those that use the ISO/IEC 10646-1:2000 standard position values as the encoding values, shall define characteristics for the coded character set and the encoding for the characters specified in [Table 6-1](#) (on page 111), and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in [Table 6-1](#) (on page 111) shall be included in the file and shall be mapped to a unique coding value, except as noted below. The glyphs `'{', '}', '_-', '\/', '\', ' . ', and '^'` have more than one symbolic name; all symbolic names for each such glyph shall be included, each with identical encoding. If some or all of the control characters identified in [Table 6-2](#) (on page 116) are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the encodings associated with the symbolic names in [Table 6-2](#) (on page 116) may be the same as characters found in [Table 6-1](#) (on page 111); both names shall be provided for each encoding.

3584

Table 6-2 Control Character Set

3585
3586
3587
3588
3589
3590

<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>
<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>
<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>
<CAN>		<ETB>	<IS1>	<RS>	<SYN>
<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>
<DC1>		<FF>	<IS3>	<SO>	<VT>

3591
3592
3593

The following declarations can precede the character definitions. Each shall consist of the symbol shown in the following list, starting in column 1, including the surrounding brackets, followed by one or more <blank>s, followed by the value to be assigned to the symbol.

3594
3595
3596

<code_set_name> The name of the coded character set for which the character set description file is defined. The characters of the name shall be taken from the set of characters with visible glyphs defined in Table 6-1 (on page 111).

3597
3598

<mb_cur_max> The maximum number of bytes in a multi-byte character. This shall default to 1.

3599
3600
3601

XSI

<mb_cur_min> An unsigned positive integer value that defines the minimum number of bytes in a character for the encoded character set. On XSI-conformant systems, <mb_cur_min> shall always be 1.

3602
3603
3604
3605

<escape_char> The character used to indicate that the characters following shall be interpreted in a special way, as defined later in this section. This shall default to backslash ('\''), which is the character used in all the following text and examples, unless otherwise noted.

3606
3607
3608

<comment_char> The character that, when placed in column 1 of a charmap line, is used to indicate that the line shall be ignored. The default character shall be the number sign ('#').

3609
3610
3611
3612
3613
3614

The character set mapping definitions shall be all the lines immediately following an identifier line containing the string "CHARMAP" starting in column 1, and preceding a trailer line containing the string "END CHARMAP" starting in column 1. Empty lines and lines containing a <comment_char> in the first column shall be ignored. Each non-comment line of the character set mapping definition (that is, between the "CHARMAP" and "END CHARMAP" lines of the file) shall be in either of two forms:

3615
3616

```
"%s %s %s\n", <symbolic-name>, <encoding>, <comments>
```

or:

3617
3618

```
"%s...%s %s %s\n", <symbolic-name>, <symbolic-name>,  
  <encoding>, <comments>
```

3619
3620
3621
3622
3623

In the first format, the line in the character set mapping definition shall define a single symbolic name and a corresponding encoding. A symbolic name is one or more characters from the set shown with visible glyphs in Table 6-1 (on page 111), enclosed between angle brackets. A character following an escape character is interpreted as itself; for example, the sequence "<\\>" represents the symbolic name ">" enclosed between angle brackets.

3624
3625
3626
3627
3628

In the second format, the line in the character set mapping definition shall define a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more non-numeric characters from the set shown with visible glyphs in Table 6-1 (on page 111), followed by an integer formed by one or more decimal digits. Both integers shall contain the same number of digits. The characters preceding the integer shall be identical in the two symbolic

3629 names, and the integer formed by the digits in the second symbolic name shall be equal to or
 3630 greater than the integer formed by the digits in the first name. This shall be interpreted as a
 3631 series of symbolic names formed from the common part and each of the integers between the
 3632 first and the second integer, inclusive. As an example, <j0101>...<j0104> is interpreted as the
 3633 symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

3634 A character set mapping definition line shall exist for all symbolic names specified in Table 6-1
 3635 (on page 111), and shall define the coded character value that corresponds to the character
 3636 indicated in the table, or the coded character value that corresponds to the control character
 3637 symbolic name. If the control characters commonly associated with the symbolic names in Table
 3638 6-2 (on page 116) are supported by the implementation, the symbolic name and the
 3639 corresponding encoding value shall be included in the file. Additional unique symbolic names
 3640 may be included. A coded character value can be represented by more than one symbolic name.

3641 The encoding part is expressed as one (for single-byte character values) or more concatenated
 3642 decimal, octal, or hexadecimal constants in the following formats:

```
3643 "%cd%u", <escape_char>, <decimal byte value>
3644 "%cx%x", <escape_char>, <hexadecimal byte value>
3645 "%c%o", <escape_char>, <octal byte value>
```

3646 Decimal constants shall be represented by two or three decimal digits, preceded by the escape
 3647 character and the lowercase letter 'd'; for example, "\d05", "\d97", or "\d143".
 3648 Hexadecimal constants shall be represented by two hexadecimal digits, preceded by the escape
 3649 character and the lowercase letter 'x'; for example, "\x05", "\x61", or "\x8f". Octal
 3650 constants shall be represented by two or three octal digits, preceded by the escape character;
 3651 for example, "\05", "\141", or "\217". In a portable charmap file, each constant represents an
 3652 8-bit byte. When constants are concatenated for multi-byte character values, they shall be of the
 3653 same type, and interpreted in sequence from first to last with the first byte of the multi-
 3654 byte character specified by the first byte in the sequence. The manner in which these constants
 3655 are represented in the character stored in the system is implementation-defined. (This notation
 3656 was chosen for reasons of portability. There is no requirement that the internal representation in
 3657 the computer memory be in this same order.) Omitting bytes from a multi-byte character
 3658 definition produces undefined results.

3659 In lines defining ranges of symbolic names, the encoded value shall be the value for the first
 3660 symbolic name in the range (the symbolic name preceding the ellipsis). Subsequent symbolic
 3661 names defined by the range shall have encoding values in increasing order. Bytes shall be
 3662 treated as unsigned octets, and carry shall be propagated between the bytes as necessary to
 3663 represent the range. However, because this causes a null byte in the second or subsequent bytes
 3664 of a character, such a declaration should not be specified. For example, the line:

```
3665 <j0101>...<j0104> \d129\d254
```

3666 is interpreted as:

```
3667 <j0101>          \d129\d254
3668 <j0102>          \d129\d255
3669 <j0103>          \d130\d00
3670 <j0104>          \d130\d01
```

3671 The expanded declaration of the symbol <j0103> in the above example is an invalid
 3672 specification, because it contains a null byte in the second byte of a character.

3673 The comment is optional.

3674 POSIX.1-200x provides no means of defining a wide-character codeset. |

3675 The following declarations can follow the character set mapping definitions (after the "END

3676 CHARMAP" statement). Each shall consist of the keyword shown in the following list, starting in
3677 column 1, followed by the value(s) to be associated to the keyword, as defined below.

3678 **WIDTH** A non-negative integer value defining the column width (see [Section 3.102](#), on
3679 page 47) for the printable characters in the coded character set specified in [Table](#)
3680 [6-1](#) (on page 111) and [Table 6-2](#) (on page 116). Coded character set character values
3681 shall be defined using symbolic character names followed by column width
3682 values. Defining a character with more than one **WIDTH** produces undefined
3683 results. The **END WIDTH** keyword shall be used to terminate the **WIDTH**
3684 definitions. Specifying the width of a non-printable character in a **WIDTH**
3685 declaration produces undefined results.

3686 **WIDTH_DEFAULT**
3687 A non-negative integer value defining the default column width for any printable
3688 character not listed by one of the **WIDTH** keywords. If no **WIDTH_DEFAULT**
3689 keyword is included in the charmap, the default character width shall be 1.

3690 Example

3691 After the "END CHARMAP" statement, a syntax for a width definition would be:

```
3692 WIDTH
3693 <A> 1
3694 <B> 1
3695 <C>...<Z> 1
3696 ...
3697 <fool>...<foon> 2
3698 ...
3699 END WIDTH
```

3700 In this example, the numerical code point values represented by the symbols **<A>** and **** are
3701 assigned a width of 1. The code point values **<C>** to **<Z>** inclusive (**<C>**, **<D>**, **<E>**, and so on)
3702 are also assigned a width of 1. Using **<A>...<Z>** would have required fewer lines, but the
3703 alternative was shown to demonstrate flexibility. The keyword **WIDTH_DEFAULT** could have
3704 been added as appropriate.

3705 6.4.1 State-Dependent Character Encodings

3706 This section addresses the use of state-dependent character encodings (that is, those in which the
3707 encoding of a character is dependent on one or more shift codes that may precede it).

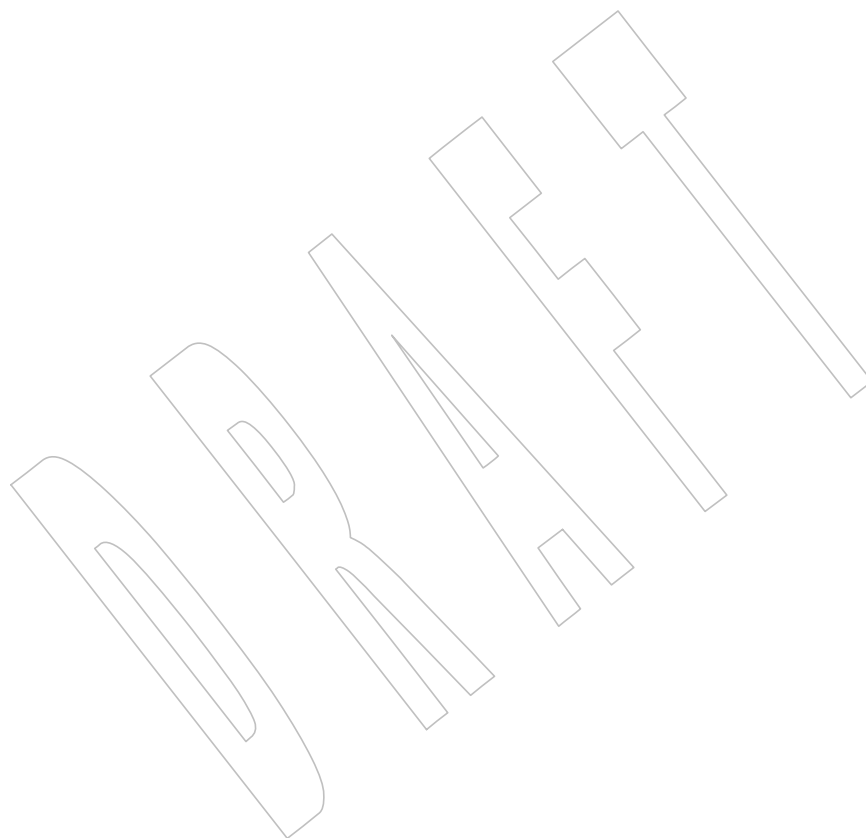
3708 A single-shift encoding (where each character not in the initial shift state is preceded by a shift
3709 code) can be defined in the charmap format if each shift-code/character sequence is considered
3710 a multi-byte character, defined using the concatenated-constant format described in [Section 6.4](#)
3711 (on page 115). If the implementation supports a character encoding of this type, all of the
3712 standard utilities shall support it. A locking-shift encoding (where the state of the character is
3713 determined by a shift code that may affect more than the single character following it) could be
3714 defined with an extension to the charmap format described in [Section 6.4](#) (on page 115). If the
3715 implementation supports a character encoding of this type, any of the standard utilities that
3716 describe character (*versus* byte) or text-file manipulation shall have the following characteristics:

- 3717 1. The utility shall process the statefully encoded data as a concatenation of state-
3718 independent characters. The presence of redundant locking shifts shall not affect the
3719 comparison of two statefully encoded strings.

3720
3721
3722

2. A utility that divides, truncates, or extracts substrings from statefully encoded data shall produce output that contains locking shifts at the beginning or end of the resulting data, if appropriate, to retain correct state information.





7.1 General

A locale is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE Character classification and case conversion.

LC_COLLATE Collation order.

LC_MONETARY Monetary formatting.

LC_NUMERIC Numeric, non-monetary formatting.

LC_TIME Date and time formats.

LC_MESSAGES Formats of informative and diagnostic messages and interactive responses.

The standard utilities in the Shell and Utilities volume of POSIX.1-200x shall base their behavior on the current locale, as defined in the ENVIRONMENT VARIABLES section for each utility. The behavior of some of the C-language functions defined in the System Interfaces volume of POSIX.1-200x shall also be modified based on the current locale, as defined by the last call to *setlocale()*.

Locales other than those supplied by the implementation can be created via the *localedef* utility, provided that the *_POSIX2_LOCALEDEF* symbol is defined on the system. Even if *localedef* is not provided, all implementations conforming to the System Interfaces volume of POSIX.1-200x shall provide one or more locales that behave as described in this chapter. The input to the utility is described in Section 7.3 (on page 122). The value that is used to specify a locale when using environment variables shall be the string specified as the *name* operand to the *localedef* utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX locale (see Section 7.2, on page 122). When the value of a locale environment variable begins with a slash ('/'), it shall be interpreted as the pathname of the locale definition; the type of file (regular, directory, and so on) used to store the locale definition is implementation-defined. If the value does not begin with a slash, the mechanism used to locate the locale is implementation-defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories are undefined. Likewise, if different codesets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the codeset is different from the codeset assumed when the locale was created, the result is also undefined.

Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) with the appropriate value. If the function is invoked with an empty string, such as:

```
setlocale(LC_ALL, "");
```

the value of the corresponding environment variable is used. If the environment variable is unset or is set to the empty string, the implementation shall set the appropriate environment as

3762 defined in [Chapter 8](#) (on page 159).

3763 7.2 POSIX Locale

3764 Conforming systems shall provide a POSIX locale, also known as the C locale. The behavior of
3765 standard utilities and functions in the POSIX locale shall be as if the locale was defined via the
3766 *localedef* utility with input data from the POSIX locale tables in [Section 7.3](#).

3767 The tables in [Section 7.3](#) describe the characteristics and behavior of the POSIX locale for data
3768 consisting entirely of characters from the portable character set and the control character set. For
3769 other characters, the behavior is unspecified. For C-language programs, the POSIX locale shall
3770 be the default locale when the *setlocale()* function is not called.

3771 The POSIX locale can be specified by assigning to the appropriate environment variables the
3772 values "C" or "POSIX".

3773 All implementations shall define a locale as the default locale, to be invoked when no
3774 environment variables are set, or set to the empty string. This default locale can be the POSIX
3775 locale or any other implementation-defined locale. Some implementations may provide facilities
3776 for local installation administrators to set the default locale, customizing it for each location.
3777 POSIX.1-200x does not require such a facility.

3778 7.3 Locale Definition

3779 The capability to specify additional locales to those provided by an implementation is optional,
3780 denoted by the `_POSIX2_LOCALEDEF` symbol. If the option is not supported, only
3781 implementation-supplied locales are available. Such locales shall be documented using the
3782 format specified in this section.

3783 Locales can be described with the file format presented in this section. The file format is that
3784 accepted by the *localedef* utility. For the purposes of this section, the file is referred to as the
3785 "locale definition file", but no locales shall be affected by this file unless it is processed by
3786 *localedef* or some similar mechanism. Any requirements in this section imposed upon the utility
3787 shall apply to *localedef* or to any other similar utility used to install locale information using the
3788 locale definition file format described here.

3789 The locale definition file shall contain one or more locale category source definitions, and shall
3790 not contain more than one definition for the same locale category. If the file contains source
3791 definitions for more than one category, implementation-defined categories, if present, shall
3792 appear after the categories defined by [Section 7.1](#) (on page 121). A category source definition
3793 contains either the definition of a category or a **copy** directive. For a description of the **copy**
3794 directive, see *localedef*. In the event that some of the information for a locale category, as
3795 specified in this volume of POSIX.1-200x, is missing from the locale source definition, the
3796 behavior of that category, if it is referenced, is unspecified.

3797 A category source definition shall consist of a category header, a category body, and a category
3798 trailer. A category header shall consist of the character string naming of the category, beginning
3799 with the characters `LC_`. The category trailer shall consist of the string "END", followed by one
3800 or more <blank>s and the string used in the corresponding category header.

3801 The category body shall consist of one or more lines of text. Each line shall contain an identifier,
3802 optionally followed by one or more operands. Identifiers shall be either keywords, identifying a
3803 particular locale element, or collating elements. In addition to the keywords defined in this
3804 volume of POSIX.1-200x, the source can contain implementation-defined keywords. Each
3805 keyword within a locale shall have a unique name (that is, two categories cannot have a

3806 commonly-named keyword); no keyword shall start with the characters *LC_*. Identifiers shall be
 3807 separated from the operands by one or more <blank>s.

3808 Operands shall be characters, collating elements, or strings of characters. Strings shall be
 3809 enclosed in double-quotes. Literal double-quotes within strings shall be preceded by the <escape
 3810 character>, described below. When a keyword is followed by more than one operand, the
 3811 operands shall be separated by semicolons; <blank>s shall be allowed both before and after a
 3812 semicolon.

3813 The first category header in the file can be preceded by a line modifying the comment character.
 3814 It shall have the following format, starting in column 1:

```
3815 "comment_char %c\n", <comment character>
```

3816 The comment character shall default to the number sign ('#'). Blank lines and lines containing
 3817 the <comment character> in the first position shall be ignored.

3818 The first category header in the file can be preceded by a line modifying the escape character to
 3819 be used in the file. It shall have the following format, starting in column 1:

```
3820 "escape_char %c\n", <escape character>
```

3821 The escape character shall default to backslash, which is the character used in all examples
 3822 shown in this volume of POSIX.1-200x.

3823 A line can be continued by placing an escape character as the last character on the line; this
 3824 continuation character shall be discarded from the input. Although the implementation need not
 3825 accept any one portion of a continued line with a length exceeding {LINE_MAX} bytes, it shall
 3826 place no limits on the accumulated length of the continued line. Comment lines shall not be
 3827 continued on a subsequent line using an escaped <newline>.

3828 Individual characters, characters in strings, and collating elements shall be represented using
 3829 symbolic names, as defined below. In addition, characters can be represented using the
 3830 characters themselves or as octal, hexadecimal, or decimal constants. When non-symbolic
 3831 notation is used, the resultant locale definitions are in many cases not portable between systems.
 3832 The left angle bracket ('<') is a reserved symbol, denoting the start of a symbolic name; when
 3833 used to represent itself it shall be preceded by the escape character. The following rules apply to
 3834 character representation:

- 3835 1. A character can be represented via a symbolic name, enclosed within angle brackets '<'
 3836 and '>'. The symbolic name, including the angle brackets, shall exactly match a
 3837 symbolic name defined in the charmap file specified via the *localedef -f* option, and it shall
 3838 be replaced by a character value determined from the value associated with the symbolic
 3839 name in the charmap file. The use of a symbolic name not found in the charmap file shall
 3840 constitute an error, unless the category is *LC_CTYPE* or *LC_COLLATE*, in which case it
 3841 shall constitute a warning condition (see *localedef* for a description of actions resulting
 3842 from errors and warnings). The specification of a symbolic name in a **collating-element**
 3843 or **collating-symbol** section that duplicates a symbolic name in the charmap file (if
 3844 present) shall be an error. Use of the escape character or a right angle bracket within a
 3845 symbolic name is invalid unless the character is preceded by the escape character.

3846 For example:

```
3847 <c>; <c-cedilla> " <M><a><y> "
```

- 3848 2. A character in the portable character set can be represented by the character itself, in
 3849 which case the value of the character is implementation-defined. (Implementations may
 3850 allow other characters to be represented as themselves, but such locale definitions are not
 3851 portable.) Within a string, the double-quote character, the escape character, and the right

3852 angle bracket character shall be escaped (preceded by the escape character) to be
3853 interpreted as the character itself. Outside strings, the characters:

3854 `, ; < > escape_char`

3855 shall be escaped to be interpreted as the character itself.

3856 For example:

3857 `c "May"`

3858 3. A character can be represented as an octal constant. An octal constant shall be specified as
3859 the escape character followed by two or three octal digits. Each constant shall represent a
3860 byte value. Multi-byte values can be represented by concatenated constants specified in
3861 byte order with the last constant specifying the least significant byte of the character.

3862 For example:

3863 `\143;\347;\143\150 "\115\141\171"`

3864 4. A character can be represented as a hexadecimal constant. A hexadecimal constant shall
3865 be specified as the escape character followed by an 'x' followed by two hexadecimal
3866 digits. Each constant shall represent a byte value. Multi-byte values can be represented by
3867 concatenated constants specified in byte order with the last constant specifying the least
3868 significant byte of the character.

3869 For example:

3870 `\x63;\xe7;\x63\x68 "\x4d\x61\x79"`

3871 5. A character can be represented as a decimal constant. A decimal constant shall be
3872 specified as the escape character followed by a 'd' followed by two or three decimal
3873 digits. Each constant represents a byte value. Multi-byte values can be represented by
3874 concatenated constants specified in byte order with the last constant specifying the least
3875 significant byte of the character.

3876 For example:

3877 `\d99;\d231;\d99\d104 "\d77\d97\d121"`

3878 Implementations may accept single-digit octal, decimal, or hexadecimal constants following the
3879 escape character. Only characters existing in the character set for which the locale definition is
3880 created shall be specified, whether using symbolic names, the characters themselves, or octal,
3881 decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the
3882 charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not
3883 present in the charmap file can be specified and shall be ignored, as specified under item 1
3884 above.

3885 7.3.1 LC_CTYPE

3886 The *LC_CTYPE* category shall define character classification, case conversion, and other
3887 character attributes. In addition, a series of characters can be represented by three adjacent
3888 periods representing an ellipsis symbol ("..."). The ellipsis specification shall be interpreted
3889 as meaning that all values between the values preceding and following it represent valid
3890 characters. The ellipsis specification shall be valid only within a single encoded character set;
3891 that is, within a group of characters of the same size. An ellipsis shall be interpreted as including
3892 in the list all characters with an encoded value higher than the encoded value of the character
3893 preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

3894 For example:

3895 `\x30; . . . ; \x39;`

3896 includes in the character class all characters with encoded values between the endpoints.

3897 The following keywords shall be recognized. In the descriptions, the term “automatically
3898 included” means that it shall not be an error either to include or omit any of the referenced
3899 characters; the implementation provides them if missing (even if the entire keyword is missing)
3900 and accepts them silently if present. When the implementation automatically includes a missing
3901 character, it shall have an encoded value dependent on the charmap file in effect (see the
3902 description of the *localedef* `-f` option); otherwise, it shall have a value derived from an
3903 implementation-defined character mapping.

3904 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included
3905 characters. These only need to be specified if the character values (that is, encoding) differ from
3906 the implementation default values. It is not possible to define a locale without these
3907 automatically included characters unless some implementation extension is used to prevent
3908 their inclusion. Such a definition would not be a proper superset of the C or POSIX locale and,
3909 thus, it might not be possible for conforming applications to work properly.

3910 **copy** Specify the name of an existing locale which shall be used as the definition of
3911 this category. If this keyword is specified, no other keyword shall be specified.

3912 **upper** Define characters to be classified as uppercase letters.

3913 In the POSIX locale, the 26 uppercase letters shall be included:

3914 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

3915 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
3916 **punct**, or **space** shall be specified. The uppercase letters <A> to <Z>, as
3917 defined in [Section 6.4](#) (on page 115) (the portable character set), are
3918 automatically included in this class.

3919 **lower** Define characters to be classified as lowercase letters.

3920 In the POSIX locale, the 26 lowercase letters shall be included:

3921 a b c d e f g h i j k l m n o p q r s t u v w x y z

3922 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
3923 **punct**, or **space** shall be specified. The lowercase letters <a> to <z> of the
3924 portable character set are automatically included in this class.

3925 **alpha** Define characters to be classified as letters.

3926 In the POSIX locale, all characters in the classes **upper** and **lower** shall be
3927 included.

3928 In a locale definition file, no character specified for the keywords **cntrl**, **digit**,
3929 **punct**, or **space** shall be specified. Characters classified as either **upper** or
3930 **lower** are automatically included in this class.

3931 **digit** Define the characters to be classified as numeric digits.

3932 In the POSIX locale, only:

3933 0 1 2 3 4 5 6 7 8 9

3934 shall be included.

3935 In a locale definition file, only the digits <zero>, <one>, <two>, <three>,

3936		<four>, <five>, <six>, <seven>, <eight>, and <nine> shall be specified, and in
3937		contiguous ascending sequence by numerical value. The digits <zero> to
3938		<nine> of the portable character set are automatically included in this class.
3939	alnum	Define characters to be classified as letters and numeric digits. Only the
3940		characters specified for the alpha and digit keywords shall be specified.
3941		Characters specified for the keywords alpha and digit are automatically
3942		included in this class.
3943	space	Define characters to be classified as white-space characters.
3944		In the POSIX locale, exactly <space>, <form-feed>, <newline>, <carriage-
3945		return>, <tab>, and <vertical-tab> shall be included.
3946		In a locale definition file, no character specified for the keywords upper ,
3947		lower , alpha , digit , graph , or xdigit shall be specified. The <space>, <form-
3948		feed>, <newline>, <carriage-return>, <tab>, and <vertical-tab> of the portable
3949		character set, and any characters included in the class blank are automatically
3950		included in this class.
3951	cntrl	Define characters to be classified as control characters.
3952		In the POSIX locale, no characters in classes alpha or print shall be included.
3953		In a locale definition file, no character specified for the keywords upper ,
3954		lower , alpha , digit , punct , graph , print , or xdigit shall be specified.
3955	punct	Define characters to be classified as punctuation characters.
3956		In the POSIX locale, neither the <space> nor any characters in classes alpha ,
3957		digit , or cntrl shall be included.
3958		In a locale definition file, no character specified for the keywords upper ,
3959		lower , alpha , digit , cntrl , xdigit , or as the <space> shall be specified.
3960	graph	Define characters to be classified as printable characters, not including the
3961		<space>.
3962		In the POSIX locale, all characters in classes alpha , digit , and punct shall be
3963		included; no characters in class cntrl shall be included.
3964		In a locale definition file, characters specified for the keywords upper , lower ,
3965		alpha , digit , xdigit , and punct are automatically included in this class. No
3966		character specified for the keyword cntrl shall be specified.
3967	print	Define characters to be classified as printable characters, including the
3968		<space>.
3969		In the POSIX locale, all characters in class graph shall be included; no
3970		characters in class cntrl shall be included.
3971		In a locale definition file, characters specified for the keywords upper , lower ,
3972		alpha , digit , xdigit , punct , graph , and the <space> are automatically included
3973		in this class. No character specified for the keyword cntrl shall be specified.
3974	xdigit	Define the characters to be classified as hexadecimal digits.
3975		In the POSIX locale, only:
3976		0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f
3977		shall be included.

3978		In a locale definition file, only the characters defined for the class digit shall be specified, in contiguous ascending sequence by numerical value, followed by one or more sets of six characters representing the hexadecimal digits 10 to 15 inclusive, with each set in ascending order (for example, <A>, , <C>, <D>, <E>, <F>, <a>, , <c>, <d>, <e>, <f>). The digits <zero> to <nine>, the uppercase letters <A> to <F>, and the lowercase letters <a> to <f> of the portable character set are automatically included in this class.
3979		
3980		
3981		
3982		
3983		
3984		
3985	blank	Define characters to be classified as <blank>s.
3986		In the POSIX locale, only the <space> and <tab> shall be included.
3987		In a locale definition file, the <space> and <tab> are automatically included in this class.
3988		
3989	charclass	Define one or more locale-specific character class names as strings separated by semicolons. Each named character class can then be defined subsequently in the <i>LC_CTYPE</i> definition. A character class name shall consist of at least one and at most {CHARCLASS_NAME_MAX} bytes of alphanumeric characters from the portable filename character set. The first character of a character class name shall not be a digit. The name shall not match any of the <i>LC_CTYPE</i> keywords defined in this volume of POSIX.1-200x. Future versions of this standard will not specify any <i>LC_CTYPE</i> keywords containing uppercase letters.
3990		
3991		
3992		
3993		
3994		
3995		
3996		
3997		
3998	<i>charclass-name</i>	Define characters to be classified as belonging to the named locale-specific character class. In the POSIX locale, locale-specific named character classes need not exist.
3999		
4000		
4001		If a class name is defined by a charclass keyword, but no characters are subsequently assigned to it, this is not an error; it represents a class without any characters belonging to it.
4002		
4003		
4004		The <i>charclass-name</i> can be used as the <i>property</i> argument to the <i>wctype()</i> function, in regular expression and shell pattern-matching bracket expressions, and by the <i>tr</i> command.
4005		
4006		
4007	toupper	Define the mapping of lowercase letters to uppercase letters.
4008		In the POSIX locale, at a minimum, the 26 lowercase characters:
4009		a b c d e f g h i j k l m n o p q r s t u v w x y z
4010		shall be mapped to the corresponding 26 uppercase characters:
4011		A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
4012		In a locale definition file, the operand shall consist of character pairs, separated by semicolons. The characters in each character pair shall be separated by a comma and the pair enclosed by parentheses. The first character in each pair is the lowercase letter, the second the corresponding uppercase letter. Only characters specified for the keywords lower and upper shall be specified. The lowercase letters <a> to <z>, and their corresponding uppercase letters <A> to <Z>, of the portable character set are automatically included in this mapping, but only when the toupper keyword is omitted from the locale definition.
4013		
4014		
4015		
4016		
4017		
4018		
4019		
4020		
4021	tolower	Define the mapping of uppercase letters to lowercase letters.
4022		In the POSIX locale, at a minimum, the 26 uppercase characters:

4023 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 4024 shall be mapped to the corresponding 26 lowercase characters:
 4025 a b c d e f g h i j k l m n o p q r s t u v w x y z

4026 In a locale definition file, the operand shall consist of character pairs,
 4027 separated by semicolons. The characters in each character pair shall be
 4028 separated by a comma and the pair enclosed by parentheses. The first
 4029 character in each pair is the uppercase letter, the second the corresponding
 4030 lowercase letter. Only characters specified for the keywords **lower** and **upper**
 4031 shall be specified. If the **tolower** keyword is omitted from the locale definition,
 4032 the mapping is the reverse mapping of the one specified for **toupper**.

4033 The following table shows the character class combinations allowed:

4034 **Table 7-1** Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
4037 upper	—	—	A	x	x	x	x	A	A	—	x
4038 lower	—	—	A	x	x	x	x	A	A	—	x
4039 alpha	—	—	—	x	x	x	x	A	A	—	x
4040 digit	x	x	x	x	x	x	x	A	A	A	x
4041 space	x	x	x	x	—	—	*	*	*	x	—
4042 cntrl	x	x	x	x	—	—	x	x	x	x	—
4043 punct	x	x	x	x	—	x	—	A	A	x	—
4044 graph	—	—	—	—	—	x	—	—	A	—	—
4045 print	—	—	—	—	—	x	—	—	—	—	—
4046 xdigit	—	—	—	—	x	x	x	A	A	—	x
4047 blank	x	x	x	x	A	—	*	*	*	x	—

4048 **Notes:**

- 4049 1. Explanation of codes:
 4050 A Automatically included; see text.
 4051 — Permitted.
 4052 x Mutually-exclusive.
 4053 * See note 2.
- 4054 2. The <space>, which is part of the **space** and **blank** classes, cannot belong to **punct** or
 4055 **graph**, but shall automatically belong to the **print** class. Other **space** or **blank** characters
 4056 can be classified as any of **punct**, **graph**, or **print**.

4057 **7.3.1.1 LC_CTYPE Category in the POSIX Locale**

4058 The character classifications for the POSIX locale follow; the code listing depicts the *localedef*
 4059 input, and the table represents the same information, sorted by character.

```

4060 LC_CTYPE
4061 # The following is the POSIX locale LC_CTYPE.
4062 # "alpha" is by default "upper" and "lower"
4063 # "alnum" is by definition "alpha" and "digit"
4064 # "print" is by default "alnum", "punct", and the <space>
4065 # "graph" is by default "alnum" and "punct"
4066 #
```



```

4067     upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
4068           <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
4069     #
4070     lower    <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
4071           <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
4072     #
4073     digit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
4074           <seven>;<eight>;<nine>
4075     #
4076     space    <tab>;<newline>;<vertical-tab>;<form-feed>;\
4077           <carriage-return>;<space>
4078     #
4079     cntrl    <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
4080           <form-feed>;<carriage-return>;\
4081           <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
4082           <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
4083           <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
4084           <IS1>;<DEL>
4085     #
4086     punct    <exclamation-mark>;<quotation-mark>;<number-sign>;\
4087           <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
4088           <left-parenthesis>;<right-parenthesis>;<asterisk>;\
4089           <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
4090           <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
4091           <greater-than-sign>;<question-mark>;<commercial-at>;\
4092           <left-square-bracket>;<backslash>;<right-square-bracket>;\
4093           <circumflex>;<underscore>;<grave-accent>;<left-curly-bracket>;\
4094           <vertical-line>;<right-curly-bracket>;<tilde>
4095     #
4096     xdigit    <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;\
4097           <eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
4098     #
4099     blank    <space>;<tab>
4100     #
4101     toupper   (<a>, <A>); (<b>, <B>); (<c>, <C>); (<d>, <D>); (<e>, <E>); \
4102           (<f>, <F>); (<g>, <G>); (<h>, <H>); (<i>, <I>); (<j>, <J>); \
4103           (<k>, <K>); (<l>, <L>); (<m>, <M>); (<n>, <N>); (<o>, <O>); \
4104           (<p>, <P>); (<q>, <Q>); (<r>, <R>); (<s>, <S>); (<t>, <T>); \
4105           (<u>, <U>); (<v>, <V>); (<w>, <W>); (<x>, <X>); (<y>, <Y>); (<z>, <Z>)
4106     #
4107     tolower   (<A>, <a>); (<B>, <b>); (<C>, <c>); (<D>, <d>); (<E>, <e>); \
4108           (<F>, <f>); (<G>, <g>); (<H>, <h>); (<I>, <i>); (<J>, <j>); \
4109           (<K>, <k>); (<L>, <l>); (<M>, <m>); (<N>, <n>); (<O>, <o>); \
4110           (<P>, <p>); (<Q>, <q>); (<R>, <r>); (<S>, <s>); (<T>, <t>); \
4111           (<U>, <u>); (<V>, <v>); (<W>, <w>); (<X>, <x>); (<Y>, <y>); (<Z>, <z>)
4112     END LC_CTYPE

```

	Symbolic Name	Other Case	Character Classes
4113			
4114	<NUL>		cntrl
4115	<SOH>		cntrl
4116	<STX>		cntrl
4117	<ETX>		cntrl
4118	<EOT>		cntrl
4119	<ENQ>		cntrl
4120	<ACK>		cntrl
4121	<alert>		cntrl
4122	<backspace>		cntrl
4123	<tab>		cntrl, space, blank
4124	<newline>		cntrl, space
4125	<vertical-tab>		cntrl, space
4126	<form-feed>		cntrl, space
4127	<carriage-return>		cntrl, space
4128	<SO>		cntrl
4129	<SI>		cntrl
4130	<DLE>		cntrl
4131	<DC1>		cntrl
4132	<DC2>		cntrl
4133	<DC3>		cntrl
4134	<DC4>		cntrl
4135	<NAK>		cntrl
4136	<SYN>		cntrl
4137	<ETB>		cntrl
4138	<CAN>		cntrl
4139			cntrl
4140	<SUB>		cntrl
4141	<ESC>		cntrl
4142	<IS4>		cntrl
4143	<IS3>		cntrl
4144	<IS2>		cntrl
4145	<IS1>		cntrl
4146	<space>		space, print, blank
4147	<exclamation-mark>		punct, print, graph
4148	<quotation-mark>		punct, print, graph
4149	<number-sign>		punct, print, graph
4150	<dollar-sign>		punct, print, graph
4151	<percent-sign>		punct, print, graph
4152	<ampersand>		punct, print, graph
4153	<apostrophe>		punct, print, graph
4154	<left-parenthesis>		punct, print, graph
4155	<right-parenthesis>		punct, print, graph
4156	<asterisk>		punct, print, graph
4157	<plus-sign>		punct, print, graph
4158	<comma>		punct, print, graph
4159	<hyphen>		punct, print, graph
4160	<period>		punct, print, graph
4161	<slash>		punct, print, graph
4162	<zero>		digit, xdigit, print, graph
4163	<one>		digit, xdigit, print, graph
4164	<two>		digit, xdigit, print, graph

	Symbolic Name	Other Case	Character Classes
4165			
4166	<three>		digit, xdigit, print, graph
4167	<four>		digit, xdigit, print, graph
4168	<five>		digit, xdigit, print, graph
4169	<six>		digit, xdigit, print, graph
4170	<seven>		digit, xdigit, print, graph
4171	<eight>		digit, xdigit, print, graph
4172	<nine>		digit, xdigit, print, graph
4173	<colon>		punct, print, graph
4174	<semicolon>		punct, print, graph
4175	<less-than-sign>		punct, print, graph
4176	<equals-sign>		punct, print, graph
4177	<greater-than-sign>		punct, print, graph
4178	<question-mark>		punct, print, graph
4179	<commercial-at>		punct, print, graph
4180	<A>	<a>	upper, xdigit, alpha, print, graph
4181			upper, xdigit, alpha, print, graph
4182	<C>	<c>	upper, xdigit, alpha, print, graph
4183	<D>	<d>	upper, xdigit, alpha, print, graph
4184	<E>	<e>	upper, xdigit, alpha, print, graph
4185	<F>	<f>	upper, xdigit, alpha, print, graph
4186	<G>	<g>	upper, alpha, print, graph
4187	<H>	<h>	upper, alpha, print, graph
4188	<I>	<i>	upper, alpha, print, graph
4189	<J>	<j>	upper, alpha, print, graph
4190	<K>	<k>	upper, alpha, print, graph
4191	<L>	<l>	upper, alpha, print, graph
4192	<M>	<m>	upper, alpha, print, graph
4193	<N>	<n>	upper, alpha, print, graph
4194	<O>	<o>	upper, alpha, print, graph
4195	<P>	<p>	upper, alpha, print, graph
4196	<Q>	<q>	upper, alpha, print, graph
4197	<R>	<r>	upper, alpha, print, graph
4198	<S>	<s>	upper, alpha, print, graph
4199	<T>	<t>	upper, alpha, print, graph
4200	<U>	<u>	upper, alpha, print, graph
4201	<V>	<v>	upper, alpha, print, graph
4202	<W>	<w>	upper, alpha, print, graph
4203	<X>	<x>	upper, alpha, print, graph
4204	<Y>	<y>	upper, alpha, print, graph
4205	<Z>	<z>	upper, alpha, print, graph
4206	<left-square-bracket>		punct, print, graph
4207	<backslash>		punct, print, graph
4208	<right-square-bracket>		punct, print, graph
4209	<circumflex>		punct, print, graph
4210	<underscore>		punct, print, graph
4211	<grave-accent>		punct, print, graph
4212	<a>	<A>	lower, xdigit, alpha, print, graph
4213			lower, xdigit, alpha, print, graph
4214	<c>	<C>	lower, xdigit, alpha, print, graph
4215	<d>	<D>	lower, xdigit, alpha, print, graph
4216	<e>	<E>	lower, xdigit, alpha, print, graph

	Symbolic Name	Other Case	Character Classes
4217			
4218	<f>	<F>	lower, xdigit, alpha, print, graph
4219	<g>	<G>	lower, alpha, print, graph
4220	<h>	<H>	lower, alpha, print, graph
4221	<i>	<I>	lower, alpha, print, graph
4222	<j>	<J>	lower, alpha, print, graph
4223	<k>	<K>	lower, alpha, print, graph
4224	<l>	<L>	lower, alpha, print, graph
4225	<m>	<M>	lower, alpha, print, graph
4226	<n>	<N>	lower, alpha, print, graph
4227	<o>	<O>	lower, alpha, print, graph
4228	<p>	<P>	lower, alpha, print, graph
4229	<q>	<Q>	lower, alpha, print, graph
4230	<r>	<R>	lower, alpha, print, graph
4231	<s>	<S>	lower, alpha, print, graph
4232	<t>	<T>	lower, alpha, print, graph
4233	<u>	<U>	lower, alpha, print, graph
4234	<v>	<V>	lower, alpha, print, graph
4235	<w>	<W>	lower, alpha, print, graph
4236	<x>	<X>	lower, alpha, print, graph
4237	<y>	<Y>	lower, alpha, print, graph
4238	<z>	<Z>	lower, alpha, print, graph
4239	<left-curly-bracket>		punct, print, graph
4240	<vertical-line>		punct, print, graph
4241	<right-curly-bracket>		punct, print, graph
4242	<tilde>		punct, print, graph
4243			cntrl

7.3.2 LC_COLLATE

The `LC_COLLATE` category provides a collation sequence definition for numerous utilities in the Shell and Utilities volume of POSIX.1-200x (*sort*, *uniq*, and so on), regular expression matching (see [Chapter 9](#), on page 167), and the *strcoll()*, *strxfrm()*, *wcscoll()*, and *wcsxfrm()* functions in the System Interfaces volume of POSIX.1-200x.

A collation sequence definition shall define the relative order between collating elements (characters and multi-character collating elements) in the locale. This order is expressed in terms of collation values; that is, by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale behaves as if such assignment is done and used in the collation process. At least the following capabilities are provided:

1. **Multi-character collating elements.** Specification of multi-character collating elements (that is, sequences of two or more characters to be collated as an entity).
2. **User-defined ordering of collating elements.** Each collating element shall be assigned a collation value defining its order in the character (or basic) collation sequence. This ordering is used by regular expressions and pattern matching and, unless collation weights are explicitly specified, also as the collation weight to be used in sorting.
3. **Multiple weights and equivalence classes.** Collating elements can be assigned one or more (up to the limit `[COLL_WEIGHTS_MAX]`), as defined in [<limits.h>](#) collating weights for use in sorting. The first weight is hereafter referred to as the primary weight.

- 4265 4. **One-to-many mapping.** A single character is mapped into a string of collating elements.
- 4266 5. **Equivalence class definition.** Two or more collating elements have the same collation
- 4267 value (primary weight).
- 4268 6. **Ordering by weights.** When two strings are compared to determine their relative order,
- 4269 the two strings are first broken up into a series of collating elements; the elements in each
- 4270 successive pair of elements are then compared according to the relative primary weights
- 4271 for the elements. If equal, and more than one weight has been assigned, then the pairs of
- 4272 collating elements are re-compared according to the relative subsequent weights, until
- 4273 either a pair of collating elements compare unequal or the weights are exhausted.

4274 The following keywords shall be recognized in a collation sequence definition. They are

4275 described in detail in the following sections.

4276	copy	Specify the name of an existing locale which shall be used as the
4277		definition of this category. If this keyword is specified, no other keyword
4278		shall be specified.
4279	collating-element	Define a collating-element symbol representing a multi-character
4280		collating element. This keyword is optional.
4281	collating-symbol	Define a collating symbol for use in collation order statements. This
4282		keyword is optional.
4283	order_start	Define collation rules. This statement shall be followed by one or more
4284		collation order statements, assigning character collation values and
4285		collation weights to collating elements.
4286	order_end	Specify the end of the collation-order statements.

4287 7.3.2.1 *The collating-element Keyword*

4288 In addition to the collating elements in the character set, the **collating-element** keyword can be

4289 used to define multi-character collating elements. The syntax is as follows:

4290 `"collating-element %s from \"%s\"\\n", <collating-symbol>, <string>`

4291 The `<collating-symbol>` operand shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. The string operand is a string of two or more characters that collates as an entity. A `<collating-element>` defined via this keyword is only recognized with the `LC_COLLATE` category.

4296 For example:

```
4297 collating-element <ch> from "<c><h>"
4298 collating-element <e-acute> from "<acute><e>"
4299 collating-element <ll> from "ll"
```

4300 7.3.2.2 *The collating-symbol Keyword*

4301 This keyword shall be used to define symbols for use in collation sequence statements; that is,

4302 between the **order_start** and the **order_end** keywords. The syntax is as follows:

4303 `"collating-symbol %s\\n", <collating-symbol>`

4304 The `<collating-symbol>` shall be a symbolic name, enclosed between angle brackets ('<' and '>'), and shall not duplicate any symbolic name in the current charmap file (if any), or any other symbolic name defined in this collation definition. A `<collating-symbol>` defined via this keyword is only recognized within the `LC_COLLATE` category.

4308 For example:

```
4309 collating-symbol <UPPER_CASE>
4310 collating-symbol <HIGH>
```

4311 The **collating-symbol** keyword defines a symbolic name that can be associated with a relative
4312 position in the character order sequence. While such a symbolic name does not represent any
4313 collating element, it can be used as a weight.

4314 7.3.2.3 The *order_start* Keyword

4315 The **order_start** keyword shall precede collation order entries and also define the number of
4316 weights for this collation sequence definition and other collation rules. The syntax is as follows:

```
4317 "order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

4318 The operands to the **order_start** keyword are optional. If present, the operands define rules to be
4319 applied when strings are compared. The number of operands define how many weights each
4320 element is assigned; if no operands are present, one **forward** operand is assumed. If present, the
4321 first operand defines rules to be applied when comparing strings using the first (primary)
4322 weight; the second when comparing strings using the second weight, and so on. Operands shall
4323 be separated by semicolons (';'). Each operand shall consist of one or more collation
4324 directives, separated by commas (','). If the number of operands exceeds the
4325 {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following
4326 directives shall be supported:

4327 **forward** Specifies that comparison operations for the weight level shall proceed from start
4328 of string towards the end of string.

4329 **backward** Specifies that comparison operations for the weight level shall proceed from end of
4330 string towards the beginning of string.

4331 **position** Specifies that comparison operations for the weight level shall consider the relative
4332 position of elements in the strings not subject to **IGNORE**. The string containing
4333 an element not subject to **IGNORE** after the fewest collating elements subject to
4334 **IGNORE** from the start of the compare shall collate first. If both strings contain a
4335 character not subject to **IGNORE** in the same relative position, the collating values
4336 assigned to the elements shall determine the ordering. In case of equality,
4337 subsequent characters not subject to **IGNORE** shall be considered in the same
4338 manner.

4339 The directives **forward** and **backward** are mutually-exclusive.

4340 If no operands are specified, a single **forward** operand shall be assumed.

4341 For example:

```
4342 order_start    forward;backward
```

4343 7.3.2.4 Collation Order

4344 The **order_start** keyword shall be followed by collating identifier entries. The syntax for the
4345 collating element entries is as follows:

```
4346 "%s %s;%s;...;%s\n", <collating-identifier>, <weight>, <weight>, ...
```

4347 Each *collating-identifier* shall consist of either a character (in any of the forms defined in [Section](#)
4348 [7.3](#), on page 122), a *collating-element*, a *collating-symbol*, an ellipsis, or the special symbol
4349 **UNDEFINED**. The order in which collating elements are specified determines the character
4350 order sequence, such that each collating element shall compare less than the elements following

4351 it.

4352 A *<collating-element>* shall be used to specify multi-character collating elements, and indicates
 4353 that the character sequence specified via the *<collating-element>* is to be collated as a unit and in
 4354 the relative order specified by its place.

4355 A *<collating-symbol>* can be used to define a position in the relative order for use in weights. No
 4356 weights shall be specified with a *<collating-symbol>*.

4357 The ellipsis symbol specifies that a sequence of characters shall collate according to their
 4358 encoded character values. It shall be interpreted as indicating that all characters with a coded
 4359 character set value higher than the value of the character in the preceding line, and lower than
 4360 the coded character set value for the character in the following line, in the current coded
 4361 character set, shall be placed in the character collation order between the previous and the
 4362 following character in ascending order according to their coded character set values. An initial
 4363 ellipsis shall be interpreted as if the preceding line specified the NUL character, and a trailing
 4364 ellipsis as if the following line specified the highest coded character set value in the current
 4365 coded character set. An ellipsis shall be treated as invalid if the preceding or following lines do
 4366 not specify characters in the current coded character set. The use of the ellipsis symbol ties the
 4367 definition to a specific coded character set and may preclude the definition from being portable
 4368 between implementations.

4369 The symbol **UNDEFINED** shall be interpreted as including all coded character set values not
 4370 specified explicitly or via the ellipsis symbol. Such characters shall be inserted in the character
 4371 collation order at the point indicated by the symbol, and in ascending order according to their
 4372 coded character set values. If no **UNDEFINED** symbol is specified, and the current coded
 4373 character set contains characters not specified in this section, the utility shall issue a warning
 4374 message and place such characters at the end of the character collation order.

4375 The optional operands for each collation-element shall be used to define the primary, secondary,
 4376 or subsequent weights for the collating element. The first operand specifies the relative primary
 4377 weight, the second the relative secondary weight, and so on. Two or more collation-elements can
 4378 be assigned the same weight; they belong to the same "equivalence class" if they have the same
 4379 primary weight. Collation shall behave as if, for each weight level, elements subject to **IGNORE**
 4380 are removed, unless the **position** collation directive is specified for the corresponding level with
 4381 the **order_start** keyword. Then each successive pair of elements shall be compared according to
 4382 the relative weights for the elements. If the two strings compare equal, the process shall be
 4383 repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

4384 Weights shall be expressed as characters (in any of the forms specified in [Section 7.3](#), on page
 4385 122), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol **IGNORE**. A
 4386 single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative position
 4387 in the character collating sequence of the character or symbol, rather than the character or
 4388 characters themselves. Thus, rather than assigning absolute values to weights, a particular
 4389 weight is expressed using the relative order value assigned to a collating element based on its
 4390 order in the character collation sequence.

4391 One-to-many mapping is indicated by specifying two or more concatenated characters or
 4392 symbolic names. For example, if the *<eszet>* is given the string "*<s><s>*" as a weight,
 4393 comparisons are performed as if all occurrences of the *<eszet>* are replaced by "*<s><s>*"
 4394 (assuming that "*<s>*" has the collating weight "*<s>*"). If it is necessary to define *<eszet>* and
 4395 "*<s><s>*" as an equivalence class, then a collating element must be defined for the string "ss".

4396 All characters specified via an ellipsis shall by default be assigned unique weights, equal to the
 4397 relative order of characters. Characters specified via an explicit or implicit **UNDEFINED** special
 4398 symbol shall by default be assigned the same primary weight (that is, they belong to the same
 4399 equivalence class). An ellipsis symbol as a weight shall be interpreted to mean that each

4400 character in the sequence shall have unique weights, equal to the relative order of their character
 4401 in the character collation sequence. The use of the ellipsis as a weight shall be treated as an error
 4402 if the collating element is neither an ellipsis nor the special symbol **UNDEFINED**.

4403 The special keyword **IGNORE** as a weight shall indicate that when strings are compared using
 4404 the weights at the level where **IGNORE** is specified, the collating element shall be ignored; that
 4405 is, as if the string did not contain the collating element. In regular expressions and pattern
 4406 matching, all characters that are subject to **IGNORE** in their primary weight form an
 4407 equivalence class.

4408 An empty operand shall be interpreted as the collating element itself.

4409 For example, the order statement:

```
4410 <a>    <a>;<a>
```

4411 is equal to:

```
4412 <a>
```

4413 An ellipsis can be used as an operand if the collating element was an ellipsis, and shall be
 4414 interpreted as the value of each character defined by the ellipsis.

4415 The collation order as defined in this section affects the interpretation of bracket expressions in
 4416 regular expressions (see [Section 9.3.5](#), on page 170).

4417 For example:

```
4418 order_start  forward;backward
4419 UNDEFINED   IGNORE;IGNORE
4420 <LOW>
4421 <space>     <LOW>;<space>
4422 ...         <LOW>;...
4423 <a>         <a>;<a>
4424 <a-acute>   <a>;<a-acute>
4425 <a-grave>   <a>;<a-grave>
4426 <A>        <a>;<A>
4427 <A-acute>   <a>;<A-acute>
4428 <A-grave>   <a>;<A-grave>
4429 <ch>       <ch>;<ch>
4430 <Ch>       <ch>;<Ch>
4431 <s>        <s>;<s>
4432 <eszet>    "<s><s>";"<eszet><eszet>"
4433 order_end
```

4434 This example is interpreted as follows:

- 4435 1. The **UNDEFINED** means that all characters not specified in this definition (explicitly or
 4436 via the ellipsis) shall be ignored for collation purposes.
- 4437 2. All characters between <space> and 'a' shall have the same primary equivalence class
 4438 and individual secondary weights based on their ordinal encoded values.
- 4439 3. All characters based on the uppercase or lowercase character 'a' belong to the same
 4440 primary equivalence class.
- 4441 4. The multi-character collating element <ch> is represented by the collating symbol <ch>
 4442 and belongs to the same primary equivalence class as the multi-character collating
 4443 element <Ch>.

4444 7.3.2.5 *The order_end Keyword*4445 The collating order entries shall be terminated with an **order_end** keyword.4446 7.3.2.6 *LC_COLLATE Category in the POSIX Locale*4447 The collation sequence definition of the POSIX locale follows; the code listing depicts the
4448 *localedef* input.

```

4449 LC_COLLATE
4450 # This is the POSIX locale definition for the LC_COLLATE category.
4451 # The order is the same as in the ASCII codeset.
4452 order_start forward
4453 <NUL>
4454 <SOH>
4455 <STX>
4456 <ETX>
4457 <EOT>
4458 <ENQ>
4459 <ACK>
4460 <alert>
4461 <backspace>
4462 <tab>
4463 <newline>
4464 <vertical-tab>
4465 <form-feed>
4466 <carriage-return>
4467 <SO>
4468 <SI>
4469 <DLE>
4470 <DC1>
4471 <DC2>
4472 <DC3>
4473 <DC4>
4474 <NAK>
4475 <SYN>
4476 <ETB>
4477 <CAN>
4478 <EM>
4479 <SUB>
4480 <ESC>
4481 <IS4>
4482 <IS3>
4483 <IS2>
4484 <IS1>
4485 <space>
4486 <exclamation-mark>
4487 <quotation-mark>
4488 <number-sign>
4489 <dollar-sign>
4490 <percent-sign>
4491 <ampersand>
4492 <apostrophe>
4493 <left-parenthesis>

```

4494 <right-parenthesis>
 4495 <asterisk>
 4496 <plus-sign>
 4497 <comma>
 4498 <hyphen>
 4499 <period>
 4500 <slash>
 4501 <zero>
 4502 <one>
 4503 <two>
 4504 <three>
 4505 <four>
 4506 <five>
 4507 <six>
 4508 <seven>
 4509 <eight>
 4510 <nine>
 4511 <colon>
 4512 <semicolon>
 4513 <less-than-sign>
 4514 <equals-sign>
 4515 <greater-than-sign>
 4516 <question-mark>
 4517 <commercial-at>
 4518 <A>
 4519
 4520 <C>
 4521 <D>
 4522 <E>
 4523 <F>
 4524 <G>
 4525 <H>
 4526 <I>
 4527 <J>
 4528 <K>
 4529 <L>
 4530 <M>
 4531 <N>
 4532 <O>
 4533 <P>
 4534 <Q>
 4535 <R>
 4536 <S>
 4537 <T>
 4538 <U>
 4539 <V>
 4540 <W>
 4541 <X>
 4542 <Y>
 4543 <Z>
 4544 <left-square-bracket>
 4545 <backslash>
 4546 <right-square-bracket>

```

4547     <circumflex>
4548     <underscore>
4549     <grave-accent>
4550     <a>
4551     <b>
4552     <c>
4553     <d>
4554     <e>
4555     <f>
4556     <g>
4557     <h>
4558     <i>
4559     <j>
4560     <k>
4561     <l>
4562     <m>
4563     <n>
4564     <o>
4565     <p>
4566     <q>
4567     <r>
4568     <s>
4569     <t>
4570     <u>
4571     <v>
4572     <w>
4573     <x>
4574     <y>
4575     <z>
4576     <left-curly-bracket>
4577     <vertical-line>
4578     <right-curly-bracket>
4579     <tilde>
4580     <DEL>
4581     order_end
4582     #
4583     END LC_COLLATE

```

7.3.3 LC_MONETARY

The *LC_MONETARY* category shall define the rules and symbols that are used to format monetary numeric information.

This information is available through the *localeconv()* function and is used by the *strfmon()* function.

Some of the information is also available in an alternative form via the *nl_langinfo()* function (see CRNCYSTR in [<langinfo.h>](#)).

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the empty string (" ") for unspecified or size zero string items.

4596 In a locale definition file, the operands are strings, formatted as indicated by the grammar in
 4597 [Section 7.4](#) (on page 151). For some keywords, the strings can contain only integers. Keywords
 4598 that are not provided, string values set to the empty string (" "), or integer keywords set to -1,
 4599 are used to indicate that the value is not available in the locale. The following keywords shall be
 4600 recognized:

4601	copy	Specify the name of an existing locale which shall be used as the 4602 definition of this category. If this keyword is specified, no other keyword 4603 shall be specified.
4604		Note: This is a <i>localedef</i> utility keyword, unavailable through <i>localeconv()</i> .
4605	int_curr_symbol	The international currency symbol. The operand shall be a four-character 4606 string, with the first three characters containing the alphabetic 4607 international currency symbol. The international currency symbol should 4608 be chosen in accordance with those specified in the ISO 4217 standard. 4609 The fourth character shall be the character used to separate the 4610 international currency symbol from the monetary quantity.
4611	currency_symbol	The string that shall be used as the local currency symbol.
4612	mon_decimal_point	The operand is a string containing the symbol that shall be used as the 4613 decimal delimiter (radix character) in monetary formatted quantities.
4614	mon_thousands_sep	The operand is a string containing the symbol that shall be used as a 4615 separator for groups of digits to the left of the decimal delimiter in 4616 formatted monetary quantities.
4617	mon_grouping	Define the size of each group of digits in formatted monetary quantities. 4618 The operand is a sequence of integers separated by semicolons. Each 4619 integer specifies the number of digits in each group, with the initial 4620 integer defining the size of the group immediately preceding the decimal 4621 delimiter, and the following integers defining the preceding groups. If the 4622 last integer is not -1, then the size of the previous group (if any) shall be 4623 repeatedly used for the remainder of the digits. If the last integer is -1, 4624 then no further grouping shall be performed.
4625	positive_sign	A string that shall be used to indicate a non-negative-valued formatted 4626 monetary quantity.
4627	negative_sign	A string that shall be used to indicate a negative-valued formatted 4628 monetary quantity.
4629	int_frac_digits	An integer representing the number of fractional digits (those to the right 4630 of the decimal delimiter) to be written in a formatted monetary quantity 4631 using int_curr_symbol .
4632	frac_digits	An integer representing the number of fractional digits (those to the right 4633 of the decimal delimiter) to be written in a formatted monetary quantity 4634 using currency_symbol .
4635	p_cs_precedes	An integer set to 1 if the currency_symbol precedes the value for a 4636 monetary quantity with a non-negative value, and set to 0 if the symbol 4637 succeeds the value.
4638	p_sep_by_space	Set to a value indicating the separation of the currency_symbol , the sign 4639 string, and the value for a non-negative formatted monetary quantity.
4640		The values of p_sep_by_space , n_sep_by_space , int_p_sep_by_space , 4641 and int_n_sep_by_space are interpreted according to the following:

4642		0	No space separates the currency symbol and value.
4643		1	If the currency symbol and sign string are adjacent, a space separates them from the value; otherwise, a space separates the currency symbol from the value.
4644			
4645			
4646		2	If the currency symbol and sign string are adjacent, a space separates them; otherwise, a space separates the sign string from the value.
4647			
4648	n_cs_precedes		An integer set to 1 if the currency_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4649			
4650			
4651	n_sep_by_space		Set to a value indicating the separation of the currency_symbol , the sign string, and the value for a negative formatted monetary quantity.
4652			
4653	p_sign_posn		An integer set to a value indicating the positioning of the positive_sign for a monetary quantity with a non-negative value. The following integer values shall be recognized for int_n_sign_posn , int_p_sign_posn , n_sign_posn , and p_sign_posn :
4654			
4655			
4656			
4657		0	Parentheses enclose the quantity and the currency_symbol .
4658		1	The sign string precedes the quantity and the currency_symbol .
4659		2	The sign string succeeds the quantity and the currency_symbol .
4660		3	The sign string precedes the currency_symbol .
4661		4	The sign string succeeds the currency_symbol .
4662	n_sign_posn		An integer set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity.
4663			
4664	int_p_cs_precedes		An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a non-negative value, and set to 0 if the symbol succeeds the value.
4665			
4666			
4667	int_n_cs_precedes		An integer set to 1 if the int_curr_symbol precedes the value for a monetary quantity with a negative value, and set to 0 if the symbol succeeds the value.
4668			
4669			
4670	int_p_sep_by_space		Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a non-negative internationally formatted monetary quantity.
4671			
4672			
4673	int_n_sep_by_space		Set to a value indicating the separation of the int_curr_symbol , the sign string, and the value for a negative internationally formatted monetary quantity.
4674			
4675			
4676	int_p_sign_posn		An integer set to a value indicating the positioning of the positive_sign for a positive monetary quantity formatted with the international format.
4677			
4678	int_n_sign_posn		An integer set to a value indicating the positioning of the negative_sign for a negative monetary quantity formatted with the international format.
4679			

4680 7.3.3.1 *LC_MONETARY Category in the POSIX Locale*

4681 The monetary formatting definitions for the POSIX locale follow; the code listing depicting the
 4682 *localedef* input, the table representing the same information with the addition of *localeconv()* and
 4683 *nl_langinfo()* formats. All values are unspecified in the POSIX locale.

```

4684 LC_MONETARY
4685 # This is the POSIX locale definition for
4686 # the LC_MONETARY category.
4687 #
4688 int_curr_symbol      ""
4689 currency_symbol     ""
4690 mon_decimal_point    ""
4691 mon_thousands_sep   ""
4692 mon_grouping         -1
4693 positive_sign        ""
4694 negative_sign        ""
4695 int_frac_digits      -1
4696 frac_digits          -1
4697 p_cs_precedes        -1
4698 p_sep_by_space       -1
4699 n_cs_precedes        -1
4700 n_sep_by_space       -1
4701 p_sign_posn          -1
4702 n_sign_posn          -1
4703 int_p_cs_precedes   -1
4704 int_p_sep_by_space  -1
4705 int_n_cs_precedes   -1
4706 int_n_sep_by_space  -1
4707 int_p_sign_posn     -1
4708 int_n_sign_posn     -1
4709 #
4710 END LC_MONETARY

```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
int_curr_symbol	—	N/A	""	""
currency_symbol	CRNCYSTR	N/A	""	""
mon_decimal_point	—	N/A	""	""
mon_thousands_sep	—	N/A	""	""
mon_grouping	—	N/A	""	-1
positive_sign	—	N/A	""	""
negative_sign	—	N/A	""	""
int_frac_digits	—	N/A	{CHAR_MAX}	-1
frac_digits	—	N/A	{CHAR_MAX}	-1
p_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
p_sep_by_space	—	N/A	{CHAR_MAX}	-1
n_cs_precedes	CRNCYSTR	N/A	{CHAR_MAX}	-1
n_sep_by_space	—	N/A	{CHAR_MAX}	-1
p_sign_posn	—	N/A	{CHAR_MAX}	-1
n_sign_posn	—	N/A	{CHAR_MAX}	-1
int_p_cs_precedes	—	N/A	{CHAR_MAX}	-1
int_p_sep_by_space	—	N/A	{CHAR_MAX}	-1
int_n_cs_precedes	—	N/A	{CHAR_MAX}	-1
int_n_sep_by_space	—	N/A	{CHAR_MAX}	-1
int_p_sign_posn	—	N/A	{CHAR_MAX}	-1
int_n_sign_posn	—	N/A	{CHAR_MAX}	-1

The entry N/A indicates that the value is not available in the POSIX locale.

7.3.4 LC_NUMERIC

The *LC_NUMERIC* category shall define the rules and symbols that are used to format non-monetary numeric information. This information is available through the *localeconv()* function.

Some of the information is also available in an alternative form via the *nl_langinfo()* function.

The following items are defined in this category of the locale. The item names are the keywords recognized by the *localedef* utility when defining a locale. They are also similar to the member names of the *lconv* structure defined in [<locale.h>](#); see [<locale.h>](#) for the exact symbols in the header. The *localeconv()* function returns {CHAR_MAX} for unspecified integer items and the empty string (" ") for unspecified or size zero string items.

In a locale definition file, the operands are strings, formatted as indicated by the grammar in [Section 7.4](#) (on page 151). For some keywords, the strings can only contain integers. Keywords that are not provided, string values set to the empty string (" "), or integer keywords set to -1, shall be used to indicate that the value is not available in the locale. The following keywords shall be recognized:

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

Note: This is a *localedef* utility keyword, unavailable through *localeconv()*.

decimal_point The operand is a string containing the symbol that shall be used as the decimal delimiter (radix character) in numeric, non-monetary formatted quantities. This keyword cannot be omitted and cannot be set to the empty string. In contexts where standards limit the **decimal_point** to a single byte, the result of specifying a multi-byte operand shall be unspecified.

4757 **thousands_sep** The operand is a string containing the symbol that shall be used as a separator
4758 for groups of digits to the left of the decimal delimiter in numeric, non-
4759 monetary formatted monetary quantities. In contexts where standards limit
4760 the **thousands_sep** to a single byte, the result of specifying a multi-byte
4761 operand shall be unspecified.

4762 **grouping** Define the size of each group of digits in formatted non-monetary quantities.
4763 The operand is a sequence of integers separated by semicolons. Each integer
4764 specifies the number of digits in each group, with the initial integer defining
4765 the size of the group immediately preceding the decimal delimiter, and the
4766 following integers defining the preceding groups. If the last integer is not -1 ,
4767 then the size of the previous group (if any) shall be repeatedly used for the
4768 remainder of the digits. If the last integer is -1 , then no further grouping shall
4769 be performed.

4770 7.3.4.1 *LC_NUMERIC Category in the POSIX Locale*

4771 The non-monetary numeric formatting definitions for the POSIX locale follow; the code listing
4772 depicting the *localedef* input, the table representing the same information with the addition of
4773 *localeconv()* values, and *nl_langinfo()* constants.

```
4774 LC_NUMERIC
4775 # This is the POSIX locale definition for
4776 # the LC_NUMERIC category.
4777 #
4778 decimal_point    "<period>"
4779 thousands_sep    ""
4780 grouping         -1
4781 #
4782 END LC_NUMERIC
```

Item	langinfo Constant	POSIX Locale Value	localeconv() Value	localedef Value
decimal_point	RADIXCHAR	"."	"."	.
thousands_sep	THOUSEP	N/A	""	""
grouping	—	N/A	""	-1

4788 The entry N/A indicates that the value is not available in the POSIX locale.

4789 7.3.5 LC_TIME

4790 The *LC_TIME* category shall define the interpretation of the conversion specifications supported
4791 by the *date* utility and shall affect the behavior of the *strftime()*, *wcsftime()*, *strptime()*, and
4792 *nl_langinfo()* functions. Since the interfaces for C-language access and locale definition differ
4793 significantly, they are described separately.

4794 7.3.5.1 *LC_TIME Locale Definition*

4795 In a locale definition, the following mandatory keywords shall be recognized:

4796 **copy** Specify the name of an existing locale which shall be used as the definition of
4797 this category. If this keyword is specified, no other keyword shall be specified.

4798 **abday** Define the abbreviated weekday names, corresponding to the $\%a$ conversion
4799 specification (conversion specification in the *strftime()*, *wcsftime()*, and
4800 *strptime()* functions). The operand shall consist of seven semicolon-separated

4801		strings, each surrounded by double-quotes. The first string shall be the
4802		abbreviated name of the day corresponding to Sunday, the second the
4803		abbreviated name of the day corresponding to Monday, and so on.
4804	day	Define the full weekday names, corresponding to the %A conversion
4805		specification. The operand shall consist of seven semicolon-separated strings,
4806		each surrounded by double-quotes. The first string is the full name of the day
4807		corresponding to Sunday, the second the full name of the day corresponding
4808		to Monday, and so on.
4809	abmon	Define the abbreviated month names, corresponding to the %b conversion
4810		specification. The operand shall consist of twelve semicolon-separated strings,
4811		each surrounded by double-quotes. The first string shall be the abbreviated
4812		name of the first month of the year (January), the second the abbreviated
4813		name of the second month, and so on.
4814	mon	Define the full month names, corresponding to the %B conversion
4815		specification. The operand shall consist of twelve semicolon-separated strings,
4816		each surrounded by double-quotes. The first string shall be the full name of
4817		the first month of the year (January), the second the full name of the second
4818		month, and so on.
4819	d_t_fmt	Define the appropriate date and time representation, corresponding to the %c
4820		conversion specification. The operand shall consist of a string containing any
4821		combination of characters and conversion specifications. In addition, the
4822		string can contain escape sequences defined in the table in Table 5-1 (on page
4823		108) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v').
4824	d_fmt	Define the appropriate date representation, corresponding to the %x
4825		conversion specification. The operand shall consist of a string containing any
4826		combination of characters and conversion specifications. In addition, the
4827		string can contain escape sequences defined in Table 5-1 (on page 108).
4828	t_fmt	Define the appropriate time representation, corresponding to the %X
4829		conversion specification. The operand shall consist of a string containing any
4830		combination of characters and conversion specifications. In addition, the
4831		string can contain escape sequences defined in Table 5-1 (on page 108).
4832	am_pm	Define the appropriate representation of the <i>ante-meridiem</i> and <i>post-meridiem</i>
4833		strings, corresponding to the %p conversion specification. The operand shall
4834		consist of two strings, separated by a semicolon, each surrounded by double-
4835		quotes. The first string shall represent the <i>ante-meridiem</i> designation, the last
4836		string the <i>post-meridiem</i> designation.
4837	t_fmt_ampm	Define the appropriate time representation in the 12-hour clock format with
4838		am_pm , corresponding to the %r conversion specification. The operand shall
4839		consist of a string and can contain any combination of characters and
4840		conversion specifications. If the string is empty, the 12-hour format is not
4841		supported in the locale.
4842	era	Define how years are counted and displayed for each era in a locale. The
4843		operand shall consist of semicolon-separated strings. Each string shall be an
4844		era description segment with the format:
4845		<i>direction:offset:start_date:end_date:era_name:era_format</i>
4846		according to the definitions below. There can be as many era description
4847		segments as are necessary to describe the different eras.

4848	Note:	The start of an era might not be the earliest point in the era—it may be the
4849		latest. For example, the Christian era BC starts on the day before January 1,
4850		AD 1, and increases with earlier time.
4851	<i>direction</i>	Either a '+' or a '-' character. The '+' character shall indicate
4852		that years closer to the <i>start_date</i> have lower numbers than those
4853		closer to the <i>end_date</i> . The '-' character shall indicate that years
4854		closer to the <i>start_date</i> have higher numbers than those closer to
4855		the <i>end_date</i> .
4856	<i>offset</i>	The number of the year closest to the <i>start_date</i> in the era,
4857		corresponding to the %Ey conversion specification.
4858	<i>start_date</i>	A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the
4859		year, month, and day numbers respectively of the start of the era.
4860		Years prior to AD 1 shall be represented as negative numbers.
4861	<i>end_date</i>	The ending date of the era, in the same format as the <i>start_date</i> ,
4862		or one of the two special values "-*" or "+*". The value "-*" shall
4863		indicate that the ending date is the beginning of time. The
4864		value "+*" shall indicate that the ending date is the end of time.
4865	<i>era_name</i>	A string representing the name of the era, corresponding to the
4866		%EC conversion specification.
4867	<i>era_format</i>	A string for formatting the year in the era, corresponding to the
4868		%EY conversion specification.
4869	era_d_fmt	Define the format of the date in alternative era notation, corresponding to the
4870		%Ex conversion specification.
4871	era_t_fmt	Define the locale's appropriate alternative time format, corresponding to the
4872		%EX conversion specification.
4873	era_d_t_fmt	Define the locale's appropriate alternative date and time format,
4874		corresponding to the %Ec conversion specification.
4875	alt_digits	Define alternative symbols for digits, corresponding to the %O modified
4876		conversion specification. The operand shall consist of semicolon-separated
4877		strings, each surrounded by double-quotes. The first string shall be the
4878		alternative symbol corresponding with zero, the second string the symbol
4879		corresponding with one, and so on. Up to 100 alternative symbol strings can
4880		be specified. The %O modifier shall indicate that the string corresponding to
4881		the value specified via the conversion specification shall be used instead of the
4882		value.

4883 7.3.5.2 LC_TIME C-Language Access

4884 The following constants used to identify items of *langinfo* data can be used as arguments to the
 4885 *nl_langinfo()* function to access information in the *LC_TIME* category. These constants are
 4886 defined in the **<langinfo.h>** header.

4887	ABDAY_x	The abbreviated weekday names (for example, Sun), where <i>x</i> is a number
4888		from 1 to 7.
4889	DAY_x	The full weekday names (for example, Sunday), where <i>x</i> is a number from 1 to
4890		7.

4891	ABMON_x	The abbreviated month names (for example, Jan), where <i>x</i> is a number from 1 to 12.
4892		
4893	MON_x	The full month names (for example, January), where <i>x</i> is a number from 1 to 12.
4894		
4895	D_T_FMT	The appropriate date and time representation.
4896	D_FMT	The appropriate date representation.
4897	T_FMT	The appropriate time representation.
4898	AM_STR	The appropriate ante-meridiem affix.
4899	PM_STR	The appropriate post-meridiem affix.
4900	T_FMT_AMPM	The appropriate time representation in the 12-hour clock format with AM_STR and PM_STR.
4901		
4902	ERA	The era description segments, which describe how years are counted and displayed for each era in a locale. Each era description segment shall have the format:
4903		
4904		
4905		<i>direction:offset:start_date:end_date:era_name:era_format</i>
4906		according to the definitions below. There can be as many era description segments as are necessary to describe the different eras. Era description segments are separated by semicolons.
4907		
4908		
4909		<i>direction</i> Either a '+' or a '-' character. The '+' character shall indicate that years closer to the <i>start_date</i> have lower numbers than those closer to the <i>end_date</i> . The '-' character shall indicate that years closer to the <i>start_date</i> have higher numbers than those closer to the <i>end_date</i> .
4910		
4911		
4912		
4913		
4914		<i>offset</i> The number of the year closest to the <i>start_date</i> in the era.
4915		<i>start_date</i> A date in the form <i>yyyy/mm/dd</i> , where <i>yyyy</i> , <i>mm</i> , and <i>dd</i> are the year, month, and day numbers respectively of the start of the era. Years prior to AD 1 shall be represented as negative numbers.
4916		
4917		
4918		<i>end_date</i> The ending date of the era, in the same format as the <i>start_date</i> , or one of the two special values "-*" or "+*". The value "-*" shall indicate that the ending date is the beginning of time. The value "+*" shall indicate that the ending date is the end of time.
4919		
4920		
4921		
4922		<i>era_name</i> The era, corresponding to the %EC conversion specification.
4923		<i>era_format</i> The format of the year in the era, corresponding to the %EY conversion specification.
4924		
4925	ERA_D_FMT	The era date format.
4926	ERA_T_FMT	The locale's appropriate alternative time format, corresponding to the %EX conversion specification.
4927		
4928	ERA_D_T_FMT	The locale's appropriate alternative date and time format, corresponding to the %Ec conversion specification.
4929		
4930	ALT_DIGITS	The alternative symbols for digits, corresponding to the %O conversion specification modifier. The value consists of semicolon-separated symbols. The first is the alternative symbol corresponding to zero, the second is the symbol corresponding to one, and so on. Up to 100 alternative symbols may be
4931		
4932		
4933		

4934 specified.

4935 7.3.5.3 LC_TIME Category in the POSIX Locale

4936 The *LC_TIME* category definition of the POSIX locale follows; the code listing depicts the
4937 *localedef* input; the table represents the same information with the addition of *localedef* keywords,
4938 conversion specifiers used by the *date* utility and the *strftime()*, *wcsftime()*, and *strptime()*
4939 functions, and *nl_langinfo()* constants.

```

4940 LC_TIME
4941 # This is the POSIX locale definition for
4942 # the LC_TIME category.
4943 #
4944 # Abbreviated weekday names (%a)
4945 abday      "<S><u><n>" ; "<M><o><n>" ; "<T><u><e>" ; "<W><e><d>" ; \
4946            "<T><h><u>" ; "<F><r><i>" ; "<S><a><t>"
4947 #
4948 # Full weekday names (%A)
4949 day        "<S><u><n><d><a><y>" ; "<M><o><n><d><a><y>" ; \
4950            "<T><u><e><s><d><a><y>" ; "<W><e><d><n><e><s><d><a><y>" ; \
4951            "<T><h><u><r><s><d><a><y>" ; "<F><r><i><d><a><y>" ; \
4952            "<S><a><t><u><r><d><a><y>"
4953 #
4954 # Abbreviated month names (%b)
4955 abmon      "<J><a><n>" ; "<F><e><b>" ; "<M><a><r>" ; \
4956            "<A><p><r>" ; "<M><a><y>" ; "<J><u><n>" ; \
4957            "<J><u><l>" ; "<A><u><g>" ; "<S><e><p>" ; \
4958            "<O><c><t>" ; "<N><o><v>" ; "<D><e><c>"
4959 #
4960 # Full month names (%B)
4961 mon        "<J><a><n><u><a><r><y>" ; "<F><e><b><r><u><a><r><y>" ; \
4962            "<M><a><r><c><h>" ; "<A><p><r><i><l>" ; \
4963            "<M><a><y>" ; "<J><u><n><e>" ; \
4964            "<J><u><l><y>" ; "<A><u><g><u><s><t>" ; \
4965            "<S><e><p><t><e><m><b><e><r>" ; "<O><c><t><o><b><e><r>" ; \
4966            "<N><o><v><e><m><b><e><r>" ; "<D><e><c><e><m><b><e><r>"
4967 #
4968 # Equivalent of AM/PM (%p)      "AM" ; "PM"
4969 am_pm      "<A><M>" ; "<P><M>"
4970 #
4971 # Appropriate date and time representation (%c)
4972 #      "%a %b %e %H:%M:%S %Y"
4973 d_t_fmt    "<percent-sign><a><space><percent-sign><b>\
4974            <space><percent-sign><e><space><percent-sign><H>\
4975            <colon><percent-sign><M><colon><percent-sign><S>\
4976            <space><percent-sign><Y>"
4977 #
4978 # Appropriate date representation (%x)      "%m/%d/%y"
4979 d_fmt      "<percent-sign><m><slash><percent-sign><d>\
4980            <slash><percent-sign><y>"
4981 #
4982 # Appropriate time representation (%X)      "%H:%M:%S"
4983 t_fmt      "<percent-sign><H><colon><percent-sign><M>\
4984            <colon><percent-sign><S>"

```

```

4985 #
4986 # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
4987 t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
4988 <percent-sign><S><space><percent_sign><p>"
4989 #
4990 END LC_TIME

```

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
4991 d_t_fmt	D_T_FMT	%c	"%a %b %e %H:%M:%S %Y"
4992 d_fmt	D_FMT	%x	"%m/%d/%y"
4993 t_fmt	T_FMT	%X	"%H:%M:%S"
4994 am_pm	AM_STR	%p	"AM"
4995 am_pm	PM_STR	%p	"PM"
4996 t_fmt_ampm	T_FMT_AMPMM	%r	"%I:%M:%S %p"
4997 day	DAY_1	%A	"Sunday"
4998 day	DAY_2	%A	"Monday"
4999 day	DAY_3	%A	"Tuesday"
5000 day	DAY_4	%A	"Wednesday"
5001 day	DAY_5	%A	"Thursday"
5002 day	DAY_6	%A	"Friday"
5003 day	DAY_7	%A	"Saturday"
5004 abday	ABDAY_1	%a	"Sun"
5005 abday	ABDAY_2	%a	"Mon"
5006 abday	ABDAY_3	%a	"Tue"
5007 abday	ABDAY_4	%a	"Wed"
5008 abday	ABDAY_5	%a	"Thu"
5009 abday	ABDAY_6	%a	"Fri"
5010 abday	ABDAY_7	%a	"Sat"
5011 mon	MON_1	%B	"January"
5012 mon	MON_2	%B	"February"
5013 mon	MON_3	%B	"March"
5014 mon	MON_4	%B	"April"
5015 mon	MON_5	%B	"May"
5016 mon	MON_6	%B	"June"
5017 mon	MON_7	%B	"July"
5018 mon	MON_8	%B	"August"
5019 mon	MON_9	%B	"September"
5020 mon	MON_10	%B	"October"
5021 mon	MON_11	%B	"November"
5022 mon	MON_12	%B	"December"
5023 abmon	ABMON_1	%b	"Jan"
5024 abmon	ABMON_2	%b	"Feb"
5025 abmon	ABMON_3	%b	"Mar"
5026 abmon	ABMON_4	%b	"Apr"
5027 abmon	ABMON_5	%b	"May"
5028 abmon	ABMON_6	%b	"Jun"
5029 abmon	ABMON_7	%b	"Jul"
5030 abmon	ABMON_8	%b	"Aug"
5031 abmon	ABMON_9	%b	"Sep"
5032 abmon	ABMON_10	%b	"Oct"
5033 abmon	ABMON_11	%b	"Nov"

5036
5037
5038
5039
5040
5041
5042
5043

localedef Keyword	langinfo Constant	Conversion Specification	POSIX Locale Value
abmon	ABMON_12	%b	"Dec "
era	ERA	%EC, %Ey, %EY	N/A
era_d_fmt	ERA_D_FMT	%Ex	N/A
era_t_fmt	ERA_T_FMT	%EX	N/A
era_d_t_fmt	ERA_D_T_FMT	%Ec	N/A
alt_digits	ALT_DIGITS	%O	N/A

5044

The entry N/A indicates the value is not available in the POSIX locale.

5045

7.3.6 LC_MESSAGES

5046
5047
5048

The *LC_MESSAGES* category shall define the format and values used by various utilities for affirmative and negative responses. This information is available through the *nl_langinfo()* function.

5049
5050
5051

The message catalog used by the standard utilities and selected by the *catopen()* function shall be determined by the setting of *NLSPATH*; see [Chapter 8](#) (on page 159). The *LC_MESSAGES* category can be specified as part of an *NLSPATH* substitution field.

5052

The following keywords shall be recognized as part of the locale definition file.

5053
5054

copy Specify the name of an existing locale which shall be used as the definition of this category. If this keyword is specified, no other keyword shall be specified.

5055

Note: This is a *localedef* keyword, unavailable through *nl_langinfo()*.

5056
5057
5058

yesexpr The operand consists of an extended regular expression (see [Section 9.4](#), on page 174) that describes the acceptable affirmative response to a question expecting an affirmative or negative response.

5059
5060
5061

noexpr The operand consists of an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.

5062

7.3.6.1 LC_MESSAGES Category in the POSIX Locale

5063
5064
5065

The format and values for affirmative and negative responses of the POSIX locale follow; the code listing depicting the *localedef* input, the table representing the same information with the addition of *nl_langinfo()* constants.

5066
5067
5068
5069
5070
5071
5072
5073
5074

```
LC_MESSAGES
# This is the POSIX locale definition for
# the LC_MESSAGES category.
#
yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
#
noexpr "<circumflex><left-square-bracket><n><N><right-square-bracket>"
#
END LC_MESSAGES
```

5075
5076
5077

localedef Keyword	langinfo Constant	POSIX Locale Value
yesexpr	YESEXPR	"^[yY]"
noexpr	NOEXPR	"^[nN]"

7.4 Locale Definition Grammar

5078

5079

5080

5081

The grammar and lexical conventions in this section shall together describe the syntax for the locale definition source. The general conventions for this style of grammar are described in XCU [Section 1.3](#) (on page 2235). The grammar shall take precedence over the text in this chapter.

7.4.1 Locale Lexical Conventions

5082

The lexical conventions for the locale definition grammar are described in this section.

5083

5084

5085

The following tokens shall be processed (in addition to those string constants shown in the grammar):

5086

LOC_NAME A string of characters representing the name of a locale.

5087

CHAR Any single character.

5088

NUMBER A decimal number, represented by one or more decimal digits.

5089

5090

5091

COLLSYMBOL A symbolic name, enclosed between angle brackets. The string cannot duplicate any charmap symbol defined in the current charmap (if any), or a **COLLELEMENT** symbol.

5092

5093

COLLELEMENT A symbolic name, enclosed between angle brackets, which cannot duplicate either any charmap symbol or a **COLLSYMBOL** symbol.

5094

5095

5096

5097

CHARCLASS A string of alphanumeric characters from the portable character set, the first of which is not a digit, consisting of at least one and at most {**CHARCLASS_NAME_MAX**} bytes, and optionally surrounded by double-quotes.

5098

5099

CHARSYMBOL A symbolic name, enclosed between angle brackets, from the current charmap (if any).

5100

5101

5102

5103

OCTAL_CHAR One or more octal representations of the encoding of each byte in a single character. The octal representation consists of an escape character (normally a backslash) followed by two or more octal digits.

5104

5105

5106

5107

HEX_CHAR One or more hexadecimal representations of the encoding of each byte in a single character. The hexadecimal representation consists of an escape character followed by the constant *x* and two or more hexadecimal digits.

5108

5109

5110

5111

DECIMAL_CHAR One or more decimal representations of the encoding of each byte in a single character. The decimal representation consists of an escape character followed by a character 'd' and two or more decimal digits.

5112

ELLIPSIS The string " . . . ".

5113

5114

EXTENDED_REG_EXP An extended regular expression as defined in the grammar in [Section 9.5](#) (on page 177).

5115

EOL The line termination character <newline>.

7.4.2 Locale Grammar

5116
5117 This section presents the grammar for the locale definition.

```

5118 %token      LOC_NAME
5119 %token      CHAR
5120 %token      NUMBER
5121 %token      COLLSYMBOL COLLELEMENT
5122 %token      CHARSYMBOL OCTAL_CHAR HEX_CHAR DECIMAL_CHAR
5123 %token      ELLIPSIS
5124 %token      EXTENDED_REG_EXP
5125 %token      EOL

5126 %start      locale_definition

5127 %%

5128 locale_definition : global_statements locale_categories
5129                  | locale_categories
5130                  ;

5131 global_statements : global_statements symbol_redefine
5132                  | symbol_redefine
5133                  ;

5134 symbol_redefine   : 'escape_char' CHAR EOL
5135                  | 'comment_char' CHAR EOL
5136                  ;

5137 locale_categories : locale_categories locale_category
5138                  | locale_category
5139                  ;

5140 locale_category   : lc_ctype | lc_collate | lc_messages
5141                  | lc_monetary | lc_numeric | lc_time
5142                  ;

5143 /* The following grammar rules are common to all categories */

5144 char_list        : char_list char_symbol
5145                  | char_symbol
5146                  ;

5147 char_symbol      : CHAR | CHARSYMBOL
5148                  | OCTAL_CHAR | HEX_CHAR | DECIMAL_CHAR
5149                  ;

5150 elem_list        : elem_list char_symbol
5151                  | elem_list COLLSYMBOL
5152                  | elem_list COLLELEMENT
5153                  | char_symbol
5154                  | COLLSYMBOL
5155                  | COLLELEMENT
5156                  ;

5157 symb_list        : symb_list COLLSYMBOL
5158                  | COLLSYMBOL
5159                  ;

5160 locale_name      : LOC_NAME

```



```

5161         | ' ' LOC_NAME ' '
5162         ;

5163     /* The following is the LC_CTYPE category grammar */
5164     lc_ctype      : ctype_hdr ctype_keywords      ctype_tlr
5165                   | ctype_hdr 'copy' locale_name EOL ctype_tlr
5166                   ;

5167     ctype_hdr     : 'LC_CTYPE' EOL
5168                   ;

5169     ctype_keywords : ctype_keywords ctype_keyword
5170                   | ctype_keyword
5171                   ;

5172     ctype_keyword  : charclass_keyword charclass_list EOL
5173                   | charconv_keyword charconv_list EOL
5174                   | 'charclass' charclass_namelist EOL
5175                   ;

5176     charclass_namelist : charclass_namelist ';' CHARCLASS
5177                       | CHARCLASS
5178                       ;

5179     charclass_keyword : 'upper' | 'lower' | 'alpha' | 'digit'
5180                       | 'punct' | 'xdigit' | 'space' | 'print'
5181                       | 'graph' | 'blank' | 'cntrl' | 'alnum'
5182                       | CHARCLASS
5183                       ;

5184     charclass_list   : charclass_list ';' char_symbol
5185                       | charclass_list ';' ELLIPSIS ';' char_symbol
5186                       | char_symbol
5187                       ;

5188     charconv_keyword : 'toupper'
5189                       | 'tolower'
5190                       ;

5191     charconv_list    : charconv_list ';' charconv_entry
5192                       | charconv_entry
5193                       ;

5194     charconv_entry    : '(' char_symbol ',' char_symbol ')'
5195                       ;

5196     ctype_tlr       : 'END' 'LC_CTYPE' EOL
5197                       ;

5198     /* The following is the LC_COLLATE category grammar */
5199     lc_collate      : collate_hdr collate_keywords      collate_tlr
5200                   | collate_hdr 'copy' locale_name EOL collate_tlr
5201                   ;

5202     collate_hdr     : 'LC_COLLATE' EOL
5203                   ;

5204     collate_keywords :          order_statements
5205                   | opt_statements order_statements

```

```

5206             ;
5207     opt_statements : opt_statements collating_symbols
5208                   | opt_statements collating_elements
5209                   | collating_symbols
5210                   | collating_elements
5211             ;
5212     collating_symbols : 'collating-symbol' COLLSYMBOL EOL
5213             ;
5214     collating_elements : 'collating-element' COLLELEMENT
5215                       | 'from' "" elem_list "" EOL
5216             ;
5217     order_statements : order_start collation_order order_end
5218             ;
5219     order_start : 'order_start' EOL
5220               | 'order_start' order_opts EOL
5221             ;
5222     order_opts : order_opts ';' order_opt
5223               | order_opt
5224             ;
5225     order_opt : order_opt ',' opt_word
5226               | opt_word
5227             ;
5228     opt_word : 'forward' | 'backward' | 'position'
5229             ;
5230     collation_order : collation_order collation_entry
5231                   | collation_entry
5232             ;
5233     collation_entry : COLLSYMBOL EOL
5234                   | collation_element weight_list EOL
5235                   | collation_element EOL
5236             ;
5237     collation_element : char_symbol
5238                      | COLLELEMENT
5239                      | ELLIPSIS
5240                      | 'UNDEFINED'
5241             ;
5242     weight_list : weight_list ';' weight_symbol
5243                | weight_list ';'
5244                | weight_symbol
5245             ;
5246     weight_symbol : /* empty */
5247                  | char_symbol
5248                  | COLLSYMBOL
5249                  | "" elem_list ""
5250                  | "" symb_list ""
5251                  | ELLIPSIS

```

```

5252         | 'IGNORE'
5253         ;
5254     order_end      : 'order_end' EOL
5255         ;
5256     collate_tlr    : 'END' 'LC_COLLATE' EOL
5257         ;
5258     /* The following is the LC_MESSAGES category grammar */
5259     lc_messages     : messages_hdr messages_keywords      messages_tlr
5260         | messages_hdr 'copy' locale_name EOL messages_tlr
5261         ;
5262     messages_hdr    : 'LC_MESSAGES' EOL
5263         ;
5264     messages_keywords : messages_keywords messages_keyword
5265         | messages_keyword
5266         ;
5267     messages_keyword : 'yesexpr' ' "' EXTENDED_REG_EXP '"' EOL
5268         | 'noexpr' ' "' EXTENDED_REG_EXP '"' EOL
5269         ;
5270     messages_tlr    : 'END' 'LC_MESSAGES' EOL
5271         ;
5272     /* The following is the LC_MONETARY category grammar */
5273     lc_monetary     : monetary_hdr monetary_keywords      monetary_tlr
5274         | monetary_hdr 'copy' locale_name EOL monetary_tlr
5275         ;
5276     monetary_hdr    : 'LC_MONETARY' EOL
5277         ;
5278     monetary_keywords : monetary_keywords monetary_keyword
5279         | monetary_keyword
5280         ;
5281     monetary_keyword : mon_keyword_string mon_string EOL
5282         | mon_keyword_char NUMBER EOL
5283         | mon_keyword_char '-1' EOL
5284         | mon_keyword_grouping mon_group_list EOL
5285         ;
5286     mon_keyword_string : 'int_curr_symbol' | 'currency_symbol'
5287         | 'mon_decimal_point' | 'mon_thousands_sep'
5288         | 'positive_sign' | 'negative_sign'
5289         ;
5290     mon_string       : '"' char_list '"'
5291         | '""'
5292         ;
5293     mon_keyword_char : 'int_frac_digits' | 'frac_digits'
5294         | 'p_cs_precedes' | 'p_sep_by_space'
5295         | 'n_cs_precedes' | 'n_sep_by_space'
5296         | 'p_sign_posn' | 'n_sign_posn'

```

```

5297 | 'int_p_cs_precedes' | 'int_p_sep_by_space'
5298 | 'int_n_cs_precedes' | 'int_n_sep_by_space'
5299 | 'int_p_sign_posn' | 'int_n_sign_posn'
5300 ;

5301 mon_keyword_grouping : 'mon_grouping'
5302 ;

5303 mon_group_list      : NUMBER
5304 | mon_group_list ';' NUMBER
5305 ;

5306 monetary_tlr       : 'END' 'LC_MONETARY' EOL
5307 ;

5308 /* The following is the LC_NUMERIC category grammar */
5309 lc_numeric          : numeric_hdr numeric_keywords      numeric_tlr
5310 | numeric_hdr 'copy' locale_name EOL numeric_tlr
5311 ;

5312 numeric_hdr         : 'LC_NUMERIC' EOL
5313 ;

5314 numeric_keywords    : numeric_keywords numeric_keyword
5315 | numeric_keyword
5316 ;

5317 numeric_keyword     : num_keyword_string num_string EOL
5318 | num_keyword_grouping num_group_list EOL
5319 ;

5320 num_keyword_string  : 'decimal_point'
5321 | 'thousands_sep'
5322 ;

5323 num_string          : '"' char_list '"'
5324 | '""'
5325 ;

5326 num_keyword_grouping : 'grouping'
5327 ;

5328 num_group_list      : NUMBER
5329 | num_group_list ';' NUMBER
5330 ;

5331 numeric_tlr         : 'END' 'LC_NUMERIC' EOL
5332 ;

5333 /* The following is the LC_TIME category grammar */
5334 lc_time             : time_hdr time_keywords          time_tlr
5335 | time_hdr 'copy' locale_name EOL time_tlr
5336 ;

5337 time_hdr            : 'LC_TIME' EOL
5338 ;

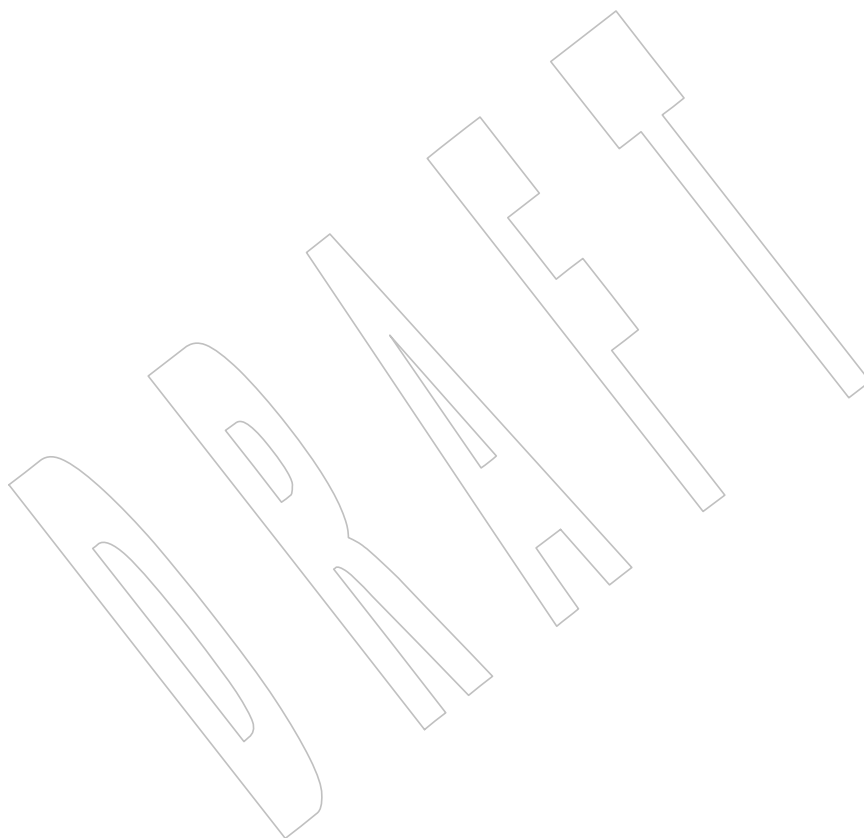
5339 time_keywords       : time_keywords time_keyword
5340 | time_keyword

```

```

5341                                     ;
5342     time_keyword                       : time_keyword_name time_list EOL
5343     | time_keyword_fmt time_string EOL
5344     | time_keyword_opt time_list EOL
5345     ;
5346     time_keyword_name                   : 'abday' | 'day' | 'abmon' | 'mon'
5347     ;
5348     time_keyword_fmt                    : 'd_t_fmt' | 'd_fmt' | 't_fmt'
5349     | 'am_pm' | 't_fmt_ampm'
5350     ;
5351     time_keyword_opt                     : 'era' | 'era_d_fmt' | 'era_t_fmt'
5352     | 'era_d_t_fmt' | 'alt_digits'
5353     ;
5354     time_list                            : time_list ';' time_string
5355     | time_string
5356     ;
5357     time_string                          : '"' char_list '"'
5358     ;
5359     time_tlr                             : 'END' 'LC_TIME' EOL
5360     ;

```



8.1 Environment Variable Definition

Environment variables defined in this chapter affect the operation of multiple utilities, functions, and applications. There are other environment variables that are of interest only to specific utilities. Environment variables that apply to a single utility only are defined as part of the utility description. See the ENVIRONMENT VARIABLES section of the utility descriptions in the Shell and Utilities volume of POSIX.1-200x for information on environment variable usage.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. The array is pointed to by the external variable *environ*, which is defined as:

```
extern char **environ;
```

These strings have the form *name=value*; *names* shall not contain the character '='. For values to be portable across systems conforming to POSIX.1-200x, the value shall be composed of characters from the portable character set (except NUL and as indicated below). There is no meaning associated with the order of strings in the environment. If more than one string in an environment of a process has the same *name*, the consequences are undefined.

Environment variable names used by the utilities in the Shell and Utilities volume of POSIX.1-200x consist solely of uppercase letters, digits, and the '_' (underscore) from the characters defined in Table 6-1 (on page 111) and do not begin with a digit. Other characters may be permitted by an implementation; applications shall tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for applications. Applications can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

Note: Other applications may have difficulty dealing with environment variable names that start with a digit. For this reason, use of such names is not recommended anywhere.

The *values* that the environment variables may be assigned are not restricted except that they are considered to end with a null byte and the total space used to store the environment and the arguments to the process is limited to {ARG_MAX} bytes.

Other *name=value* pairs may be placed in the environment by, for example, calling any of the *setenv()*, *unsetenv()*, or *putenv()* functions, manipulating the *environ* variable, or by using *envp* arguments when creating a process; see *exec* in the System Interfaces volume of POSIX.1-200x.

It is unwise to conflict with certain variables that are frequently exported by widely used command interpreters and applications:

5396	ARFLAGS	IFS	MAILPATH	PS1
5397	CC	LANG	MAILRC	PS2
5398	CDPATH	LC_ALL	MAKEFLAGS	PS3
5399	CFLAGS	LC_COLLATE	MAKESHELL	PS4
5400	CHARSET	LC_CTYPE	MANPATH	PWD
5401	COLUMNS	LC_MESSAGES	MBOX	RANDOM
5402	DATMSK	LC_MONETARY	MORE	SECONDS
5403	DEAD	LC_NUMERIC	MSGVERB	SHELL
5404	EDITOR	LC_TIME	NLSPATH	TERM
5405	ENV	LDFLAGS	NPROC	TERMCAP
5406	EXINIT	LEX	OLDPWD	TERMINFO
5407	FC	LFLAGS	OPTARG	TMPDIR
5408	FCEDIT	LINENO	OPTERR	TZ
5409	FFLAGS	LINES	OPTIND	USER
5410	GET	LISTER	PAGER	VISUAL
5411	GFLAGS	LOGNAME	PATH	YACC
5412	HISTFILE	LPDEST	PPID	YFLAGS
5413	HISTORY	MAIL	PRINTER	
5414	HISTSIZE	MAILCHECK	PROCLANG	
5415	HOME	MAILER	PROJECTDIR	

5416 If the variables in the following two sections are present in the environment during the
 5417 execution of an application or utility, they shall be given the meaning described below. Some are
 5418 placed into the environment by the implementation at the time the user logs in; all can be added
 5419 or changed by the user or any ancestor of the current process. The implementation adds or
 5420 changes environment variables named in POSIX.1-200x only as specified in POSIX.1-200x. If
 5421 they are defined in the application's environment, the utilities in the Shell and Utilities volume
 5422 of POSIX.1-200x and the functions in the System Interfaces volume of POSIX.1-200x assume they
 5423 have the specified meaning. Conforming applications shall not set these environment variables
 5424 to have meanings other than as described. See *getenv()* (on page 973) and XCU Section 2.12 (on
 5425 page 2277) for methods of accessing these variables.

5426 8.2 Internationalization Variables

5427 This section describes environment variables that are relevant to the operation of
 5428 internationalized interfaces described in POSIX.1-200x.

5429 Users may use the following environment variables to announce specific localization
 5430 requirements to applications. Applications can retrieve this information using the *setlocale()*
 5431 function to initialize the correct behavior of the internationalized interfaces. The descriptions of
 5432 the internationalization environment variables describe the resulting behavior only when the
 5433 application locale is initialized in this way. The use of the internationalization variables by
 5434 utilities described in the Shell and Utilities volume of POSIX.1-200x is described in the
 5435 ENVIRONMENT VARIABLES section for those utilities in addition to the global effects
 5436 described in this section.

5437 **LANG** This variable shall determine the locale category for native language, local
 5438 customs, and coded character set in the absence of the *LC_ALL* and other *LC_**
 5439 (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*, *LC_MONETARY*, *LC_NUMERIC*,
 5440 *LC_TIME*) environment variables. This can be used by applications to
 5441 determine the language to use for error messages and instructions, collating
 5442 sequences, date formats, and so on.

5443 5444 5445 5446 5447	<i>LC_ALL</i>	This variable shall determine the values for all locale categories. The value of the <i>LC_ALL</i> environment variable has precedence over any of the other environment variables starting with <i>LC_</i> (<i>LC_COLLATE</i> , <i>LC_CTYPE</i> , <i>LC_MESSAGES</i> , <i>LC_MONETARY</i> , <i>LC_NUMERIC</i> , <i>LC_TIME</i>) and the <i>LANG</i> environment variable.
5448 5449 5450 5451 5452	<i>LC_COLLATE</i>	This variable shall determine the locale category for character collation. It determines collation information for regular expressions and sorting, including equivalence classes and multi-character collating elements, in various utilities and the <i>strcoll()</i> and <i>strxfrm()</i> functions. Additional semantics of this variable, if any, are implementation-defined.
5453 5454 5455 5456 5457 5458 5459	<i>LC_CTYPE</i>	This variable shall determine the locale category for character handling functions, such as <i>tolower()</i> , <i>toupper()</i> , and <i>isalpha()</i> . This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multi-byte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation-defined.
5460 5461 5462 5463 5464 5465 5466 5467	<i>LC_MESSAGES</i>	This variable shall determine the locale category for processing affirmative and negative responses and the language and cultural conventions in which messages should be written. It also affects the behavior of the <i>catopen()</i> function in determining the message catalog. Additional semantics of this variable, if any, are implementation-defined. The language and cultural conventions of diagnostic and informative messages whose format is unspecified by POSIX.1-200x should be affected by the setting of <i>LC_MESSAGES</i> .
5468 5469 5470	<i>LC_MONETARY</i>	This variable shall determine the locale category for monetary-related numeric formatting information. Additional semantics of this variable, if any, are implementation-defined.
5471 5472 5473 5474 5475	<i>LC_NUMERIC</i>	This variable shall determine the locale category for numeric formatting (for example, thousands separator and radix character) information in various utilities as well as the formatted I/O operations in <i>printf()</i> and <i>scanf()</i> and the string conversion functions in <i>strtod()</i> . Additional semantics of this variable, if any, are implementation-defined.
5476 5477 5478	<i>LC_TIME</i>	This variable shall determine the locale category for date and time formatting information. It affects the behavior of the time functions in <i>strptime()</i> . Additional semantics of this variable, if any, are implementation-defined.
5479 5480 5481 5482	<i>NLSPATH</i>	This variable shall contain a sequence of templates that the <i>catopen()</i> function uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more conversion specifications, a filename, and an optional suffix.
5483		For example:
5484		<code>NLSPATH="/system/nlslib/%N.cat"</code>
5485 5486 5487		defines that <i>catopen()</i> should look for all message catalogs in the directory /system/nlslib , where the catalog name should be constructed from the <i>name</i> parameter passed to <i>catopen()</i> (<i>%N</i>), with the suffix .cat .
5488 5489		Conversion specifications consist of a <code>'%'</code> symbol, followed by a single-letter keyword. The following keywords are currently defined:

- 5490 %N The value of the *name* parameter passed to *catopen()*.
- 5491 %L The value of the *LC_MESSAGES* category.
- 5492 %l The *language* element from the *LC_MESSAGES* category.
- 5493 %t The *territory* element from the *LC_MESSAGES* category.
- 5494 %c The *codeset* element from the *LC_MESSAGES* category.
- 5495 %% A single '%' character.

5496 An empty string is substituted if the specified value is not currently defined.
 5497 The separators underscore ('_') and period ('.') are not included in the %t
 5498 and %c conversion specifications.

5499 Templates defined in *NLSPATH* are separated by colons (':'). A leading or
 5500 two adjacent colons "::" is equivalent to specifying %N. For example:

5501 `NLSPATH=":%N.cat:/nlslib/%L/%N.cat"`

5502 indicates to *catopen()* that it should look for the requested message catalog in
 5503 *name*, *name.cat*, and */nlslib/category/name.cat*, where *category* is the value of the
 5504 *LC_MESSAGES* category of the current locale.

5505 Users should not set the *NLSPATH* variable unless they have a specific reason
 5506 to override the default system path. Setting *NLSPATH* to override the default
 5507 system path produces undefined results in the standard utilities and in
 5508 applications with appropriate privileges.

5509 The environment variables *LANG*, *LC_ALL*, *LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
 5510 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*, and *NLSPATH* provide for the support of
 5511 internationalized applications. The standard utilities shall make use of these environment
 5512 variables as described in this section and the individual ENVIRONMENT VARIABLES sections
 5513 for the utilities. If these variables specify locale categories that are not based upon the same
 5514 underlying codeset, the results are unspecified.

5515 The values of locale categories shall be determined by a precedence order; the first condition met
 5516 below determines the value:

- 5517 1. If the *LC_ALL* environment variable is defined and is not null, the value of *LC_ALL* shall
 5518 be used.
- 5519 2. If the *LC_** environment variable (*LC_COLLATE*, *LC_CTYPE*, *LC_MESSAGES*,
 5520 *LC_MONETARY*, *LC_NUMERIC*, *LC_TIME*) is defined and is not null, the value of the
 5521 environment variable shall be used to initialize the category that corresponds to the
 5522 environment variable.
- 5523 3. If the *LANG* environment variable is defined and is not null, the value of the *LANG*
 5524 environment variable shall be used.
- 5525 4. If the *LANG* environment variable is not set or is set to the empty string, the
 5526 implementation-defined default locale shall be used.

5527 If the locale value is "C" or "POSIX", the POSIX locale shall be used and the standard utilities
 5528 behave in accordance with the rules in [Section 7.2](#) (on page 122) for the associated category.

5529 If the locale value begins with a slash, it shall be interpreted as the pathname of a file that was
 5530 created in the output format used by the *localedef* utility; see OUTPUT FILES under *localedef*.
 5531 Referencing such a pathname shall result in that locale being used for the indicated category.

5532 XSI If the locale value has the form:

5533 `language[_territory][.codeset]`

5534 it refers to an implementation-provided locale, where settings of language, territory, and codeset
5535 are implementation-defined.

5536 `LC_COLLATE`, `LC_CTYPE`, `LC_MESSAGES`, `LC_MONETARY`, `LC_NUMERIC`, and `LC_TIME` are
5537 defined to accept an additional field `@modifier`, which allows the user to select a specific instance
5538 of localization data within a single category (for example, for selecting the dictionary as opposed
5539 to the character ordering of data). The syntax for these environment variables is thus defined as:

5540 `[language[_territory][.codeset][@modifier]]`

5541 For example, if a user wanted to interact with the system in French, but required to sort German
5542 text files, `LANG` and `LC_COLLATE` could be defined as:

5543 `LANG=Fr_FR`
5544 `LC_COLLATE=De_DE`

5545 This could be extended to select dictionary collation (say) by use of the `@modifier` field; for
5546 example:

5547 `LC_COLLATE=De_DE@dict`

5548 An implementation may support other formats.

5549 If the locale value is not recognized by the implementation, the behavior is unspecified.

5550 At runtime, these values are bound to the locale of a process by calling the `setlocale()` function.

5551 Additional criteria for determining a valid locale name are implementation-defined.

5552 8.3 Other Environment Variables

5553 **`COLUMNS`** This variable shall represent a decimal integer >0 used to indicate the user's
5554 preferred width in column positions for the terminal screen or window; see
5555 [Section 3.102](#) (on page 47). If this variable is unset or null, the implementation
5556 determines the number of columns, appropriate for the terminal or window,
5557 in an unspecified manner. When `COLUMNS` is set, any terminal-width
5558 information implied by `TERM` is overridden. Users and conforming
5559 applications should not set `COLUMNS` unless they wish to override the
5560 system selection and produce output unrelated to the terminal characteristics.

5561 Users should not need to set this variable in the environment unless there is a
5562 specific reason to override the implementation's default behavior, such as to
5563 display data in an area arbitrarily smaller than the terminal or window.

5564 XSI **`DATEMSK`** Indicates the pathname of the template file used by `getdate()`.

5565 **`HOME`** The system shall initialize this variable at the time of login to be a pathname of
5566 the user's home directory. See [<pwd.h>](#).

5567 **`LINES`** This variable shall represent a decimal integer >0 used to indicate the user's
5568 preferred number of lines on a page or the vertical screen or window size in
5569 lines. A line in this case is a vertical measure large enough to hold the tallest
5570 character in the character set being displayed. If this variable is unset or null,

5571		the implementation determines the number of lines, appropriate for the
5572		terminal or window (size, terminal baud rate, and so on), in an unspecified
5573		manner. When <i>LINES</i> is set, any terminal-height information implied by
5574		<i>TERM</i> is overridden. Users and conforming applications should not set <i>LINES</i>
5575		unless they wish to override the system selection and produce output
5576		unrelated to the terminal characteristics.
5577		Users should not need to set this variable in the environment unless there is a
5578		specific reason to override the implementation's default behavior, such as to
5579		display data in an area arbitrarily smaller than the terminal or window.
5580	<i>LOGNAME</i>	The system shall initialize this variable at the time of login to be the user's
5581		login name. See <pwd.h> . For a value of <i>LOGNAME</i> to be portable across
5582		implementations of POSIX.1-200x, the value should be composed of characters
5583		from the portable filename character set.
5584	XSI <i>MSGVERB</i>	Describes which message components shall be used in writing messages by
5585		<i>fmtmsg()</i> .
5586	<i>PATH</i>	This variable shall represent the sequence of path prefixes that certain
5587		functions and utilities apply in searching for an executable file known only by
5588		a filename. The prefixes shall be separated by a colon (':'). When a non-
5589		zero-length prefix is applied to this filename, a slash shall be inserted between
5590		the prefix and the filename. A zero-length prefix is a legacy feature that
5591		indicates the current working directory. It appears as two adjacent colons
5592		("::"), as an initial colon preceding the rest of the list, or as a trailing colon
5593		following the rest of the list. A strictly conforming application shall use an
5594		actual pathname (such as <i>.</i>) to represent the current working directory in
5595		<i>PATH</i> . The list shall be searched from beginning to end, applying the filename
5596		to each prefix, until an executable file with the specified name and appropriate
5597		execution permissions is found. If the pathname being sought contains a slash,
5598		the search through the path prefixes shall not be performed. If the pathname
5599		begins with a slash, the specified path is resolved (see Section 4.12 , on page
5600		99). If <i>PATH</i> is unset or is set to null, the path search is implementation-
5601		defined.
5602	<i>PWD</i>	This variable shall represent an absolute pathname of the current working
5603		directory. It shall not contain any filename components of dot or dot-dot. The
5604		value is set by the <i>cd</i> utility.
5605	<i>SHELL</i>	This variable shall represent a pathname of the user's preferred command
5606		language interpreter. If this interpreter does not conform to the Shell
5607		Command Language in XCU Chapter 2 (on page 2245), utilities may behave
5608		differently from those described in POSIX.1-200x.
5609	<i>TMPDIR</i>	This variable shall represent a pathname of a directory made available for
5610		programs that need a place to create temporary files.
5611	<i>TERM</i>	This variable shall represent the terminal type for which output is to be
5612		prepared. This information is used by utilities and application programs
5613		wishing to exploit special capabilities specific to a terminal. The format and
5614		allowable values of this environment variable are unspecified.
5615	<i>TZ</i>	This variable shall represent timezone information. The contents of the
5616		environment variable named <i>TZ</i> shall be used by the <i>ctime()</i> , <i>ctime_r()</i> ,
5617		<i>localtime()</i> , <i>localtime_r()</i> , <i>strftime()</i> , <i>mktime()</i> , functions, and by various
5618		utilities, to override the default timezone. The value of <i>TZ</i> has one of the two

5619 forms (spaces inserted for clarity):

5620 *:characters*

5621 or:

5622 *std offset dst offset, rule*

5623 If *TZ* is of the first format (that is, if the first character is a colon), the
5624 characters following the colon are handled in an implementation-defined
5625 manner.

5626 The expanded format (for all *TZs* whose value does not have a colon as the
5627 first character) is as follows:

5628 *stdoffset[dst[offset]][,start[/time],end[/time]]*

5629 Where:

5630 *std* and *dst* Indicate no less than three, nor more than {TZNAME_MAX},
5631 bytes that are the designation for the standard (*std*) or the
5632 alternative (*dst*—such as Daylight Savings Time) timezone. Only
5633 *std* is required; if *dst* is missing, then the alternative time does
5634 not apply in this locale.

5635 Each of these fields may occur in either of two formats quoted or
5636 unquoted:

5637 — In the quoted form, the first character shall be the less-than
5638 ('<') character and the last character shall be the greater-
5639 than ('>') character. All characters between these quoting
5640 characters shall be alphanumeric characters from the portable character set in the current locale, the plus-sign
5641 ('+') character, or the minus-sign ('-') character. The *std*
5642 and *dst* fields in this case shall not include the quoting
5643 characters.
5644

5645 — In the unquoted form, all characters in these fields shall be
5646 alphabetic characters from the portable character set in the
5647 current locale.

5648 The interpretation of these fields is unspecified if either field is
5649 less than three bytes (except for the case when *dst* is missing),
5650 more than {TZNAME_MAX} bytes, or if they contain characters
5651 other than those specified.

5652 *offset* Indicates the value added to the local time to arrive at
5653 Coordinated Universal Time. The *offset* has the form:

5654 *hh[:mm[:ss]]*

5655 The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*)
5656 shall be required and may be a single digit. The *offset* following
5657 *std* shall be required. If no *offset* follows *dst*, the alternative time
5658 is assumed to be one hour ahead of standard time. One or more
5659 digits may be used; the value is always interpreted as a decimal
5660 number. The hour shall be between zero and 24, and the minutes
5661 (and seconds)—if present—between zero and 59. The result of
5662 using values outside of this range is unspecified. If preceded by
5663 a '-', the timezone shall be east of the Prime Meridian;

5664 otherwise, it shall be west (which may be indicated by an
 5665 optional preceding '+').

5666 *rule* Indicates when to change to and back from the alternative time.
 5667 The *rule* has the form:

5668 *date[/time],date[/time]*

5669 where the first *date* describes when the change from standard to
 5670 alternative time occurs and the second *date* describes when the
 5671 change back happens. Each *time* field describes when, in current
 5672 local time, the change to the other time is made.

5673 The format of *date* is one of the following:

5674 *Jn* The Julian day n ($1 \leq n \leq 365$). Leap days shall not be
 5675 counted. That is, in all years—including leap years—
 5676 February 28 is day 59 and March 1 is day 60. It is
 5677 impossible to refer explicitly to the occasional February
 5678 29.

5679 *n* The zero-based Julian day ($0 \leq n \leq 365$). Leap days shall
 5680 be counted, and it is possible to refer to February 29.

5681 *Mm.n.d* The d 'th day ($0 \leq d \leq 6$) of week n of month m of the
 5682 year ($1 \leq n \leq 5$, $1 \leq m \leq 12$, where week 5 means "the last
 5683 d day in month m " which may occur in either the fourth
 5684 or the fifth week). Week 1 is the first week in which the
 5685 d 'th day occurs. Day zero is Sunday.

5686 The *time* has the same format as *offset* except that no leading sign
 5687 ('-' or '+') is allowed. The default, if *time* is not given, shall be
 5688 02:00:00.

Regular Expressions

5689

5690

5691

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

5692

5693

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

5694

5695

5696

5697

5698

The Basic Regular Expression (BRE) notation and construction rules in [Section 9.3](#) (on page 169) shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in [Section 9.4](#) (on page 174); any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in the System Interfaces volume of POSIX.1-200x under *regcomp()*, *regexec()*, and related functions.

5699

5700

5701

5702

5703

5704

9.1 Regular Expression Definitions

5705

For the purposes of this section, the following definitions shall apply:

5706

entire regular expression

The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

5708

5709

matched

A sequence of zero or more characters shall be said to be matched by a BRE or ERE when the characters in the sequence correspond to a sequence of characters defined by the pattern.

5710

5711

5712

5713

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one codeset, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

5714

5715

5716

5717

5718

5719

The search for a matching sequence starts at the beginning of a string and stops when the first sequence matching the expression is found, where “first” is defined to mean “begins earliest in the string”. If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence is matched. For example, the BRE “bb*” matches the second to fourth characters of the string “abbbc”, and the ERE “(wee|week)(knights|night)” matches all ten characters of the string “weeknights”.

5720

5721

5722

5723

5724

5725

5726

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE “\(. *\) . *” against “abcdef”, the subexpression “(\1)” is “abcdef”, and matching

5727

5728

5729

5730 the BRE "`\(a*\)`" against "bc", the subexpression "`(\1)`" is the null string.

5731 When a multi-character collating element in a bracket expression (see [Section 9.3.5](#), on page
5732 170) is involved, the longest sequence shall be measured in characters consumed from the
5733 string to be matched; that is, the collating element counts not as one element, but as the
5734 number of characters it matches.

5735 **BRE (ERE) matching a single character**

5736 A BRE or ERE that shall match either a single character or a single collating element.

5737 Only a BRE or ERE of this type that includes a bracket expression (see [Section 9.3.5](#), on page
5738 170) can match a collating element.

5739 **BRE (ERE) matching multiple characters**

5740 A BRE or ERE that shall match a concatenation of single characters or collating elements.

5741 Such a BRE or ERE is made up from a BRE (ERE) matching a single character and BRE
5742 (ERE) special characters.

5743 **invalid**

5744 This section uses the term "invalid" for certain constructs or conditions. Invalid REs shall
5745 cause the utility or function using the RE to generate an error condition. When invalid is not
5746 used, violations of the specified syntax or semantics for REs produce undefined results: this
5747 may entail an error, enabling an extended syntax for that RE, or using the construct in error
5748 as literal characters to be matched. For example, the BRE construct "`\{1, 2, 3\}`" does not
5749 comply with the grammar. A conforming application cannot rely on it producing an error
5750 nor matching the literal characters "`\{1, 2, 3\}`".

5751 **9.2 Regular Expression General Requirements**

5752 The requirements in this section shall apply to both basic and extended regular expressions.

5753 The use of regular expressions is generally associated with text processing. REs (BREs and EREs)
5754 operate on text strings; that is, zero or more characters followed by an end-of-string delimiter
5755 (typically NUL). Some utilities employing regular expressions limit the processing to lines; that
5756 is, zero or more characters followed by a <newline>. In the regular expression processing
5757 described in POSIX.1-200x, the <newline> is regarded as an ordinary character and both a
5758 period and a non-matching list can match one. The Shell and Utilities volume of POSIX.1-200x
5759 specifies within the individual descriptions of those standard utilities employing regular
5760 expressions whether they permit matching of <newline>s; if not stated otherwise, the use of
5761 literal <newline>s or any escape sequence equivalent produces undefined results. Those utilities
5762 (like *grep*) that do not allow <newline>s to match are responsible for eliminating any <newline>
5763 from strings before matching against the RE. The *regcomp()* function in the System Interfaces
5764 volume of POSIX.1-200x, however, can provide support for such processing without violating
5765 the rules of this section.

5766 The interfaces specified in POSIX.1-200x do not permit the inclusion of a NUL character in an RE
5767 or in the string to be matched. If during the operation of a standard utility a NUL is included in
5768 the text designated to be matched, that NUL may designate the end of the text string for the
5769 purposes of matching.

5770 When a standard utility or function that uses regular expressions specifies that pattern matching
5771 shall be performed without regard to the case (uppercase or lowercase) of either data or
5772 patterns, then when each character in the string is matched against the pattern, not only the
5773 character, but also its case counterpart (if any), shall be matched. This definition of case-
5774 insensitive processing is intended to allow matching of multi-character collating elements as
5775 well as characters, as each character in the string is matched using both its cases. For example, in

5776 a locale where "Ch" is a multi-character collating element and where a matching list expression
 5777 matches such elements, the RE "[[.Ch.]]" when matched against the string "char" is in
 5778 reality matched against "ch", "Ch", "cH", and "CH".

5779 The implementation shall support any regular expression that does not exceed 256 bytes in
 5780 length.

5781 9.3 Basic Regular Expressions

5782 9.3.1 BREs Matching a Single Character or Collating Element

5783 A BRE ordinary character, a special character preceded by a backslash, or a period shall match a
 5784 single character. A bracket expression shall match a single character or a single collating
 5785 element.

5786 9.3.2 BRE Ordinary Characters

5787 An ordinary character is a BRE that matches itself: any character in the supported character set,
 5788 except for the BRE special characters listed in [Section 9.3.3](#).

5789 The interpretation of an ordinary character preceded by a backslash ('*ch*') is undefined, except
 5790 for:

- 5791 • The characters ') ', ' (', ' { ', and ' } '
- 5792 • The digits 1 to 9 inclusive (see [Section 9.3.6](#), on page 172)
- 5793 • A character inside a bracket expression

5794 9.3.3 BRE Special Characters

5795 A BRE special character has special properties in certain contexts. Outside those contexts, or
 5796 when preceded by a backslash, such a character is a BRE that matches the special character itself.
 5797 The BRE special characters and the contexts in which they have their special meaning are as
 5798 follows:

5799 . [*ch* The period, left-bracket, and backslash shall be special except when used in a bracket
 5800 expression (see [Section 9.3.5](#), on page 170). An expression containing a '*ch*' that is not
 5801 preceded by a backslash and is not part of a bracket expression produces undefined
 5802 results.

5803 * The asterisk shall be special except when used:

- 5804 • In a bracket expression
- 5805 • As the first character of an entire BRE (after an initial '*ch*', if any)
- 5806 • As the first character of a subexpression (after an initial '*ch*', if any); see [Section](#)
 5807 [9.3.6](#) (on page 172)

5808 ^ The circumflex shall be special when used as:

- 5809 • An anchor (see [Section 9.3.8](#), on page 173)
- 5810 • The first character of a bracket expression (see [Section 9.3.5](#), on page 170)

5811 \$ The dollar sign shall be special when used as an anchor.

5812 **9.3.4 Periods in BREs**

5813 A period (' . '), when used outside a bracket expression, is a BRE that shall match any character
5814 in the supported character set except NUL.

5815 **9.3.5 RE Bracket Expression**

5816 A bracket expression (an expression enclosed in square brackets, "[]") is an RE that shall
5817 match a single collating element contained in the non-empty set of collating elements
5818 represented by the bracket expression.

5819 The following rules and definitions apply to bracket expressions:

5820 1. A bracket expression is either a matching list expression or a non-matching list
5821 expression. It consists of one or more expressions: collating elements, collating symbols,
5822 equivalence classes, character classes, or range expressions. The right-bracket ('] ') shall
5823 lose its special meaning and represent itself in a bracket expression if it occurs first in the
5824 list (after an initial circumflex (' ^ '), if any). Otherwise, it shall terminate the bracket
5825 expression, unless it appears in a collating symbol (such as "[.] . ") or is the ending
5826 right-bracket for a collating symbol, equivalence class, or character class. The special
5827 characters ' . ', ' * ', ' [', and ' \ ' (period, asterisk, left-bracket, and backslash,
5828 respectively) shall lose their special meaning within a bracket expression.

5829 The character sequences "[. ", "[= ", and "[: " (left-bracket followed by a period,
5830 equals-sign, or colon) shall be special inside a bracket expression and are used to delimit
5831 collating symbols, equivalence class expressions, and character class expressions. These
5832 symbols shall be followed by a valid expression and the matching terminating sequence
5833 " .] ", " =] ", or " :] ", as described in the following items.

5834 2. A matching list expression specifies a list that shall match any single-character collating
5835 element in any of the expressions represented in the list. The first character in the list shall
5836 not be the circumflex; for example, "[abc]" is an RE that matches any of the characters
5837 ' a ', ' b ', or ' c '. It is unspecified whether a matching list expression matches a multi-
5838 character collating element that is matched by one of the expressions.

5839 3. A non-matching list expression begins with a circumflex (' ^ '), and specifies a list that
5840 shall match any single-character collating element except for the expressions represented
5841 in the list after the leading circumflex. For example, "[^ abc]" is an RE that matches any
5842 character except the characters ' a ', ' b ', or ' c '. It is unspecified whether a non-
5843 matching list expression matches a multi-character collating element that is not matched
5844 by any of the expressions. The circumflex shall have this special meaning only when it
5845 occurs first in the list, immediately following the left-bracket.

5846 4. A collating symbol is a collating element enclosed within bracket-period (" [. " and
5847 " .] ") delimiters. Collating elements are defined as described in [Section 7.3.2.4](#) (on page
5848 134). Conforming applications shall represent multi-character collating elements as
5849 collating symbols when it is necessary to distinguish them from a list of the individual
5850 characters that make up the multi-character collating element. For example, if the string
5851 " ch " is a collating element defined using the line:

```
5852           collating-element <ch-digraph> from "<c><h>"
```

5853 in the locale definition, the expression "[[. ch .]]" shall be treated as an RE containing
5854 the collating symbol ' ch ', while "[ch]" shall be treated as an RE matching ' c ' or ' h '.
5855 Collating symbols are recognized only inside bracket expressions. If the string is not a

collating element in the current locale, the expression is invalid.

- 5856
- 5857
- 5858
- 5859
- 5860
- 5861
- 5862
- 5863
- 5864
- 5865
5. An equivalence class expression shall represent the set of collating elements belonging to an equivalence class, as described in [Section 7.3.2.4](#) (on page 134). Only primary equivalence classes shall be recognized. The class shall be expressed by enclosing any one of the collating elements in the equivalence class within bracket-equal (" [= " and "=]") delimiters. For example, if 'a', 'â', and '^' belong to the same equivalence class, then "[[=a=]b]", "[[=â=]b]", and "[[=^=]b]" are each equivalent to "[aâ^b]". If the collating element does not belong to an equivalence class, the equivalence class expression shall be treated as a collating symbol.
 6. A character class expression shall represent the union of two sets:
 - a. The set of single-character collating elements whose characters belong to the character class, as defined in the *LC_CTYPE* category in the current locale.
 - b. An unspecified set of multi-character collating elements.

5866

5867

5868

5869

5870

5871

All character classes specified in the current locale shall be recognized. A character class expression is expressed as a character class name enclosed within bracket-colon (" [: " and " :]") delimiters.

5872

The following character class expressions shall be supported in all locales:

5873

5874

5875

[:alnum:]	[:cntrl:]	[:lower:]	[:space:]
[:alpha:]	[:digit:]	[:print:]	[:upper:]
[:blank:]	[:graph:]	[:punct:]	[:xdigit:]

5876

In addition, character class expressions of the form:

5877

[:name:]

5878

5879

are recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

- 5880
- 5881
- 5882
- 5883
- 5884
- 5885
7. In the POSIX locale, a range expression represents the set of collating elements that fall between two elements in the collation sequence, inclusive. In other locales, a range expression has unspecified behavior: strictly conforming applications shall not rely on whether the range expression is valid, or on the set of collating elements matched. A range expression shall be expressed as the starting point and the ending point separated by a hyphen ('-').

5886

In the following, all examples assume the POSIX locale.

5887

5888

5889

5890

5891

5892

The starting range point and the ending range point shall be a collating element or collating symbol. An equivalence class expression used as a starting or ending point of a range expression produces unspecified results. An equivalence class can be used portably within a bracket expression, but only outside the range. If the represented set of collating elements is empty, it is unspecified whether the expression matches nothing, or is treated as invalid.

5893

5894

The interpretation of range expressions where the ending range point is also the starting range point of a subsequent range expression (for example, "[a-m-o]") is undefined.

5895

5896

5897

5898

5899

5900

The hyphen character shall be treated as itself if it occurs first (after an initial '^', if any) or last in the list, or as an ending range point in a range expression. As examples, the expressions "[-ac]" and "[ac-]" are equivalent and match any of the characters 'a', 'c', or '-'; "[^-ac]" and "[^ac-]" are equivalent and match any characters except 'a', 'c', or '-'; the expression "[%- -]" matches any of the characters between '%' and '-' inclusive; the expression "[--@]" matches any of the characters between '-'

5901 and '@' inclusive; and the expression "[a--@]" is either invalid or equivalent to '@',
 5902 because the letter 'a' follows the symbol '-' in the POSIX locale. To use a hyphen as the
 5903 starting range point, it shall either come first in the bracket expression or be specified as a
 5904 collating symbol; for example, "[][.-.]-0]", which matches either a right bracket or
 5905 any character or collating element that collates between hyphen and 0, inclusive.

5906 If a bracket expression specifies both '-' and ']', the ']' shall be placed first (after the
 5907 '^', if any) and the '-' last within the bracket expression.

5908 9.3.6 BREs Matching Multiple Characters

5909 The following rules can be used to construct BREs matching multiple characters from BREs
 5910 matching a single character:

- 5911 1. The concatenation of BREs shall match the concatenation of the strings matched by each
 5912 component of the BRE.
- 5913 2. A subexpression can be defined within a BRE by enclosing it between the character pairs
 5914 "\" and "\". Such a subexpression shall match whatever it would have matched
 5915 without the "\" and "\" , except that anchoring within subexpressions is optional
 5916 behavior; see [Section 9.3.8](#) (on page 173). Subexpressions can be arbitrarily nested.
- 5917 3. The back-reference expression '\n' shall match the same (possibly empty) string of
 5918 characters as was matched by a subexpression enclosed between "\" and "\"
 5919 preceding the '\n'. The character 'n' shall be a digit from 1 through 9, specifying the
 5920 *n*th subexpression (the one that begins with the *n*th "\" from the beginning of the
 5921 pattern and ends with the corresponding paired "\"). The expression is invalid if less
 5922 than *n* subexpressions precede the '\n'. The string matched by a contained
 5923 subexpression shall be within the string matched by the containing subexpression. If the
 5924 containing subexpression does not match, or if there is no match for the contained
 5925 subexpression within the string matched by the containing subexpression, then back-
 5926 reference expressions corresponding to the contained subexpression shall not match.
 5927 When a subexpression matches more than one string, a back-reference expression
 5928 corresponding to the subexpression shall refer to the last matched string. For example, the
 5929 expression "^(.*)\1\$" matches lines consisting of two adjacent appearances of the
 5930 same string, and the expression "\(a)*\1" fails to match 'a', the expression
 5931 "\(a\(b)*)*\2" fails to match 'abab', and the expression "^(ab*)*\1\$" matches
 5932 'ababbabb', but fails to match 'ababbab'.
- 5933 4. When a BRE matching a single character, a subexpression, or a back-reference is followed
 5934 by the special character asterisk ('*'), together with that asterisk it shall match what zero
 5935 or more consecutive occurrences of the BRE would match. For example, "[ab]*" and
 5936 "[ab][ab]" are equivalent when matching the string "ab".
- 5937 5. When a BRE matching a single character, a subexpression, or a back-reference is followed
 5938 by an interval expression of the format "{m}", "{m,}", or "{m,n}", together
 5939 with that interval expression it shall match what repeated consecutive occurrences of the
 5940 BRE would match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences
 5941 and *n* specifies the maximum number of occurrences. The expression "{m}" shall
 5942 match exactly *m* occurrences of the preceding BRE, "{m,}" shall match at least *m*
 5943 occurrences, and "{m,n}" shall match any number of occurrences between *m* and *n*,
 5944 inclusive.
 5945

5946 For example, in the string "abababcccccd" the BRE "c\{3}" is matched by
 5947 characters seven to nine, the BRE "\(ab*)\{4,}" is not matched at all, and the BRE
 5948 "c\{1,3}d" is matched by characters ten to thirteen.

5949 The behavior of multiple adjacent duplication symbols ('*' and intervals) produces undefined
5950 results.

5951 A subexpression repeated by an asterisk ('*') or an interval expression shall not match a null
5952 expression unless this is the only match for the repetition or it is necessary to satisfy the exact or
5953 minimum number of occurrences for the interval expression.

5954 9.3.7 BRE Precedence

5955 The order of precedence shall be as shown in the following table:

BRE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Subexpressions/back-references	\(\) \n
Single-character-BRE duplication	* \{m,n\}
Concatenation	
Anchoring	^ \$

5964 9.3.8 BRE Expression Anchoring

5965 A BRE can be limited to matching strings that begin or end a line; this is called "anchoring".
5966 The circumflex and dollar sign special characters shall be considered BRE anchors in the
5967 following contexts:

- 5968 1. A circumflex ('^') shall be an anchor when used as the first character of an entire BRE.
5969 The implementation may treat the circumflex as an anchor when used as the first
5970 character of a subexpression. The circumflex shall anchor the expression (or optionally
5971 subexpression) to the beginning of a string; only sequences starting at the first character
5972 of a string shall be matched by the BRE. For example, the BRE "^ab" matches "ab" in
5973 the string "abcdef", but fails to match in the string "cdefab". The BRE "\(^ab\)"
5974 may match the former string. A portable BRE shall escape a leading circumflex in a
5975 subexpression to match a literal circumflex.
- 5976 2. A dollar sign ('\$') shall be an anchor when used as the last character of an entire BRE.
5977 The implementation may treat a dollar sign as an anchor when used as the last character
5978 of a subexpression. The dollar sign shall anchor the expression (or optionally
5979 subexpression) to the end of the string being matched; the dollar sign can be said to
5980 match the end-of-string following the last character.
- 5981 3. A BRE anchored by both '^' and '\$' shall match only an entire string. For example, the
5982 BRE "^abcdef\$" matches strings consisting only of "abcdef".

5983 9.4 Extended Regular Expressions

5984 The extended regular expression (ERE) notation and construction rules shall apply to utilities
5985 defined as using extended regular expressions; any exceptions to the following rules are noted
5986 in the descriptions of the specific utilities using EREs.

5987 9.4.1 EREs Matching a Single Character or Collating Element

5988 An ERE ordinary character, a special character preceded by a backslash, or a period shall match
5989 a single character. A bracket expression shall match a single character or a single collating
5990 element. An ERE matching a single character enclosed in parentheses shall match the same as
5991 the ERE without parentheses would have matched.

5992 9.4.2 ERE Ordinary Characters

5993 An ordinary character is an ERE that matches itself. An ordinary character is any character in the
5994 supported character set, except for the ERE special characters listed in [Section 9.4.3](#). The
5995 interpretation of an ordinary character preceded by a backslash ('\ ') is undefined.

5996 9.4.3 ERE Special Characters

5997 An ERE special character has special properties in certain contexts. Outside those contexts, or
5998 when preceded by a backslash, such a character shall be an ERE that matches the special
5999 character itself. The extended regular expression special characters and the contexts in which
6000 they shall have their special meaning are as follows:

6001 . [\ (The period, left-bracket, backslash, and left-parenthesis shall be special except when
6002 used in a bracket expression (see [Section 9.3.5](#), on page 170). Outside a bracket
6003 expression, a left-parenthesis immediately followed by a right-parenthesis produces
6004 undefined results.

6005) The right-parenthesis shall be special when matched with a preceding left-parenthesis,
6006 both outside a bracket expression.

6007 * + ? { The asterisk, plus-sign, question-mark, and left-brace shall be special except when used
6008 in a bracket expression (see [Section 9.3.5](#), on page 170). Any of the following uses
6009 produce undefined results:

- 6010 • If these characters appear first in an ERE, or immediately following a vertical-line,
6011 circumflex, or left-parenthesis
- 6012 • If a left-brace is not part of a valid interval expression (see [Section 9.4.6](#), on page
6013 175)

6014 | The vertical-line is special except when used in a bracket expression (see [Section 9.3.5](#),
6015 on page 170). A vertical-line appearing first or last in an ERE, or immediately
6016 following a vertical-line or a left-parenthesis, or immediately preceding a right-
6017 parenthesis, produces undefined results.

6018 ^ The circumflex shall be special when used as:

- 6019 • An anchor (see [Section 9.4.9](#), on page 176)
- 6020 • The first character of a bracket expression (see [Section 9.3.5](#), on page 170)

6021 \$ The dollar sign shall be special when used as an anchor.

9.4.4 Periods in EREs

A period ('.'), when used outside a bracket expression, is an ERE that shall match any character in the supported character set except NUL.

9.4.5 ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see [Section 9.3.5](#) (on page 170).

9.4.6 EREs Matching Multiple Characters

The following rules shall be used to construct EREs matching multiple characters from EREs matching a single character:

1. A concatenation of EREs shall match the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses shall match whatever the concatenation without the parentheses matches. For example, both the ERE "cd" and the ERE "(cd)" are matched by the third and fourth character of the string "abcdefabcdef".
2. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character plus-sign ('+'), together with that plus-sign it shall match what one or more consecutive occurrences of the ERE would match. For example, the ERE "b+(bc)" matches the fourth to seventh characters in the string "acabbbbcde". And, "[ab]+" and "[ab][ab]*" are equivalent.
3. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character asterisk ('*'), together with that asterisk it shall match what zero or more consecutive occurrences of the ERE would match. For example, the ERE "b*c" matches the first character in the string "cabbbbcde", and the ERE "b*cd" matches the third to seventh characters in the string "cabbbbcdebbbbbbcbdbc". And, "[ab]*" and "[ab][ab]" are equivalent when matching the string "ab".
4. When an ERE matching a single character or an ERE enclosed in parentheses is followed by the special character question-mark ('?'), together with that question-mark it shall match what zero or one consecutive occurrences of the ERE would match. For example, the ERE "b?c" matches the second character in the string "acabbbbcde".
5. When an ERE matching a single character or an ERE enclosed in parentheses is followed by an interval expression of the format "{m}", "{m,}", or "{m,n}", together with that interval expression it shall match what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* are decimal integers in the range $0 \leq m \leq n \leq \{RE_DUP_MAX\}$, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression "{m}" matches exactly *m* occurrences of the preceding ERE, "{m,}" matches at least *m* occurrences, and "{m,n}" matches any number of occurrences between *m* and *n*, inclusive.

For example, in the string "abababcccccccd" the ERE "c{3}" is matched by characters seven to nine and the ERE "(ab){2,}" is matched by characters one to six.

The behavior of multiple adjacent duplication symbols ('+', '*', '?', and intervals) produces undefined results.

An ERE matching a single character repeated by an '*', '?', or an interval expression shall not match a null expression unless this is the only match for the repetition or it is necessary to satisfy the exact or minimum number of occurrences for the interval expression.

9.4.7 ERE Alternation

Two EREs separated by the special character vertical-line ('|') shall match a string that is matched by either. For example, the ERE "a(bc|d)" matches the string "abc" and the string "ad". Single characters, or expressions matching single characters, separated by the vertical bar and enclosed in parentheses, shall be treated as an ERE matching a single character.

9.4.8 ERE Precedence

The order of precedence shall be as shown in the following table:

ERE Precedence (from high to low)	
Collation-related bracket symbols	[==] [::] [..]
Escaped characters	\<special character>
Bracket expression	[]
Grouping	()
Single-character-ERE duplication	* + ? {m,n}
Concatenation	
Anchoring	^ \$
Alternation	

For example, the ERE "abba|cde" matches either the string "abba" or the string "cde" (rather than the string "abbade" or "abbcde", because concatenation has a higher order of precedence than alternation).

9.4.9 ERE Expression Anchoring

An ERE can be limited to matching strings that begin or end a line; this is called "anchoring". The circumflex and dollar sign special characters shall be considered ERE anchors when used anywhere outside a bracket expression. This shall have the following effects:

1. A circumflex ('^') outside a bracket expression shall anchor the expression or subexpression it begins to the beginning of a string; such an expression or subexpression can match only a sequence starting at the first character of a string. For example, the EREs "^ab" and "(^ab)" match "ab" in the string "abcdef", but fail to match in the string "cdefab", and the ERE "a^b" is valid, but can never match because the 'a' prevents the expression "^b" from matching starting at the first character.
2. A dollar sign ('\$') outside a bracket expression shall anchor the expression or subexpression it ends to the end of a string; such an expression or subexpression can match only a sequence ending at the last character of a string. For example, the EREs "ef\$" and "(ef\$)" match "ef" in the string "abcdef", but fail to match in the string "cdefab", and the ERE "e\$f" is valid, but can never match because the 'f' prevents the expression "e\$" from matching ending at the last character.

9.5 Regular Expression Grammar

Grammars describing the syntax of both basic and extended regular expressions are presented in this section. The grammar takes precedence over the text. See XCU [Section 1.3](#) (on page 2235).

9.5.1 BRE/ERE Grammar Lexical Conventions

The lexical conventions for regular expressions are as described in this section.

Except as noted, the longest possible token or delimiter beginning at a given point is recognized.

The following tokens are processed (in addition to those string constants shown in the grammar):

COLL_ELEM_SINGLE

Any single-character collating element, unless it is a **META_CHAR**.

COLL_ELEM_MULTI

Any multi-character collating element.

BACKREF

Applicable only to basic regular expressions. The character string consisting of '`\`' followed by a single-digit numeral, '1' to '9'.

DUP_COUNT

Represents a numeric constant. It shall be an integer in the range $0 \leq \text{DUP_COUNT} \leq \{\text{RE_DUP_MAX}\}$. This token is only recognized when the context of the grammar requires it. At all other times, digits not preceded by '`\`' are treated as **ORD_CHAR**.

META_CHAR

One of the characters:

`^` When found first in a bracket expression

`-` When found anywhere but first (after an initial '`^`', if any) or last in a bracket expression, or as the ending range point in a range expression

`]` When found anywhere but first (after an initial '`^`', if any) in a bracket expression

L_ANCHOR

Applicable only to basic regular expressions. The character '`^`' when it appears as the first character of a basic regular expression and when not **QUOTED_CHAR**. The '`^`' may be recognized as an anchor elsewhere; see [Section 9.3.8](#) (on page 173).

ORD_CHAR

A character, other than one of the special characters in **SPEC_CHAR**.

QUOTED_CHAR

In a BRE, one of the character sequences:

`\^` `\.` `*` `\[` `\$` `\\`

In an ERE, one of the character sequences:

`\^` `\.` `\[` `\$` `\(` `\)` `\|`

`*` `\+` `\?` `\{` `\\`

R_ANCHOR

(Applicable only to basic regular expressions.) The character '`$`' when it appears as the last character of a basic regular expression and when not **QUOTED_CHAR**. The '`$`' may be recognized as an anchor elsewhere; see [Section 9.3.8](#) (on page 173).

6140 **SPEC_CHAR** For basic regular expressions, one of the following special characters:

6141 . Anywhere outside bracket expressions

6142 \
6143 [Anywhere outside bracket expressions

6144 ^ When used as an anchor (see [Section 9.3.8](#), on page 173) or when
6145 first in a bracket expression

6146 \$ When used as an anchor

6147 * Anywhere except first in an entire RE, anywhere in a bracket
6148 expression, directly following "\(", directly following an
6149 anchoring '^'

6150 For extended regular expressions, shall be one of the following special
6151 characters found anywhere outside bracket expressions:

6152 ^ . [\$ () |
6153 * + ? { \
6154 (The close-parenthesis shall be considered special in this context only if
6155 matched with a preceding open-parenthesis.)

6156 9.5.2 RE and Bracket Expression Grammar

6157 This section presents the grammar for basic regular expressions, including the bracket
6158 expression grammar that is common to both BREs and EREs.

```
6159   %token    ORD_CHAR QUOTED_CHAR DUP_COUNT
6160   %token    BACKREF L_ANCHOR R_ANCHOR
6161   %token    Back_open_paren Back_close_paren
6162   /*        '\'            '\)'            */
6163   %token    Back_open_brace Back_close_brace
6164   /*        '\{'            '\}'            */
6165   /* The following tokens are for the Bracket Expression
6166        grammar common to both REs and EREs. */
6167   %token    COLL_ELEM_SINGLE COLL_ELEM_MULTI META_CHAR
6168   %token    Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close
6169   /*        '['            ']'            '['            ']'            ':'            ':'            */
6170   %token    class_name
6171   /* class_name is a keyword to the LC_CTYPE locale category */
6172   /* (representing a character class) in the current locale */
6173   /* and is only recognized between [: and :] */
6174   %start    basic_reg_exp
6175   %%
6176   /* -----
6177        Basic Regular Expression
6178        -----
6179   */
6180   basic_reg_exp :            RE_expression
6181                | L_ANCHOR
```

```

6182         | R_ANCHOR
6183         | L_ANCHOR R_ANCHOR
6184         | L_ANCHOR RE_expression
6185         | RE_expression R_ANCHOR
6186         | L_ANCHOR RE_expression R_ANCHOR
6187     ;
6188 RE_expression : simple_RE
6189         | RE_expression simple_RE
6190     ;
6191 simple_RE : nondupl_RE
6192         | nondupl_RE RE_dupl_symbol
6193     ;
6194 nondupl_RE : one_char_or_coll_elem_RE
6195         | Back_open_paren RE_expression Back_close_paren
6196         | BACKREF
6197     ;
6198 one_char_or_coll_elem_RE : ORD_CHAR
6199         | QUOTED_CHAR
6200         | '.'
6201         | bracket_expression
6202     ;
6203 RE_dupl_symbol : '*'
6204         | Back_open_brace DUP_COUNT Back_close_brace
6205         | Back_open_brace DUP_COUNT ',' Back_close_brace
6206         | Back_open_brace DUP_COUNT ',' DUP_COUNT Back_close_brace
6207     ;
6208 /* -----
6209 Bracket Expression
6210 ----- */
6211 bracket_expression : '[' matching_list ']'
6212         | '[' nonmatching_list ']'
6213     ;
6214 matching_list : bracket_list
6215     ;
6216 nonmatching_list : '^' bracket_list
6217     ;
6218 bracket_list : follow_list
6219         | follow_list '-'
6220     ;
6221 follow_list : expression_term
6222         | follow_list expression_term
6223     ;
6224 expression_term : single_expression
6225         | range_expression
6226     ;
6227 single_expression : end_range
6228         | character_class
6229         | equivalence_class
6230     ;
6231 range_expression : start_range end_range
6232         | start_range '-'
6233     ;

```

```

6234         ;
6235     start_range      : end_range '-'
6236         ;
6237     end_range        : COLL_ELEM_SINGLE
6238         | collating_symbol
6239         ;
6240     collating_symbol : Open_dot COLL_ELEM_SINGLE Dot_close
6241         | Open_dot COLL_ELEM_MULTI Dot_close
6242         | Open_dot META_CHAR Dot_close
6243         ;
6244     equivalence_class : Open_equal COLL_ELEM_SINGLE Equal_close
6245         | Open_equal COLL_ELEM_MULTI Equal_close
6246         ;
6247     character_class  : Open_colon class_name Colon_close
6248         ;

```

6249 The BRE grammar does not permit **L_ANCHOR** or **R_ANCHOR** inside "`\(`" and "`\)`" (which
6250 implies that '`^`' and '`$`' are ordinary characters). This reflects the semantic limits on the
6251 application, as noted in [Section 9.3.8](#) (on page 173). Implementations are permitted to extend the
6252 language to interpret '`^`' and '`$`' as anchors in these locations, and as such, conforming
6253 applications cannot use unescaped '`^`' and '`$`' in positions inside "`\(`" and "`\)`" that might
6254 be interpreted as anchors.

6255 9.5.3 ERE Grammar

6256 This section presents the grammar for extended regular expressions, excluding the bracket
6257 expression grammar.

6258 **Note:** The bracket expression grammar and the associated `%token` lines are identical between BREs
6259 and EREs. It has been omitted from the ERE section to avoid unnecessary editorial duplication.

```

6260 %token  ORD_CHAR QUOTED_CHAR DUP_COUNT
6261 %start  extended_reg_exp
6262 %%
6263 /* -----
6264    Extended Regular Expression
6265    ----- */
6266
6267 extended_reg_exp : ERE_branch
6268                 | extended_reg_exp '|' ERE_branch
6269                 ;
6270 ERE_branch      : ERE_expression
6271                 | ERE_branch ERE_expression
6272                 ;
6273 ERE_expression : one_char_or_coll_elem_ERE
6274                 | '^'
6275                 | '$'
6276                 | '(' extended_reg_exp ')'
6277                 | ERE_expression ERE_dupl_symbol
6278                 ;
6279 one_char_or_coll_elem_ERE : ORD_CHAR
6280                           | QUOTED_CHAR
6281                           | '.'
6282                           | bracket_expression

```

```

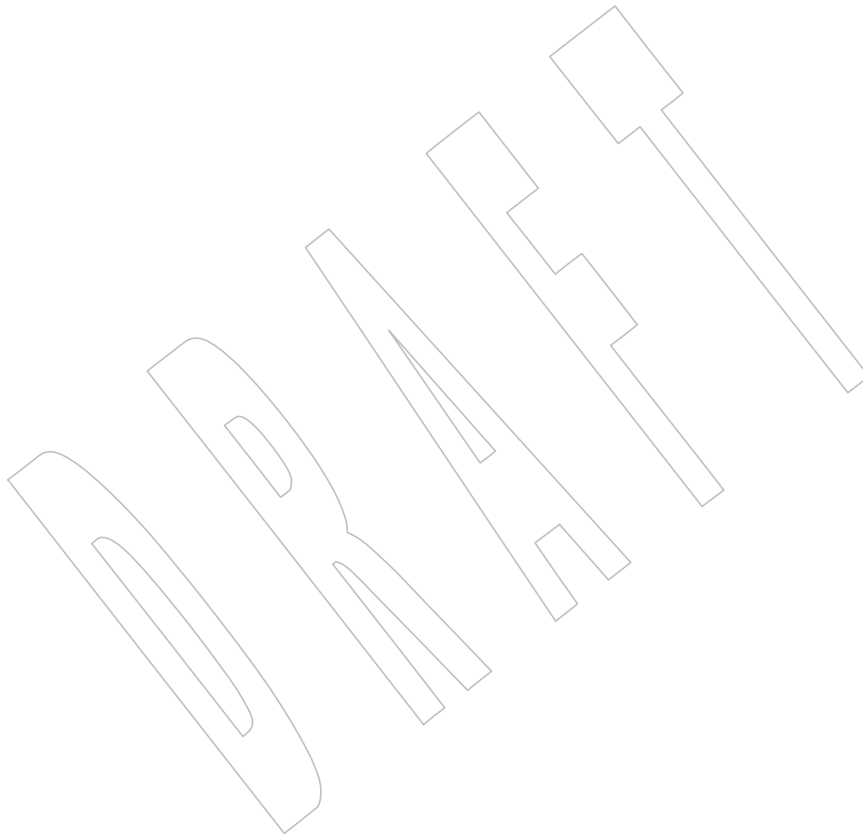
6283                                     ;
6284 ERE_dupl_symbol      : '*'
6285                       | '+'
6286                       | '?'
6287                       | '{' DUP_COUNT '}'
6288                       | '{' DUP_COUNT ',' '}'
6289                       | '{' DUP_COUNT ',' DUP_COUNT '}'
6290                                     ;

```

The ERE grammar does not permit several constructs that previous sections specify as having undefined results:

- **ORD_CHAR** preceded by '`\`'
- One or more *ERE_dupl_symbols* appearing first in an ERE, or immediately following '|', '^', or '('
- '{' not part of a valid *ERE_dupl_symbol*
- '|' appearing first or last in an ERE, or immediately following '|' or '(', or immediately preceding ')'

Implementations are permitted to extend the language to allow these. Conforming applications cannot use such constructs.



*Directory Structure and Devices***10.1 Directory Structure and Files**

The following directories shall exist on conforming systems and conforming applications shall make use of them only as described. Strictly conforming applications shall not assume the ability to create files in any of these directories, unless specified below.

/ The root directory.

/dev Contains **/dev/console**, **/dev/null**, and **/dev/tty**, described below.

The following directory shall exist on conforming systems and shall be used as described:

/tmp A directory made available for applications that need a place to create temporary files. Applications shall be allowed to create files in this directory, but shall not assume that such files are preserved between invocations of the application.

The following files shall exist on conforming systems and shall be both readable and writable:

/dev/null An infinite data source and data sink. Data written to **/dev/null** shall be discarded. Reads from **/dev/null** shall always return end-of-file (EOF).

/dev/tty In each process, a synonym for the controlling terminal associated with the process group of that process, if any. It is useful for programs or shell procedures that wish to be sure of writing messages to or reading data from the terminal no matter how output has been redirected. It can also be used for applications that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

The following file shall exist on conforming systems and need not be readable or writable:

/dev/console The **/dev/console** file is a generic name given to the system console (see [Section 3.384](#), on page 86). It is usually linked to an implementation-defined special file. It shall provide an interface to the system console conforming to the requirements of [Chapter 11](#) (on page 185).

10.2 Output Devices and Terminal Types

The utilities in the Shell and Utilities volume of POSIX.1-200x historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. POSIX.1-200x states which features are optional for certain classes of terminals in the individual utility description sections. The implementation shall document in the system documentation which terminal types it supports and which of these features and utilities are not supported by each terminal.

When a feature or utility is not supported on a specific terminal type, as allowed by POSIX.1-200x, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

POSIX.1-200x uses a notational convention based on historical practice that identifies some of the control characters defined in Section 7.3.1 (on page 124) in a manner easily remembered by users on many terminals. The correspondence between this “<control>-char” notation and the actual control characters is shown in the following table. When POSIX.1-200x refers to a character by its <control>-name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the non-alphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal are interpreted by the system as representing the special characters.

Table 10-1 Control Character Names

Name	Value	Symbolic Name	Name	Value	Symbolic Name
<control>-A	<SOH>	<SOH>	<control>-Q	<DC1>	<DC1>
<control>-B	<STX>	<STX>	<control>-R	<DC2>	<DC2>
<control>-C	<ETX>	<ETX>	<control>-S	<DC3>	<DC3>
<control>-D	<EOT>	<EOT>	<control>-T	<DC4>	<DC4>
<control>-E	<ENQ>	<ENQ>	<control>-U	<NAK>	<NAK>
<control>-F	<ACK>	<ACK>	<control>-V	<SYN>	<SYN>
<control>-G	<BEL>	<alert>	<control>-W	<ETB>	<ETB>
<control>-H	<BS>	<backspace>	<control>-X	<CAN>	<CAN>
<control>-I	<HT>	<tab>	<control>-Y		
<control>-J	<LF>	<linefeed>	<control>-Z	<SUB>	<SUB>
<control>-K	<VT>	<vertical-tab>	<control>-[<ESC>	<ESC>
<control>-L	<FF>	<form-feed>	<control>-\	<FS>	<FS>
<control>-M	<CR>	<carriage-return>	<control>-]	<GS>	<GS>
<control>-N	<SO>	<SO>	<control>-^	<RS>	<RS>
<control>-O	<SI>	<SI>	<control>-_	<US>	<US>
<control>-P	<DLE>	<DLE>	<control>-?		

Note: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

General Terminal Interface

6369

6370

6371

6372

6373

6374

This chapter describes a general terminal interface that shall be provided. It shall be supported on any asynchronous communications ports if the implementation provides them. It is implementation-defined whether it supports network connections or synchronous ports, or both.

6375

11.1 Interface Characteristics

6376

11.1.1 Opening a Terminal Device File

6377

6378

6379

When a terminal device file is opened, it normally causes the thread to wait until a connection is established. In practice, application programs seldom open these files; they are opened by special programs and become an application's standard input, output, and error files.

6380

6381

6382

6383

6384

As described in *open()*, opening a terminal device file with the `O_NONBLOCK` flag clear shall cause the thread to block until the terminal device is ready and available. If `CLOCAL` mode is not set, this means blocking until a connection is established. If `CLOCAL` mode is set in the terminal, or the `O_NONBLOCK` flag is specified in the *open()*, the *open()* function shall return a file descriptor without waiting for a connection to be established.

6385

11.1.2 Process Groups

6386

6387

6388

A terminal may have a foreground process group associated with it. This foreground process group plays a special role in handling signal-generating input characters, as discussed in [Section 11.1.9](#) (on page 189).

6389

6390

6391

6392

6393

6394

A command interpreter process supporting job control can allocate the terminal to different jobs, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's foreground process group may be set or examined by a process, assuming the permission requirements are met; see *tcgetpgrp()* and *tcsetpgrp()*. The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see [Section 11.1.4](#) (on page 186).

6395

6396

6397

6398

6399

6400

6401

When there is no longer any process whose process ID or process group ID matches the foreground process group ID, the terminal shall have no foreground process group. It is unspecified whether the terminal has a foreground process group when there is a process whose process ID matches the foreground process group ID, but whose process group ID does not. No actions defined in POSIX.1-200x, other than allocation of a controlling terminal or a successful call to *tcsetpgrp()*, shall cause a process group to become the foreground process group of the terminal.

6402 11.1.3 The Controlling Terminal

6403 A terminal may belong to a process as its controlling terminal. Each process of a session that has
 6404 a controlling terminal has the same controlling terminal. A terminal may be the controlling
 6405 terminal for at most one session. The controlling terminal for a session is allocated by the session
 6406 leader in an implementation-defined manner. If a session leader has no controlling terminal, and
 6407 opens a terminal device file that is not already associated with a session without using the
 6408 O_NOCTTY option (see *open()*), it is implementation-defined whether the terminal becomes the
 6409 controlling terminal of the session leader. If a process which is not a session leader opens a
 6410 terminal file, or the O_NOCTTY option is used on *open()*, then that terminal shall not become
 6411 the controlling terminal of the calling process. When a controlling terminal becomes associated
 6412 with a session, its foreground process group shall be set to the process group of the session
 6413 leader.

6414 The controlling terminal is inherited by a child process during a *fork()* function call. A process
 6415 relinquishes its controlling terminal when it creates a new session with the *setsid()* function;
 6416 other processes remaining in the old session that had this terminal as their controlling terminal
 6417 continue to have it. Upon the close of the last file descriptor in the system (whether or not it is in
 6418 the current session) associated with the controlling terminal, it is unspecified whether all
 6419 processes that had that terminal as their controlling terminal cease to have any controlling
 6420 terminal. Whether and how a session leader can reacquire a controlling terminal after the
 6421 controlling terminal has been relinquished in this fashion is unspecified. A process does not
 6422 relinquish its controlling terminal simply by closing all of its file descriptors associated with the
 6423 controlling terminal if other processes continue to have it open.

6424 When a controlling process terminates, the controlling terminal is dissociated from the current
 6425 session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by
 6426 other processes in the earlier session may be denied, with attempts to access the terminal treated
 6427 as if a modem disconnect had been sensed.

6428 11.1.4 Terminal Access Control

6429 If a process is in the foreground process group of its controlling terminal, read operations shall
 6430 be allowed, as described in [Section 11.1.5](#) (on page 187). Any attempts by a process in a
 6431 background process group to read from its controlling terminal cause its process group to be
 6432 sent a SIGTTIN signal unless one of the following special cases applies: if the reading process is
 6433 ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is
 6434 orphaned, the *read()* shall return -1 , with *errno* set to [EIO] and no signal shall be sent. The
 6435 default action of the SIGTTIN signal shall be to stop the process to which it is sent. See
 6436 [<signal.h>](#).

6437 If a process is in the foreground process group of its controlling terminal, write operations shall
 6438 be allowed as described in [Section 11.1.8](#) (on page 189). Attempts by a process in a background
 6439 process group to write to its controlling terminal shall cause the process group to be sent a
 6440 SIGTTOU signal unless one of the following special cases applies: if TOSTOP is not set, or if
 6441 TOSTOP is set and the process is ignoring or blocking the SIGTTOU signal, the process is
 6442 allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the
 6443 process group of the writing process is orphaned, and the writing process is not ignoring or
 6444 blocking the SIGTTOU signal, the *write()* shall return -1 , with *errno* set to [EIO] and no signal
 6445 shall be sent.

6446 Certain calls that set terminal parameters are treated in the same fashion as *write()*, except that
 6447 TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set
 6448 (see [Section 11.2.5](#) (on page 195), *tcdrain()*, *tcflow()*, *tcflush()*, *tcsendbreak()*, *tcsetattr()*, and
 6449 *tcsetpgrp()*).

6450 11.1.5 Input Processing and Reading Data

6451 A terminal device associated with a terminal device file may operate in full-duplex mode, so
 6452 that data may arrive even while output is occurring. Each terminal device file has an input
 6453 queue associated with it, into which incoming data is stored by the system before being read by
 6454 a process. The system may impose a limit, {MAX_INPUT}, on the number of bytes that may be
 6455 stored in the input queue. The behavior of the system when this limit is exceeded is
 6456 implementation-defined.

6457 Two general kinds of input processing are available, determined by whether the terminal device
 6458 file is in canonical mode or non-canonical mode. These modes are described in [Section 11.1.6](#) and
 6459 [Section 11.1.7](#) (on page 188). Additionally, input characters are processed according to the *c_iflag*
 6460 (see [Section 11.2.2](#), on page 191) and *c_lflag* (see [Section 11.2.5](#), on page 195) fields. Such
 6461 processing can include “echoing”, which in general means transmitting input characters
 6462 immediately back to the terminal when they are received from the terminal. This is useful for
 6463 terminals that can operate in full-duplex mode.

6464 The manner in which data is provided to a process reading from a terminal device file is
 6465 dependent on whether the terminal file is in canonical or non-canonical mode, and on whether
 6466 or not the O_NONBLOCK flag is set by *open()* or *fcntl()*.

6467 If the O_NONBLOCK flag is clear, then the read request shall be blocked until data is available
 6468 or a signal has been received. If the O_NONBLOCK flag is set, then the read request shall be
 6469 completed, without blocking, in one of three ways:

- 6470 1. If there is enough data available to satisfy the entire request, the *read()* shall complete
 6471 successfully and shall return the number of bytes read.
- 6472 2. If there is not enough data available to satisfy the entire request, the *read()* shall complete
 6473 successfully, having read as much data as possible, and shall return the number of bytes it
 6474 was able to read.
- 6475 3. If there is no data available, the *read()* shall return -1 , with *errno* set to [EAGAIN].

6476 When data is available depends on whether the input processing mode is canonical or non-
 6477 canonical. [Section 11.1.6](#) and [Section 11.1.7](#) (on page 188) describe each of these input processing
 6478 modes.

6479 11.1.6 Canonical Mode Input Processing

6480 In canonical mode input processing, terminal input is processed in units of lines. A line is
 6481 delimited by a newline character (NL), an end-of-file character (EOF), or an end-of-line (EOL)
 6482 character. See [Section 11.1.9](#) (on page 189) for more information on EOF and EOL. This means
 6483 that a read request shall not return until an entire line has been typed or a signal has been
 6484 received. Also, no matter how many bytes are requested in the *read()* call, at most one line shall
 6485 be returned. It is not, however, necessary to read a whole line at once; any number of bytes, even
 6486 one, may be requested in a *read()* without losing information.

6487 If {MAX_CANON} is defined for this terminal device, it shall be a limit on the number of bytes
 6488 in a line. The behavior of the system when this limit is exceeded is implementation-defined. If
 6489 {MAX_CANON} is not defined, there shall be no such limit; see *pathconf()*.

6490 Erase and kill processing occur when either of two special characters, the ERASE and KILL
 6491 characters (see [Section 11.1.9](#), on page 189), is received. This processing shall affect data in the
 6492 input queue that has not yet been delimited by an NL, EOF, or EOL character. This un-delimited
 6493 data makes up the current line. The ERASE character shall delete the last character in the current
 6494 line, if there is one. The KILL character shall delete all data in the current line, if there is any.
 6495 The ERASE and KILL characters shall have no effect if there is no data in the current line. The

6496 ERASE and KILL characters themselves shall not be placed in the input queue.

6497 11.1.7 Non-Canonical Mode Input Processing

6498 In non-canonical mode input processing, input bytes are not assembled into lines, and erase and
 6499 kill processing shall not occur. The values of the MIN and TIME members of the *c_cc* array are
 6500 used to determine how to process the bytes received. POSIX.1-200x does not specify whether
 6501 the setting of O_NONBLOCK takes precedence over MIN or TIME settings. Therefore, if
 6502 O_NONBLOCK is set, *read()* may return immediately, regardless of the setting of MIN or TIME.
 6503 Also, if no data is available, *read()* may either return 0, or return -1 with *errno* set to [EAGAIN].

6504 MIN represents the minimum number of bytes that should be received when the *read()* function
 6505 returns successfully. TIME is a timer of 0.1 second granularity that is used to time out bursty and
 6506 short-term data transmissions. If MIN is greater than {MAX_INPUT}, the response to the request
 6507 is undefined. The four possible values for MIN and TIME and their interactions are described
 6508 below.

6509 Case A: MIN>0, TIME>0

6510 In case A, TIME serves as an inter-byte timer which shall be activated after the first byte is
 6511 received. Since it is an inter-byte timer, it shall be reset after a byte is received. The interaction
 6512 between MIN and TIME is as follows. As soon as one byte is received, the inter-byte timer shall
 6513 be started. If MIN bytes are received before the inter-byte timer expires (remember that the timer
 6514 is reset upon receipt of each byte), the read shall be satisfied. If the timer expires before MIN
 6515 bytes are received, the characters received to that point shall be returned to the user. Note that if
 6516 TIME expires at least one byte shall be returned because the timer would not have been enabled
 6517 unless a byte was received. In this case (MIN>0, TIME>0) the read shall block until the MIN and
 6518 TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is
 6519 in the buffer at the time of the *read()*, the result shall be as if data has been received immediately
 6520 after the *read()*.

6521 Case B: MIN>0, TIME=0

6522 In case B, since the value of TIME is zero, the timer plays no role and only MIN is significant. A
 6523 pending read shall not be satisfied until MIN bytes are received (that is, the pending read shall
 6524 block until MIN bytes are received), or a signal is received. A program that uses case B to read
 6525 record-based terminal I/O may block indefinitely in the read operation.

6526 Case C: MIN=0, TIME>0

6527 In case C, since MIN=0, TIME no longer represents an inter-byte timer. It now serves as a read
 6528 timer that shall be activated as soon as the *read()* function is processed. A read shall be satisfied
 6529 as soon as a single byte is received or the read timer expires. Note that in case C if the timer
 6530 expires, no bytes shall be returned. If the timer does not expire, the only way the read can be
 6531 satisfied is if a byte is received. If bytes are not received, the read shall not block indefinitely
 6532 waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the
 6533 *read()* shall return a value of zero, having read no data. If data is in the buffer at the time of the
 6534 *read()*, the timer shall be started as if data has been received immediately after the *read()*.

6535 **Case D: MIN=0, TIME=0**

6536 The minimum of either the number of bytes requested or the number of bytes currently
 6537 available shall be returned without waiting for more bytes to be input. If no characters are
 6538 available, *read()* shall return a value of zero, having read no data.

6539 **11.1.8 Writing Data and Output Processing**

6540 When a process writes one or more bytes to a terminal device file, they are processed according
 6541 to the *c_oflag* field (see [Section 11.2.3](#), on page 192). The implementation may provide a
 6542 buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have
 6543 been scheduled for transmission to the device, but the transmission has not necessarily
 6544 completed. See *write()* for the effects of *O_NONBLOCK* on *write()*.

6545 **11.1.9 Special Characters**

6546 Certain characters have special functions on input or output or both. These functions are
 6547 summarized as follows:

6548 **INTR** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 6549 *SIGINT* signal which is sent to all processes in the foreground process group for which
 6550 the terminal is the controlling terminal. If *ISIG* is set, the *INTR* character shall be
 6551 discarded when processed.

6552 **QUIT** Special character on input, which is recognized if the *ISIG* flag is set. Generates a
 6553 *SIGQUIT* signal which is sent to all processes in the foreground process group for
 6554 which the terminal is the controlling terminal. If *ISIG* is set, the *QUIT* character shall be
 6555 discarded when processed.

6556 **ERASE** Special character on input, which is recognized if the *ICANON* flag is set. Erases the
 6557 last character in the current line; see [Section 11.1.6](#) (on page 187). It shall not erase
 6558 beyond the start of a line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is
 6559 set, the *ERASE* character shall be discarded when processed.

6560 **KILL** Special character on input, which is recognized if the *ICANON* flag is set. Deletes the
 6561 entire line, as delimited by an *NL*, *EOF*, or *EOL* character. If *ICANON* is set, the *KILL*
 6562 character shall be discarded when processed.

6563 **EOF** Special character on input, which is recognized if the *ICANON* flag is set. When
 6564 received, all the bytes waiting to be read are immediately passed to the process without
 6565 waiting for a newline, and the *EOF* is discarded. Thus, if there are no bytes waiting
 6566 (that is, the *EOF* occurred at the beginning of a line), a byte count of zero shall be
 6567 returned from the *read()*, representing an end-of-file indication. If *ICANON* is set, the
 6568 *EOF* character shall be discarded when processed.

6569 **NL** Special character on input, which is recognized if the *ICANON* flag is set. It is the line
 6570 delimiter newline. It cannot be changed.

6571 **EOL** Special character on input, which is recognized if the *ICANON* flag is set. It is an
 6572 additional line delimiter, like *NL*.

6573 **SUSP** If the *ISIG* flag is set, receipt of the *SUSP* character shall cause a *SIGTSTP* signal to be
 6574 sent to all processes in the foreground process group for which the terminal is the
 6575 controlling terminal, and the *SUSP* character shall be discarded when processed.

6576 **STOP** Special character on both input and output, which is recognized if the *IXON* (output
 6577 control) or *IXOFF* (input control) flag is set. Can be used to suspend output
 6578 temporarily. It is useful with CRT terminals to prevent output from disappearing before

6579 it can be read. If IXON is set, the STOP character shall be discarded when processed.

6580 START Special character on both input and output, which is recognized if the IXON (output
6581 control) or IXOFF (input control) flag is set. Can be used to resume output that has been
6582 suspended by a STOP character. If IXON is set, the START character shall be discarded
6583 when processed.

6584 CR Special character on input, which is recognized if the ICANON flag is set; it is the
6585 carriage-return character. When ICANON and ICRNL are set and IGNCR is not set,
6586 this character shall be translated into an NL, and shall have the same effect as an NL
6587 character.

6588 The NL and CR characters cannot be changed. It is implementation-defined whether the START
6589 and STOP characters can be changed. The values for INTR, QUIT, ERASE, KILL, EOF, EOL, and
6590 SUSP shall be changeable to suit individual tastes. Special character functions associated with
6591 changeable special control characters can be disabled individually.

6592 If two or more special characters have the same value, the function performed when that
6593 character is received is undefined.

6594 A special character is recognized not only by its value, but also by its context; for example, an
6595 implementation may support multi-byte sequences that have a meaning different from the
6596 meaning of the bytes when considered individually. Implementations may also support
6597 additional single-byte functions. These implementation-defined multi-byte or single-byte
6598 functions shall be recognized only if the IEXTEN flag is set; otherwise, data is received without
6599 interpretation, except as required to recognize the special characters defined in this section.

6600 XSI If IEXTEN is set, the ERASE, KILL, and EOF characters can be escaped by a preceding '\'
6601 character, in which case no special function shall occur.

6602 11.1.10 Modem Disconnect

6603 If a modem disconnect is detected by the terminal interface for a controlling terminal, and if
6604 CLOCAL is not set in the *c_flag* field for the terminal (see Section 11.2.4, on page 194), the
6605 SIGHUP signal shall be sent to the controlling process for which the terminal is the controlling
6606 terminal. Unless other arrangements have been made, this shall cause the controlling process to
6607 terminate (see *exit()*). Any subsequent read from the terminal device shall return the value of
6608 zero, indicating end-of-file; see *read()*. Thus, processes that read a terminal file and test for end-
6609 of-file can terminate appropriately after a disconnect. If the EIO condition as specified in *read()*
6610 also exists, it is unspecified whether on EOF condition or [EIO] is returned. Any subsequent
6611 *write()* to the terminal device shall return -1, with *errno* set to [EIO], until the device is closed.

6612 11.1.11 Closing a Terminal Device File

6613 The last process to close a terminal device file shall cause any output to be sent to the device and
6614 any input to be discarded. If HUPCL is set in the control modes and the communications port
6615 supports a disconnect function, the terminal device shall perform a disconnect.

6616 **11.2 Parameters that Can be Set**6617 **11.2.1 The termios Structure**

6618 Routines that need to control certain terminal I/O characteristics shall do so by using the
 6619 **termios** structure as defined in the `<termios.h>` header. The members of this structure include
 6620 (but are not limited to):

Member Type	Array Size	Member Name	Description
<code>tcflag_t</code>		<code>c_iflag</code>	Input modes.
<code>tcflag_t</code>		<code>c_oflag</code>	Output modes.
<code>tcflag_t</code>		<code>c_cflag</code>	Control modes.
<code>tcflag_t</code>		<code>c_lflag</code>	Local modes.
<code>cc_t</code>	NCCS	<code>c_cc[]</code>	Control characters.

6628 The types `tcflag_t` and `cc_t` are defined in the `<termios.h>` header. They shall be unsigned
 6629 integer types.

6630 **11.2.2 Input Modes**

6631 Values of the `c_iflag` field describe the basic terminal input control, and are composed of the
 6632 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6633 symbols in this table are defined in `<termios.h>`:

Mask Name	Description
BRKINT	Signal interrupt on break.
ICRNL	Map CR to NL on input.
IGNBRK	Ignore break condition.
IGNCR	Ignore CR.
IGNPAR	Ignore characters with parity errors.
INLCR	Map NL to CR on input.
INPCK	Enable input parity check.
ISTRIP	Strip character.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IXON	Enable start/stop output control.
PARMRK	Mark parity errors.

6647 In the context of asynchronous serial data transmission, a break condition shall be defined as a
 6648 sequence of zero-valued bits that continues for more than the time to send one byte. The entire
 6649 sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a
 6650 time equivalent to more than one byte. In contexts other than asynchronous serial data
 6651 transmission, the definition of a break condition is implementation-defined.

6652 If IGNBRK is set, a break condition detected on input shall be ignored; that is, not put on the
 6653 input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the
 6654 break condition shall flush the input and output queues, and if the terminal is the controlling
 6655 terminal of a foreground process group, the break condition shall generate a single SIGINT
 6656 signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break
 6657 condition shall be read as a single 0x00, or if PARMRK is set, as 0xff 0x00 0x00.

6658 If IGNPAR is set, a byte with a framing or parity error (other than break) shall be ignored.

6659 If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than
 6660 break) shall be given to the application as the three-byte sequence 0xff 0x00 X, where 0xff 0x00 is
 6661 a two-byte flag preceding each sequence and X is the data of the byte received in error. To avoid
 6662 ambiguity in this case, if ISTRIP is not set, a valid byte of 0xff is given to the application as 0xff
 6663 0xff. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) shall be
 6664 given to the application as a single byte 0x00.

6665 If INPCK is set, input parity checking shall be enabled. If INPCK is not set, input parity checking
 6666 shall be disabled, allowing output parity generation without input parity errors. Note that
 6667 whether input parity checking is enabled or disabled is independent of whether parity detection
 6668 is enabled or disabled (see Section 11.2.4, on page 194). If parity detection is enabled but input
 6669 parity checking is disabled, the hardware to which the terminal is connected shall recognize the
 6670 parity bit, but the terminal special file shall not check whether or not this bit is correctly set.

6671 If ISTRIP is set, valid input bytes shall first be stripped to seven bits; otherwise, all eight bits
 6672 shall be processed.

6673 If INLCR is set, a received NL character shall be translated into a CR character. If IGNCR is set, a
 6674 received CR character shall be ignored (not read). If IGNCR is not set and ICRNL is set, a
 6675 received CR character shall be translated into an NL character.

6676 XSI If IXANY is set, any input character shall restart output that has been suspended.

6677 If IXON is set, start/stop output control shall be enabled. A received STOP character shall
 6678 suspend output and a received START character shall restart output. When IXON is set, START
 6679 and STOP characters are not read, but merely perform flow control functions. When IXON is not
 6680 set, the START and STOP characters shall be read.

6681 If IXOFF is set, start/stop input control shall be enabled. The system shall transmit STOP
 6682 characters, which are intended to cause the terminal device to stop transmitting data, as needed
 6683 to prevent the input queue from overflowing and causing implementation-defined behavior,
 6684 and shall transmit START characters, which are intended to cause the terminal device to resume
 6685 transmitting data, as soon as the device can continue transmitting data without risk of
 6686 overflowing the input queue. The precise conditions under which STOP and START characters
 6687 are transmitted are implementation-defined.

6688 The initial input control value after *open()* is implementation-defined.

6689 11.2.3 Output Modes

6690 The *c_oflag* field specifies the terminal interface's treatment of output, and is composed of the
 6691 bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
 6692 symbols in the following table are defined in [<termios.h>](#):

6693	Mask Name	Description
6694	OPOST	Perform output processing.
6695	XSI	ONLCR
6696	OCRNL	Map NL to CR-NL on output.
6697	ONOCR	Map CR to NL on output.
6698	ONLRET	No CR output at column 0.
6699	ONLRET	NL performs CR function.
6700	OFILL	Use fill characters for delay.
6701	OFDEL	Fill is DEL, else NUL.
6702	NLDLY	Select newline delays:
6703	NL0	Newline character type 0.
6704	NL1	Newline character type 1.
6705	CRDLY	Select carriage-return delays:
6706	CR0	Carriage-return delay type 0.
6707	CR1	Carriage-return delay type 1.
6708	CR2	Carriage-return delay type 2.
6709	CR3	Carriage-return delay type 3.
6710	TABDLY	Select horizontal-tab delays:
6711	TAB0	Horizontal-tab delay type 0.
6712	TAB1	Horizontal-tab delay type 1.
6713	TAB2	Horizontal-tab delay type 2.
6714	TAB3	Expand tabs to spaces.
6715	BSDLY	Select backspace delays:
6716	BS0	Backspace-delay type 0.
6717	BS1	Backspace-delay type 1.
6718	VTDLY	Select vertical-tab delays:
6719	VT0	Vertical-tab delay type 0.
6720	VT1	Vertical-tab delay type 1.
6721	FFDLY	Select form-feed delays:
6722	FF0	Form-feed delay type 0.
6723	FF1	Form-feed delay type 1.

6723 If OPOST is set, output data shall be post-processed as described below, so that lines of text are
 6724 modified to appear appropriately on the terminal device; otherwise, characters shall be
 6725 transmitted without change.

6726 XSI If ONLCR is set, the NL character shall be transmitted as the CR-NL character pair. If OCRNL is
 6727 set, the CR character shall be transmitted as the NL character. If ONOCR is set, no CR character
 6728 shall be transmitted when at column 0 (first position). If ONLRET is set, the NL character is
 6729 assumed to do the carriage-return function; the column pointer shall be set to 0 and the delays
 6730 specified for CR shall be used. Otherwise, the NL character is assumed to do just the line-feed
 6731 function; the column pointer remains unchanged. The column pointer shall also be set to 0 if the
 6732 CR character is actually transmitted.

6733 The delay bits specify how long transmission stops to allow for mechanical or other movement
 6734 when certain characters are sent to the terminal. In all cases a value of 0 shall indicate no delay. If
 6735 OFILL is set, fill characters shall be transmitted for delay instead of a timed delay. This is useful
 6736 for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character
 6737 shall be DEL; otherwise, NUL.

6738 If a form-feed or vertical-tab delay is specified, it shall last for about 2 seconds.

6739 Newline delay shall last about 0.10 seconds. If ONLRET is set, the carriage-return delays shall be
 6740 used instead of the newline delays. If OFILL is set, two fill characters shall be transmitted.

6741 Carriage-return delay type 1 shall be dependent on the current column position, type 2 shall be

6742 about 0.10 seconds, and type 3 shall be about 0.15 seconds. If OFILL is set, delay type 1 shall
6743 transmit two fill characters, and type 2 four fill characters.

6744 Horizontal-tab delay type 1 shall be dependent on the current column position. Type 2 shall be
6745 about 0.10 seconds. Type 3 specifies that tabs shall be expanded into spaces. If OFILL is set, two
6746 fill characters shall be transmitted for any delay.

6747 Backspace delay shall last about 0.05 seconds. If OFILL is set, one fill character shall be
6748 transmitted.

6749 The actual delays depend on line speed and system load.

6750 The initial output control value after *open()* is implementation-defined.

6751 11.2.4 Control Modes

6752 The *c_cflag* field describes the hardware control of the terminal, and is composed of the bitwise-
6753 inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name symbols in
6754 this table are defined in [<termios.h>](#); not all values specified are required to be supported by the
6755 underlying hardware:

Mask Name	Description
CLOCAL	Ignore modem status lines.
CREAD	Enable receiver.
CSIZE	Number of bits transmitted or received per byte:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits.
CSTOPB	Send two stop bits, else one.
HUPCL	Hang up on last close.
PARENB	Parity enable.
PARODD	Odd parity, else even.

6768 In addition, the input and output baud rates are stored in the **termios** structure. The symbols in
6769 the following table are defined in [<termios.h>](#). Not all values specified are required to be
6770 supported by the underlying hardware.

6771 **Note:** The term “baud” is used historically here, but is not technically correct. This is properly “bits +
6772 per second”, which may not be the same as baud. However, the term is used because of the +
6773 historical usage and understanding. +

Name	Description	Name	Description
B0	Hang up	B600	600 baud
B50	50 baud	B1200	1200 baud
B75	75 baud	B1800	1800 baud
B110	110 baud	B2400	2400 baud
B134	134.5 baud	B4800	4800 baud
B150	150 baud	B9600	9600 baud
B200	200 baud	B19200	19200 baud
B300	300 baud	B38400	38400 baud

6783 The following functions are provided for getting and setting the values of the input and output
6784 baud rates in the **termios** structure: *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, and *cfsetospeed()*.
6785 The effects on the terminal device shall not become effective and not all errors need be detected
6786 until the *tcsetattr()* function is successfully called.

6787 The CSIZE bits shall specify the number of transmitted or received bits per byte. If ISTRIP is not
6788 set, the value of all the other bits is unspecified. If ISTRIP is set, the value of all but the 7 low-
6789 order bits shall be zero, but the value of any other bits beyond CSIZE is unspecified when read.
6790 CSIZE shall not include the parity bit, if any. If CSTOPB is set, two stop bits shall be used;
6791 otherwise, one stop bit. For example, at 110 baud, two stop bits are normally used.

6792 If CREAD is set, the receiver shall be enabled; otherwise, no characters shall be received.

6793 If PARENB is set, parity generation and detection shall be enabled and a parity bit is added to
6794 each byte. If parity is enabled, PARODD shall specify odd parity if set; otherwise, even parity
6795 shall be used.

6796 If HUPCL is set, the modem control lines for the port shall be lowered when the last process
6797 with the port open closes the port or the process terminates. The modem connection shall be
6798 broken.

6799 If CLOCAL is set, a connection shall not depend on the state of the modem status lines. If
6800 CLOCAL is clear, the modem status lines shall be monitored.

6801 Under normal circumstances, a call to the `open()` function shall wait for the modem connection
6802 to complete. However, if the `O_NONBLOCK` flag is set (see `open()`) or if CLOCAL has been set,
6803 the `open()` function shall return immediately without waiting for the connection.

6804 If the object for which the control modes are set is not an asynchronous serial connection, some
6805 of the modes may be ignored; for example, if an attempt is made to set the baud rate on a
6806 network connection to a terminal on another host, the baud rate need not be set on the
6807 connection between that terminal and the machine to which it is directly connected.

6808 The initial hardware control value after `open()` is implementation-defined.

6809 11.2.5 Local Modes

6810 The `c_iflag` field of the argument structure is used to control various functions. It is composed of
6811 the bitwise-inclusive OR of the masks shown, which shall be bitwise-distinct. The mask name
6812 symbols in this table are defined in `<termios.h>`; not all values specified are required to be
6813 supported by the underlying hardware:

Mask Name	Description
ECHO	Enable echo.
ECHOE	Echo ERASE as an error correcting backspace.
ECHOK	Echo KILL.
ECHONL	Echo <newline>.
ICANON	Canonical input (erase and kill processing).
IEXTEN	Enable extended (implementation-defined) functions.
ISIG	Enable signals.
NOFLSH	Disable flush after interrupt, quit, or suspend.
TOSTOP	Send SIGTTOU for background output.

6824 If ECHO is set, input characters shall be echoed back to the terminal. If ECHO is clear, input
6825 characters shall not be echoed.

6826 If ECHOE and ICANON are set, the ERASE character shall cause the terminal to erase, if
6827 possible, the last character in the current line from the display. If there is no character to erase, an
6828 implementation may echo an indication that this was the case, or do nothing.

6829 If ECHOK and ICANON are set, the KILL character shall either cause the terminal to erase the
6830 line from the display or shall echo the newline character after the KILL character.

- 6831 If ECHONL and ICANON are set, the newline character shall be echoed even if ECHO is not set.
- 6832 If ICANON is set, canonical processing shall be enabled. This enables the erase and kill edit
6833 functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as
6834 described in Section 11.1.6 (on page 187).
- 6835 If ICANON is not set, read requests shall be satisfied directly from the input queue. A read shall
6836 not be satisfied until at least MIN bytes have been received or the timeout value TIME expired
6837 between bytes. The time value represents tenths of a second. See Section 11.1.7 (on page 188) for
6838 more details.
- 6839 If IEXTEN is set, implementation-defined functions shall be recognized from the input data. It is
6840 implementation-defined how IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF.
6841 If IEXTEN is not set, implementation-defined functions shall not be recognized and the
6842 corresponding input characters are processed as described for ICANON, ISIG, IXON, and
6843 IXOFF.
- 6844 If ISIG is set, each input character shall be checked against the special control characters INTR,
6845 QUIT, and SUSP. If an input character matches one of these control characters, the function
6846 associated with that character shall be performed. If ISIG is not set, no checking shall be done.
6847 Thus these special input functions are possible only if ISIG is set.
- 6848 If NOFLSH is set, the normal flush of the input and output queues associated with the INTR,
6849 QUIT, and SUSP characters shall not be done.
- 6850 If TOSTOP is set, the signal SIGTTOU shall be sent to the process group of a process that tries to
6851 write to its controlling terminal if it is not in the foreground process group for that terminal. This
6852 signal, by default, stops the members of the process group. Otherwise, the output generated by
6853 that process shall be output to the current output stream. Processes that are blocking or ignoring
6854 SIGTTOU signals are excepted and allowed to produce output, and the SIGTTOU signal shall
6855 not be sent.
- 6856 The initial local control value after *open()* is implementation-defined.

6857 11.2.6 Special Control Characters

6858 The special control character values shall be defined by the array *c_cc*. The subscript name and
6859 description for each element in both canonical and non-canonical modes are as follows:

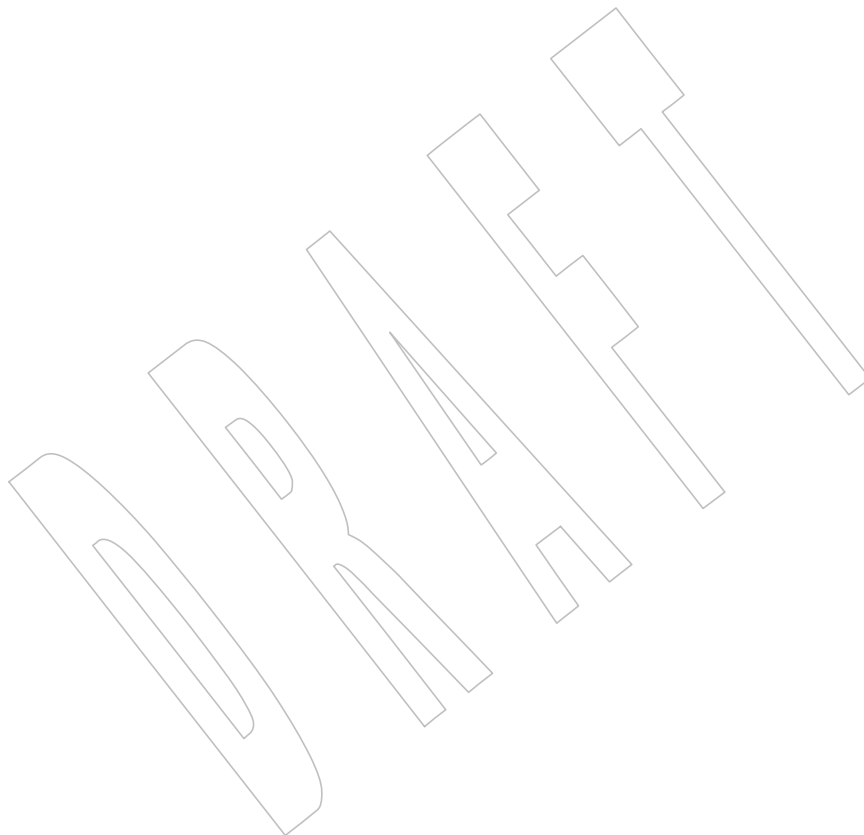
Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character
VEOL		EOL character
VERASE		ERASE character
VINTR	VINTR	INTR character
VKILL		KILL character
	VMIN	MIN value
VQUIT	VQUIT	QUIT character
VSUSP	VSUSP	SUSP character
	VTIME	TIME value
VSTART	VSTART	START character
VSTOP	VSTOP	STOP character

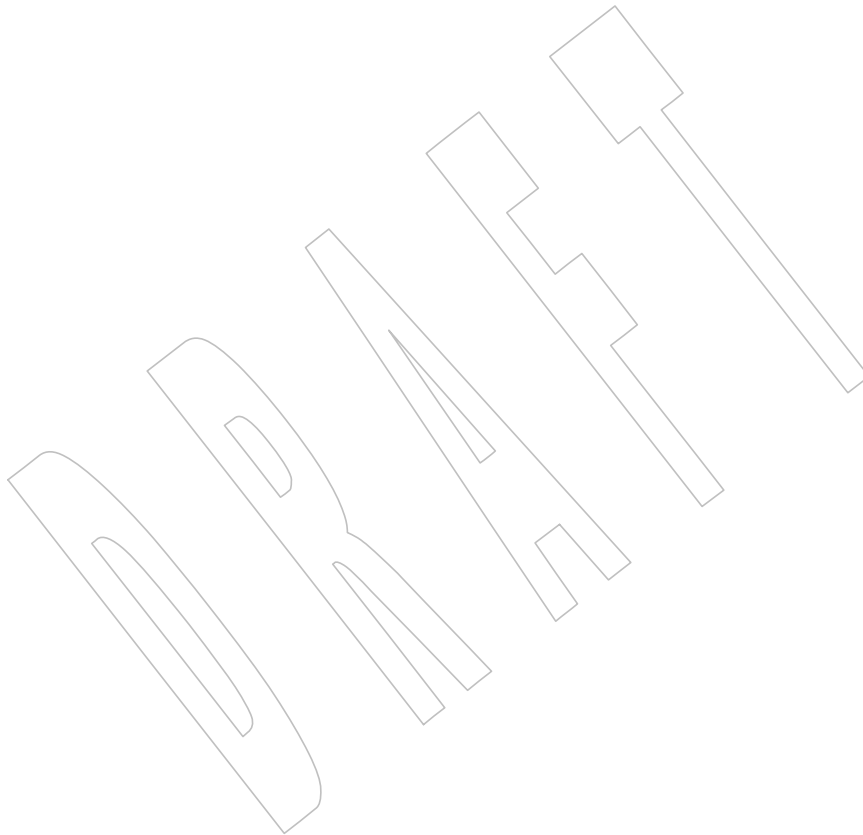
6874 The subscript values are unique, except that the VMIN and VTIME subscripts may have the
6875 same values as the VEOF and VEOL subscripts, respectively.

6876 Implementations that do not support changing the START and STOP characters may ignore the
6877 character values in the `c_cc` array indexed by the VSTART and VSTOP subscripts when
6878 `tcsetattr()` is called, but shall return the value in use when `tgetattr()` is called.

6879 The initial values of all control characters are implementation-defined.

6880 If the value of one of the changeable special control characters (see [Section 11.1.9](#), on page 189) is
6881 `_POSIX_VDISABLE`, that function shall be disabled; that is, no input data is recognized as the
6882 disabled special character. If ICANON is not set, the value of `_POSIX_VDISABLE` has no special
6883 meaning for the VMIN and VTIME entries of the `c_cc` array.





12.1 Utility Argument Syntax

This section describes the argument syntax of the standard utilities and introduces terminology used throughout POSIX.1-200x for describing the arguments processed by the utilities.

Within POSIX.1-200x, a special notation is used for describing the syntax of a utility's arguments. Unless otherwise noted, all utility descriptions use this notation, which is illustrated by this example (see XCU Section 2.9.1, on page 2263):

```
utility_name[-a][[-b]][-c option_argument]
          [-d|-e][[-f[option_argument]]][operand...]
```

The notation used for the SYNOPSIS sections imposes requirements on the implementors of the standard utilities and provides a simple reference for the application developer or system user.

1. The utility in the example is named *utility_name*. It is followed by options, option-arguments, and operands. The arguments that consist of hyphens and single letters or digits, such as 'a', are known as "options" (or, historically, "flags"). Certain options are followed by an "option-argument", as shown with [-c *option_argument*]. The arguments following the last options and option-arguments are named "operands".
2. Option-arguments are shown separated from their options by <blank>s, except when the option-argument is enclosed in the '[' and ']' notation to indicate that it is optional. This reflects the situation in which an optional option-argument (if present) is included within the same argument string as the option; for a mandatory option-argument, it is the next argument. The Utility Syntax Guidelines in Section 12.2 (on page 201) require that the option be a separate argument from its option-argument and that option-arguments not be optional, but there are some exceptions in POSIX.1-200x to ensure continued operation of historical applications:
 - a. If the SYNOPSIS of a standard utility shows an option with a mandatory option-argument (as with [-c *option_argument*] in the example), a conforming application shall use separate arguments for that option and its option-argument. However, a conforming implementation shall also permit applications to specify the option and option-argument in the same argument string without intervening <blank>s.
 - b. If the SYNOPSIS shows an optional option-argument (as with [-f[*option_argument*]] in the example), a conforming application shall place any option-argument for that option directly adjacent to the option in the same argument string, without intervening <blank>s. If the utility receives an argument containing only the option, it shall behave as specified in its description for an omitted option-argument; it shall not treat the next argument (if any) as the option-argument for that option.
3. Options are usually listed in alphabetical order unless this would make the utility description more confusing. There are no implied relationships between the options based upon the order in which they appear, unless otherwise stated in the OPTIONS section, or unless the exception in Guideline 11 of Section 12.2 (on page 201) applies. If an

6926 option that does not have option-arguments is repeated, the results are undefined, unless
6927 otherwise stated.

6928 4. Frequently, names of parameters that require substitution by actual values are shown
6929 with embedded underscores. Alternatively, parameters are shown as follows:

6930 `<parameter name>`

6931 The angle brackets are used for the symbolic grouping of a phrase representing a single
6932 parameter and conforming applications shall not include them in data submitted to the
6933 utility.

6934 5. When a utility has only a few permissible options, they are sometimes shown
6935 individually, as in the example. Utilities with many flags generally show all of the
6936 individual flags (that do not take option-arguments) grouped, as in:

6937 `utility_name [-abcDxyz][-p arg][operand]`

6938 Utilities with very complex arguments may be shown as follows:

6939 `utility_name [options][operands]`

6940 6. Unless otherwise specified, whenever an operand or option-argument is, or contains, a
6941 numeric value:

- 6942 • The number is interpreted as a decimal integer.
- 6943 • Numerals in the range 0 to 2 147 483 647 are syntactically recognized as numeric
6944 values.
- 6945 • When the utility description states that it accepts negative numbers as operands or
6946 option-arguments, numerals in the range -2 147 483 647 to 2 147 483 647 are
6947 syntactically recognized as numeric values.
- 6948 • Ranges greater than those listed here are allowed.

6949 This does not mean that all numbers within the allowable range are necessarily
6950 semantically correct. A standard utility that accepts an option-argument or operand that
6951 is to be interpreted as a number, and for which a range of values smaller than that shown
6952 above is permitted by the POSIX.1-200x, describes that smaller range along with the
6953 description of the option-argument or operand. If an error is generated, the utility's
6954 diagnostic message shall indicate that the value is out of the supported range, not that it
6955 is syntactically incorrect.

6956 7. Arguments or option-arguments enclosed in the '[' and ']' notation are optional and
6957 can be omitted. Conforming applications shall not include the '[' and ']' symbols in
6958 data submitted to the utility.

6959 8. Arguments separated by the '|' vertical bar notation are mutually-exclusive.
6960 Conforming applications shall not include the '|' symbol in data submitted to the utility.
6961 Alternatively, mutually-exclusive options and operands may be listed with multiple
6962 synopsis lines.

6963
6964
6965
6966
6967
6968
6969
6970
6971
6972
6973
6974
6975
6976
6977

For example:

```
utility_name -d[-a][-c option_argument][operand...]  
utility_name[-a][-b][operand...]
```

When multiple synopsis lines are given for a utility, it is an indication that the utility has mutually-exclusive arguments. These mutually-exclusive arguments alter the functionality of the utility so that only certain other arguments are valid in combination with one of the mutually-exclusive arguments. Only one of the mutually-exclusive arguments is allowed for invocation of the utility. Unless otherwise stated in an accompanying OPTIONS section, the relationships between arguments depicted in the SYNOPSIS sections are mandatory requirements placed on conforming applications. The use of conflicting mutually-exclusive arguments produces undefined results, unless a utility description specifies otherwise. When an option is shown without the '[' and ']' brackets, it means that option is required for that version of the SYNOPSIS. However, it is not required to be the first argument, as shown in the example above, unless otherwise stated.

6978
6979
6980

9. Ellipses ("...") are used to denote that one or more occurrences of an operand are allowed. When an option or an operand followed by ellipses is enclosed in brackets, zero or more options or operands can be specified. The form:

6981
6982
6983
6984

```
utility_name [-g option_argument]...[operand...]
```

indicates that multiple occurrences of the option and its option-argument preceding the ellipses are valid, with semantics as indicated in the OPTIONS section of the utility. (See also Guideline 11 in [Section 12.2](#).)

6985

The form:

6986

```
utility_name -f option_argument [-f option_argument]... [operand...]
```

6987
6988

indicates that the `-f` option is required to appear at least once and may appear multiple times.

6989
6990
6991

10. When the synopsis line is too long to be printed on a single line in the Shell and Utilities volume of POSIX.1-200x, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

6992

12.2 Utility Syntax Guidelines

6993
6994
6995
6996

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. The `getopt()` function in the System Interfaces volume of POSIX.1-200x assists utilities in handling options and operands that conform to these guidelines.

6997
6998

Operands and option-arguments can contain characters not specified in the portable character set.

6999
7000
7001
7002

The guidelines are intended to provide guidance to the authors of future utilities, such as those written specific to a local system or that are components of a larger application. Some of the standard utilities do not conform to all of these guidelines; in those cases, the OPTIONS sections describe the deviations.

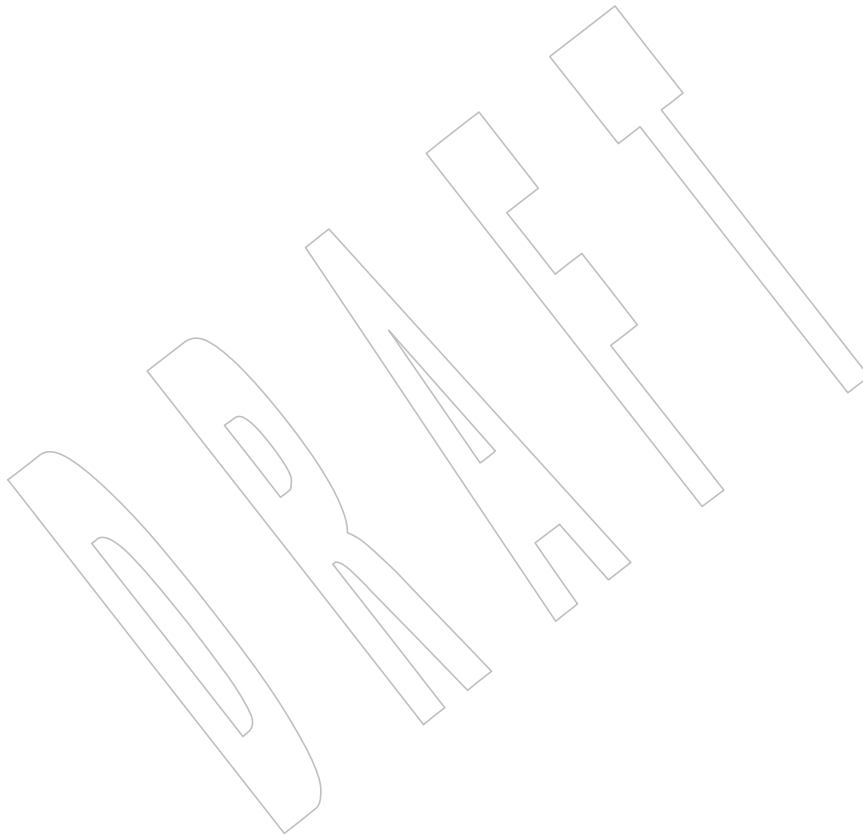
- 7003 **Guideline 1:** Utility names should be between two and nine characters, inclusive.
- 7004 **Guideline 2:** Utility names should include lowercase letters (the **lower** character
7005 classification) and digits only from the portable character set.
- 7006 **Guideline 3:** Each option name should be a single alphanumeric character (the **alnum**
7007 character classification) from the portable character set. The **-W** (capital-W)
7008 option shall be reserved for vendor options.
- 7009 Multi-digit options should not be allowed.
- 7010 **Guideline 4:** All options should be preceded by the **'-'** delimiter character.
- 7011 **Guideline 5:** Options without option-arguments should be accepted when grouped behind
7012 one **'-'** delimiter.
- 7013 **Guideline 6:** Each option and option-argument should be a separate argument, except as
7014 noted in [Section 12.1](#) (on page 199), item (2).
- 7015 **Guideline 7:** Option-arguments should not be optional.
- 7016 **Guideline 8:** When multiple option-arguments are specified to follow a single option, they
7017 should be presented as a single argument, using commas within that
7018 argument or <blank>s within that argument to separate them.
- 7019 **Guideline 9:** All options should precede operands on the command line.
- 7020 **Guideline 10:** The first **--** argument that is not an option-argument should be accepted as a
7021 delimiter indicating the end of options. Any following arguments should be
7022 treated as operands, even if they begin with the **'-'** character.
- 7023 **Guideline 11:** The order of different options relative to one another should not matter, unless
7024 the options are documented as mutually-exclusive and such an option is
7025 documented to override any incompatible options preceding it. If an option
7026 that has option-arguments is repeated, the option and option-argument
7027 combinations should be interpreted in the order specified on the command
7028 line.
- 7029 **Guideline 12:** The order of operands may matter and position-related interpretations should
7030 be determined on a utility-specific basis.
- 7031 **Guideline 13:** For utilities that use operands to represent files to be opened for either reading
7032 or writing, the **'-'** operand should be used to mean only standard input (or
7033 standard output when it is clear from context that an output file is being
7034 specified) or a file named **'-'**.
- 7035 **Guideline 14:** If an argument can be identified according to Guidelines 3 through 10 as an
7036 option, or as a group of options without option-arguments behind one **'-'**
7037 delimiter, then it should be treated as such.

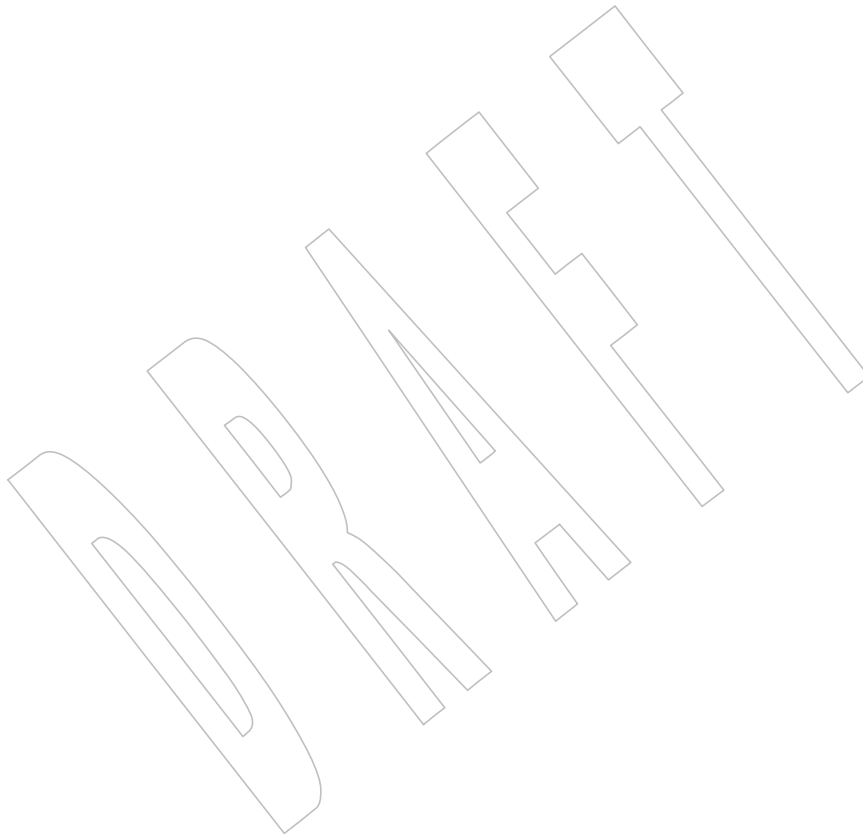
7038 The utilities in the Shell and Utilities volume of POSIX.1-200x that claim conformance to these
7039 guidelines shall conform completely to these guidelines as if these guidelines contained the term
7040 "shall" instead of "should". On some implementations, the utilities accept usage in violation of
7041 these guidelines for backwards-compatibility as well as accepting the required form.

7042 Where a utility described in the Shell and Utilities volume of POSIX.1-200x as conforming to
7043 these guidelines is required to accept the operand **'-'** to mean standard input or output, this
7044 usage is explained in the OPERANDS section. Otherwise, if such a utility uses operands to
7045 represent files, it is implementation-defined whether the operand **'-'** stands for standard input
7046 (or standard output), or for a file named **'-'**.

7047
7048
7049

It is recommended that all future utilities and applications use these guidelines to enhance user portability. The fact that some historical utilities could not be changed (to avoid breaking existing applications) should not deter this future goal.





This chapter describes the contents of headers.

Headers contain function prototypes, the definition of symbolic constants, common structures, preprocessor macros, and defined types. Each function in the System Interfaces volume of POSIX.1-200x specifies the headers that an application shall include in order to use that function. In most cases, only one header is required. These headers are present on an application development system; they need not be present on the target execution system.

Format of Entries

The entries in this chapter are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS and DESCRIPTION.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described.

DESCRIPTION

This section describes the functionality of the header.

APPLICATION USAGE

This section is informative. This section gives warnings and advice to application developers about the entry. In the event of conflict between warnings and advice and a normative part of this volume of POSIX.1-200x, the normative material is to be taken as correct.

RATIONALE

This section is informative. This section contains historical information concerning the contents of this volume of POSIX.1-200x and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative. This section provides comments which should be used as a guide to current thinking; there is not necessarily a commitment to adopt these future directions.

SEE ALSO

This section is informative. This section gives references to related information.

CHANGE HISTORY

This section is informative. This section shows the derivation of the entry and any significant changes that have been made to it.

7085 **NAME**7086 aio.h — asynchronous input and output (**REALTIME**)7087 **SYNOPSIS**

7088 #include <aio.h>

7089 **DESCRIPTION**7090 The <aio.h> header shall define the **aio_cb** structure which shall include at least the following
7091 members:

7092	int	aio_fildes	File descriptor.
7093	off_t	aio_offset	File offset.
7094	volatile void	*aio_buf	Location of buffer.
7095	size_t	aio_nbytes	Length of transfer.
7096	int	aio_reqprio	Request priority offset.
7097	struct sigevent	aio_sigevent	Signal number and value.
7098	int	aio_lio_opcode	Operation to be performed.

7099 The <aio.h> header shall define the following symbolic constants:

7100 **AIO_ALLDONE** A return value indicating that none of the requested operations could be
7101 canceled since they are already complete.7102 **AIO_CANCELED** A return value indicating that all requested operations have been
7103 canceled.7104 **AIO_NOTCANCELED**7105 A return value indicating that some of the requested operations could not
7106 be canceled since they are in progress.7107 **LIO_NOP** A *lio_listio()* element operation option indicating that no transfer is
7108 requested.7109 **LIO_NOWAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7110 is to continue execution while the *lio_listio()* operation is being
7111 performed, and no notification is given when the operation is complete.7112 **LIO_READ** A *lio_listio()* element operation option requesting a read.7113 **LIO_WAIT** A *lio_listio()* synchronization operation indicating that the calling thread
7114 is to suspend until the *lio_listio()* operation is complete.7115 **LIO_WRITE** A *lio_listio()* element operation option requesting a write.7116 The following shall be declared as functions and may also be defined as macros. Function
7117 prototypes shall be provided.

```

7118 int      aio_cancel(int, struct aio_cb *);
7119 int      aio_error(const struct aio_cb *);
7120 int      aio_fsync(int, struct aio_cb *);
7121 int      aio_read(struct aio_cb *);
7122 ssize_t  aio_return(struct aio_cb *);
7123 int      aio_suspend(const struct aio_cb *const [], int,
7124                    const struct timespec *);
7125 int      aio_write(struct aio_cb *);
7126 int      lio_listio(int, struct aio_cb *restrict const [restrict], int,
7127                    struct sigevent *restrict);

```

7128 Inclusion of the <aio.h> header may make visible symbols defined in the headers <fcntl.h>,

7129 <signal.h>, <sys/types.h>, and <time.h>.

7130 **APPLICATION USAGE**

7131 None.

7132 **RATIONALE**

7133 None.

7134 **FUTURE DIRECTIONS**

7135 None.

7136 **SEE ALSO**

7137 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

7138 XSH *fsync()*, *lseek()*, *read*, *write* |

7139 **CHANGE HISTORY**

7140 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

7141 **Issue 6**

7142 The <aio.h> header is marked as part of the Asynchronous Input and Output option.

7143 The description of the constants is expanded.

7144 The **restrict** keyword is added to the prototype for *lio_listio()*.

7145 **Issue 7**

7146 The <aio.h> header is moved from the Asynchronous Input and Output option to the Base.

7147 This reference page is clarified with respect to macros and symbolic constants. +

DRAFT

7148 **NAME**

7149 arpa/inet.h — definitions for internet operations

7150 **SYNOPSIS**

7151 #include <arpa/inet.h>

7152 **DESCRIPTION**7153 The **in_port_t** and **in_addr_t** types shall be defined as described in <netinet/in.h>.7154 The **in_addr** structure shall be defined as described in <netinet/in.h>.7155 IP6 The **INET_ADDRSTRLEN** and **INET6_ADDRSTRLEN** macros shall be defined as described in
7156 <netinet/in.h>.7157 The following shall be declared as functions, or defined as macros, or both. If functions are
7158 declared, function prototypes shall be provided.

7159 uint32_t htonl(uint32_t);

7160 uint16_t htons(uint16_t);

7161 uint32_t ntohl(uint32_t);

7162 uint16_t ntohs(uint16_t);

7163 The **uint32_t** and **uint16_t** types shall be defined as described in <inttypes.h>.7164 The following shall be declared as functions and may also be defined as macros. Function
7165 prototypes shall be provided.

7166 in_addr_t inet_addr(const char *);

7167 char *inet_ntoa(struct in_addr);

7168 const char *inet_ntop(int, const void *restrict, char *restrict,
7169 socklen_t);

7170 int inet_pton(int, const char *restrict, void *restrict);

7171 Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h>
7172 and <inttypes.h>.7173 **APPLICATION USAGE**

7174 None.

7175 **RATIONALE**

7176 None.

7177 **FUTURE DIRECTIONS**

7178 None.

7179 **SEE ALSO**

7180 <inttypes.h>, <netinet/in.h>

7181 XSH *htonl()*, *inet_addr()*7182 **CHANGE HISTORY**

7183 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

7184 The **restrict** keyword is added to the prototypes for *inet_ntop()* and *inet_pton()*.7185 **Issue 7**

7186 SD5-XBD-ERN-6 is applied.

7187 **NAME**
 7188 `assert.h` — verify program assertion

7189 **SYNOPSIS**
 7190 `#include <assert.h>`

7191 **DESCRIPTION**

7192 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7193 conflict between the requirements described here and the ISO C standard is unintentional. This
 7194 volume of POSIX.1-200x defers to the ISO C standard.

7195 The **<assert.h>** header shall define the `assert()` macro. It refers to the macro `NDEBUG` which is
 7196 not defined in the header. If `NDEBUG` is defined as a macro name before the inclusion of this
 7197 header, the `assert()` macro shall be defined simply as:

7198 `#define assert(ignore)((void) 0)`

7199 Otherwise, the macro behaves as described in `assert()`.

7200 The `assert()` macro shall be redefined according to the current state of `NDEBUG` each time
 7201 **<assert.h>** is included.

7202 The `assert()` macro shall be implemented as a macro, not as a function. If the macro definition is
 7203 suppressed in order to access an actual function, the behavior is undefined.

7204 **APPLICATION USAGE**

7205 None.

7206 **RATIONALE**

7207 None.

7208 **FUTURE DIRECTIONS**

7209 None.

7210 **SEE ALSO**

7211 XSH `assert()`

7212 **CHANGE HISTORY**

7213 First released in Issue 1. Derived from Issue 1 of the SVID.

7214 **Issue 6**

7215 The definition of the `assert()` macro is changed for alignment with the ISO/IEC 9899:1999
 7216 standard.

7217 **NAME**

7218 complex.h — complex arithmetic

7219 **SYNOPSIS**

7220 #include <complex.h>

7221 **DESCRIPTION**

7222 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7223 conflict between the requirements described here and the ISO C standard is unintentional. This
 7224 volume of POSIX.1-200x defers to the ISO C standard.

7225 The <complex.h> header shall define the following macros:

7226 complex Expands to **_Complex**.7227 **_Complex_I** Expands to a constant expression of type **const float _Complex**, with the value
 7228 of the imaginary unit (that is, a number i such that $i^2=-1$).7229 imaginary Expands to **_Imaginary**.7230 **_Imaginary_I** Expands to a constant expression of type **const float _Imaginary** with the
 7231 value of the imaginary unit.7232 **I** Expands to either **_Imaginary_I** or **_Complex_I**. If **_Imaginary_I** is not defined,
 7233 **I** expands to **_Complex_I**.7234 The macros **imaginary** and **_Imaginary_I** shall be defined if and only if the implementation
 7235 supports imaginary types.7236 An application may undefine and then, perhaps, redefine the **complex**, **imaginary**, and **I** macros.7237 The following shall be declared as functions and may also be defined as macros. Function
 7238 prototypes shall be provided.

```

7239 double      cabs(double complex);
7240 float      cabsf(float complex);
7241 long double cabsl(long double complex);
7242 double complex  cacos(double complex);
7243 float complex  cacosf(float complex);
7244 double complex  cacosh(double complex);
7245 float complex  cacoshf(float complex);
7246 long double complex  cacoshl(long double complex);
7247 long double complex  cacosl(long double complex);
7248 double      carg(double complex);
7249 float      cargf(float complex);
7250 long double cargl(long double complex);
7251 double complex  casin(double complex);
7252 float complex  casinf(float complex);
7253 double complex  casinh(double complex);
7254 float complex  casinhf(float complex);
7255 long double complex  casinhl(long double complex);
7256 long double complex  casinl(long double complex);
7257 double complex  catan(double complex);
7258 float complex  catanf(float complex);
7259 double complex  catanh(double complex);
7260 float complex  catanhf(float complex);
7261 long double complex  catanhl(long double complex);
7262 long double complex  catanl(long double complex);

```

```

7263     double complex      ccos(double complex);
7264     float complex       ccosf(float complex);
7265     double complex      ccosh(double complex);
7266     float complex       ccoshf(float complex);
7267     long double complex ccoshl(long double complex);
7268     long double complex ccosl(long double complex);
7269     double complex      cexp(double complex);
7270     float complex       cexpf(float complex);
7271     long double complex cexpl(long double complex);
7272     double              cimag(double complex);
7273     float               cimagf(float complex);
7274     long double         cimagl(long double complex);
7275     double complex      clog(double complex);
7276     float complex       clogf(float complex);
7277     long double complex clogl(long double complex);
7278     double complex      conj(double complex);
7279     float complex       conjf(float complex);
7280     long double complex conjl(long double complex);
7281     double complex      cpow(double complex, double complex);
7282     float complex       cpowf(float complex, float complex);
7283     long double complex cpowl(long double complex, long double complex);
7284     double complex      cproj(double complex);
7285     float complex       cprojf(float complex);
7286     long double complex cprojl(long double complex);
7287     double              creal(double complex);
7288     float               crealf(float complex);
7289     long double         creall(long double complex);
7290     double complex      csin(double complex);
7291     float complex       csinf(float complex);
7292     double complex      csinh(double complex);
7293     float complex       csinhf(float complex);
7294     long double complex csinhl(long double complex);
7295     long double complex csinl(long double complex);
7296     double complex      csqrt(double complex);
7297     float complex       csqrtf(float complex);
7298     long double complex csqrtl(long double complex);
7299     double complex      ctan(double complex);
7300     float complex       ctanf(float complex);
7301     double complex      ctanh(double complex);
7302     float complex       ctanhf(float complex);
7303     long double complex ctanhl(long double complex);
7304     long double complex ctanl(long double complex);

```

APPLICATION USAGE

Values are interpreted as radians, not degrees.

RATIONALE

The choice of *I* instead of *i* for the imaginary unit concedes to the widespread use of the identifier *i* for other purposes. The application can use a different identifier, say *j*, for the imaginary unit by following the inclusion of the <complex.h> header with:

```

7311     #undef I
7312     #define j _Imaginary_I

```

An *I* suffix to designate imaginary constants is not required, as multiplication by *I* provides a sufficiently convenient and more generally useful notation for imaginary terms. The

7315 corresponding real type for the imaginary unit is **float**, so that use of *I* for algorithmic or
7316 notational convenience will not result in widening types.

7317 On systems with imaginary types, the application has the ability to control whether use of the
7318 macro `I` introduces an imaginary type, by explicitly defining `I` to be `_Imaginary_I` or `_Complex_I`.
7319 Disallowing imaginary types is useful for some applications intended to run on
7320 implementations without support for such types.

7321 The macro `_Imaginary_I` provides a test for whether imaginary types are supported.

7322 The `cis()` function ($\cos(x) + I\sin(x)$) was considered but rejected because its implementation is
7323 easy and straightforward, even though some implementations could compute sine and cosine
7324 more efficiently in tandem.

7325 FUTURE DIRECTIONS

7326 The following function names and the same names suffixed with *f* or *l* are reserved for future
7327 use, and may be added to the declarations in the <complex.h> header.

7328 `cerf()` `cexpm1()` `clog2()`
7329 `cerfc()` `clog10()` `clgamma()`
7330 `cexp2()` `clog1p()` `ctgamma()`

7331 SEE ALSO

7332 XSH `cabs()`, `cacos()`, `cacosh()`, `carg()`, `casin()`, `casinh()`, `catan()`, `catanh()`, `ccos()`, `ccosh()`, `cexp()`,
7333 `cimag()`, `clog()`, `conj()`, `cpow()`, `cproj()`, `creal()`, `csin()`, `csinh()`, `csqrt()`, `ctan()`, `ctanh()`

7334 CHANGE HISTORY

7335 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7336 **NAME**
7337 cpio.h — cpio archive values

7338 **SYNOPSIS**
7339 #include <cpio.h>

7340 **DESCRIPTION**
7341 The <cpio.h> header shall define the symbolic constants needed by the *c_mode* field of the *cpio* |
7342 archive format, with the names and values given in the following table:

Name	Description	Value (Octal)
C_IRUSR	Read by owner.	0000400
C_IWUSR	Write by owner.	0000200
C_IXUSR	Execute by owner.	0000100
C_IRGRP	Read by group.	0000040
C_IWGRP	Write by group.	0000020
C_IXGRP	Execute by group.	0000010
C_IROTH	Read by others.	0000004
C_IWOTH	Write by others.	0000002
C_IXOTH	Execute by others.	0000001
C_ISUID	Set user ID.	0004000
C_ISGID	Set group ID.	0002000
C_ISVTX	On directories, restricted deletion flag.	0001000
C_ISDIR	Directory.	0040000
C_ISFIFO	FIFO.	0010000
C_ISREG	Regular file.	0100000
C_ISBLK	Block special.	0060000
C_ISCHR	Character special.	0020000
C_ISCTG	Reserved.	0110000
C_ISLNK	Symbolic link.	0120000
C_ISSOCK	Socket.	0140000

7364 The <cpio.h> header shall define the following symbolic constant as a string: |

7365 MAGIC "070707"

7366 **APPLICATION USAGE**
7367 None.

7368 **RATIONALE**
7369 None.

7370 **FUTURE DIRECTIONS**
7371 None.

7372 **SEE ALSO**
7373 XCU *pax* |

7374 **CHANGE HISTORY**
7375 First released in the Headers Interface, Issue 3 specification. Derived from the POSIX.1-1988
7376 standard.

7377 **Issue 6**
7378 The SEE ALSO is updated to refer to *pax*.

7379

Issue 7

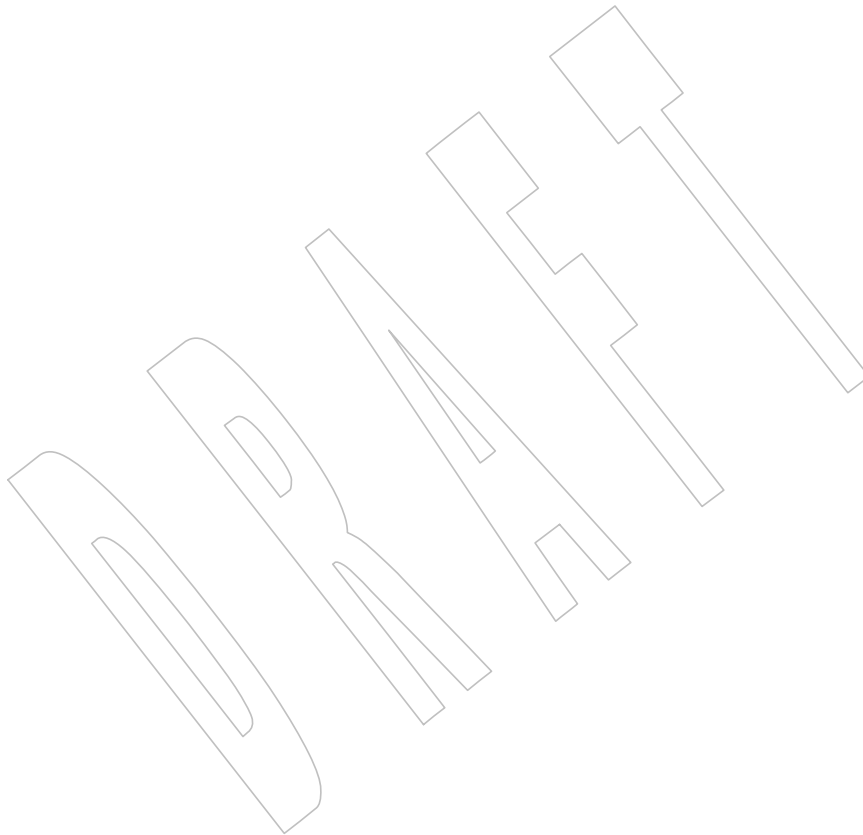
7380

The **<pio.h>** header is moved from the XSI option to the Base.

7381

This reference page is clarified with respect to macros and symbolic constants.

+



7382 **NAME**7383 `ctype.h` — character types7384 **SYNOPSIS**7385 `#include <ctype.h>`7386 **DESCRIPTION**

7387 CX Some of the functionality described on this reference page extends the ISO C standard.
 7388 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 448) to
 7389 enable the visibility of these symbols in this header.

7390 The <ctype.h> header shall provide a definition for a type `locale_t` as defined in <locale.h>
 7391 representing a locale object.

7392 The following shall be declared as functions and may also be defined as macros. Function
 7393 prototypes shall be provided for use with ISO C standard compilers.

```

7394 int isalnum(int);
7395 CX int isalnum_l(int, locale_t);
7396 int isalpha(int);
7397 CX int isalpha_l(int, locale_t);
7398 OB XSI int isascii(int);
7399 int isblank(int);
7400 CX int isblank_l(int, locale_t);
7401 int iscntrl(int);
7402 CX int iscntrl_l(int, locale_t);
7403 int isdigit(int);
7404 CX int isdigit_l(int, locale_t);
7405 int isgraph(int);
7406 CX int isgraph_l(int, locale_t);
7407 int islower(int);
7408 CX int islower_l(int, locale_t);
7409 int isprint(int);
7410 CX int isprint_l(int, locale_t);
7411 int ispunct(int);
7412 CX int ispunct_l(int, locale_t);
7413 int isspace(int);
7414 CX int isspace_l(int, locale_t);
7415 int isupper(int);
7416 CX int isupper_l(int, locale_t);
7417 int isxdigit(int);
7418 CX int isxdigit_l(int, locale_t);
7419 OB XSI int toascii(int);
7420 int tolower(int);
7421 CX int tolower_l(int, locale_t);
7422 int toupper(int);
7423 CX int toupper_l(int, locale_t);

```

7424 The <ctype.h> header shall define the following as macros:

```

7425 OB XSI int _toupper(int);
7426 int _tolower(int);

```

7427
7428
7429
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<locale.h>

XSH Section 2.2 (on page 448), *isalnum()*, *isalpha()*, *isascii()*, *isctrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *setlocale()*, *toascii()*, *tolower()*, *_tolower()*, *toupper()*, *_toupper()*, *wcstombs()*, *wctomb()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

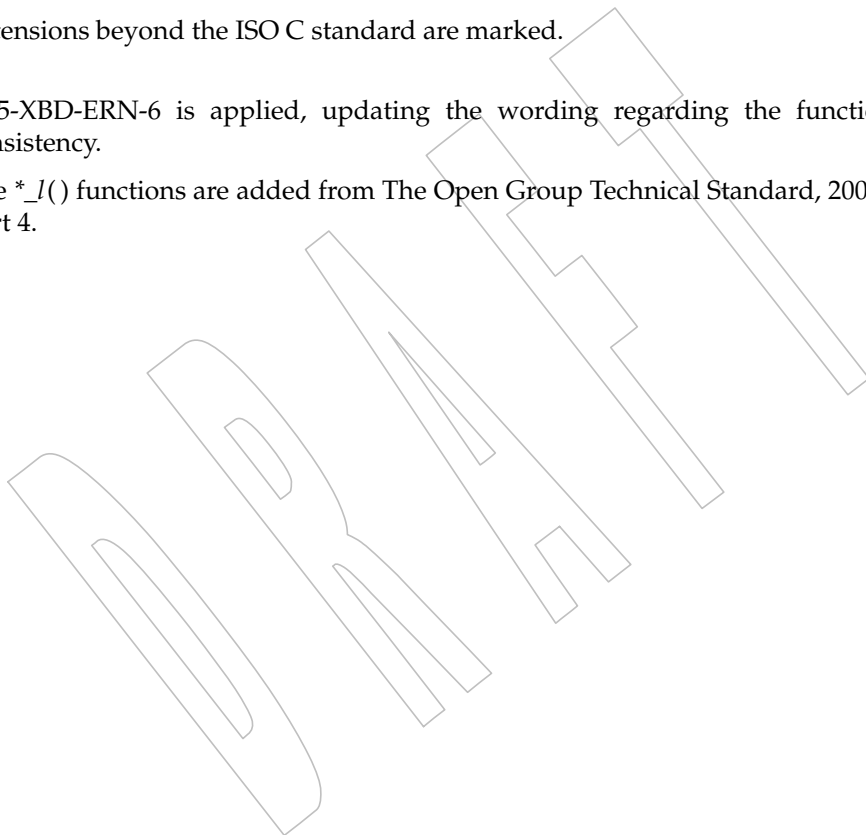
Issue 6

Extensions beyond the ISO C standard are marked.

Issue 7

SD5-XBD-ERN-6 is applied, updating the wording regarding the function declarations for consistency.

The *_l() functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



7447 **NAME**

7448 dirent.h — format of directory entries

7449 **SYNOPSIS**

7450 #include <dirent.h>

7451 **DESCRIPTION**

7452 The internal format of directories is unspecified.

7453 The <dirent.h> header shall define the following type:

7454 **DIR** A type representing a directory stream. The **DIR** type may be an incomplete type. |7455 It shall also define the structure **dirent** which shall include the following members:7456 XSI `ino_t d_ino` File serial number.7457 `char d_name[]` Name of entry.7458 XSI The type `ino_t` shall be defined as described in <sys/types.h>.7459 The character array `d_name` is of unspecified size, but the number of bytes preceding the
7460 terminating null byte shall not exceed {NAME_MAX}.7461 The following shall be declared as functions and may also be defined as macros. Function
7462 prototypes shall be provided.7463 `int alphasort(const struct dirent **, const struct dirent **);`7464 `int closedir(DIR *);`7465 `int dirfd(DIR *);`7466 `DIR *fdopendir(int);`7467 `DIR *opendir(const char *);`7468 `struct dirent *readdir(DIR *);`7469 `int readdir_r(DIR *restrict, struct dirent *restrict,`
7470 `struct dirent **restrict);`7471 `void rewinddir(DIR *);`7472 `int scandir(const char *, struct dirent ***,`7473 `int (*)(const struct dirent *),`7474 `int (*)(const struct dirent **),`7475 `const struct dirent **);`7476 XSI `void seekdir(DIR *, long);`7477 `long telldir(DIR *);`7478 **APPLICATION USAGE**

7479 None.

7480 **RATIONALE**7481 Information similar to that in the <dirent.h> header is contained in a file <sys/dir.h> in 4.2 BSD
7482 and 4.3 BSD. The equivalent in these implementations of **struct dirent** from this volume of
7483 POSIX.1-200x is **struct direct**. The filename was changed because the name <sys/dir.h> was also
7484 used in earlier implementations to refer to definitions related to the older access method; this
7485 produced name conflicts. The name of the structure was changed because this volume of
7486 POSIX.1-200x does not completely define what is in the structure, so it could be different on
7487 some implementations from **struct direct**.7488 The name of an array of **char** of an unspecified size should not be used as an lvalue. Use of:7489 `sizeof(d_name)`

7490 is incorrect; use:
 7491 `strlen(d_name)`
 7492 instead.

7493 The array of `char d_name` is not a fixed size. Implementations may need to declare `struct dirent`
 7494 with an array size for `d_name` of 1, but the actual number of characters provided matches (or
 7495 only slightly exceeds) the length of the filename.

7496 FUTURE DIRECTIONS

7497 None.

7498 SEE ALSO

7499 [<sys/types.h>](#)

7500 XSH [closedir\(\)](#), [fdopendir\(\)](#), [readdir\(\)](#), [rewinddir\(\)](#), [seekdir\(\)](#), [telldir\(\)](#) |

7501 CHANGE HISTORY

7502 First released in Issue 2.

7503 Issue 5

7504 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

7505 Issue 6

7506 The Open Group Corrigendum U026/7 is applied, correcting the prototype for `readdir_r()`.

7507 The `restrict` keyword is added to the prototype for `readdir_r()`.

7508 Issue 7

7509 The `alphasort()`, `dirfd()`, and `scandir()` functions are added from The Open Group Technical
 7510 Standard, 2006, Extended API Set Part 1.

7511 The `fopendir()` function is added from The Open Group Technical Standard, 2006, Extended API
 7512 Set Part 2.

7513 Austin Group Interpretation 1003.1-2001 #110 is applied, clarifying the definition of the `DIR` +
 7514 type.

7515 **NAME**7516 `dlfcn.h` — dynamic linking7517 **SYNOPSIS**7518 `#include <dlfcn.h>`7519 **DESCRIPTION**7520 The `<dlfcn.h>` header shall define at least the following symbolic constants for use in the
7521 construction of a `dlopen()` *mode* argument:7522 `RTLD_LAZY` Relocations are performed at an implementation-defined time.7523 `RTLD_NOW` Relocations are performed when the object is loaded.7524 `RTLD_GLOBAL` All symbols are available for relocation processing of other modules.7525 `RTLD_LOCAL` All symbols are not made available for relocation processing by other
7526 modules.7527 The following shall be declared as functions and may also be defined as macros. Function
7528 prototypes shall be provided.7529 `int dlclose(void *);`
7530 `char *dlerror(void);`
7531 `void *dlopen(const char *, int);`
7532 `void *dlsym(void *restrict, const char *restrict);`7533 **APPLICATION USAGE**

7534 None.

7535 **RATIONALE**

7536 None.

7537 **FUTURE DIRECTIONS**

7538 None.

7539 **SEE ALSO**7540 XSH [dlopen\(\)](#), [dlclose\(\)](#), [dlsym\(\)](#), [dlerror\(\)](#)7541 **CHANGE HISTORY**

7542 First released in Issue 5.

7543 **Issue 6**7544 The `restrict` keyword is added to the prototype for `dlsym()`.7545 **Issue 7**7546 The `<dlfcn.h>` header is moved from the XSI option to the Base.

7547 This reference page is clarified with respect to macros and symbolic constants. +

7548 **NAME**7549 `errno.h` — system error numbers7550 **SYNOPSIS**7551 `#include <errno.h>`7552 **DESCRIPTION**7553 **CX** Some of the functionality described on this reference page extends the ISO C standard. Any
7554 conflict between the requirements described here and the ISO C standard is unintentional. This
7555 volume of POSIX.1-200x defers to the ISO C standard.

7556 The ISO C standard only requires the symbols [EDOM], [EILSEQ], and [ERANGE] to be defined.

7557 The <errno.h> header shall provide a declaration for *errno*. The symbol *errno* shall expand to a +
7558 modifiable lvalue of type **int**. It is unspecified whether *errno* is a macro or an identifier declared |
7559 with external linkage. If a macro definition is suppressed in order to access an actual object, or a |
7560 program defines an identifier with the name *errno*, the behavior is undefined. |7561 The <errno.h> header shall define the following macros which shall expand to integer constant |
7562 expressions with type **int**, distinct positive values (except as noted below), and which shall be |
7563 suitable for use in **#if** preprocessing directives: |

7564 [E2BIG] Argument list too long.

7565 [EACCES] Permission denied.

7566 [EADDRINUSE] Address in use.

7567 [EADDRNOTAVAIL] Address not available.

7568 [EAFNOSUPPORT] Address family not supported.

7569 [EAGAIN] Resource unavailable, try again (may be the same value as |
7570 [EWOULDBLOCK]).

7571 [EALREADY] Connection already in progress.

7572 [EBADF] Bad file descriptor.

7573 [EBADMSG] Bad message.

7574 [EBUSY] Device or resource busy.

7575 [ECANCELED] Operation canceled.

7576 [ECHILD] No child processes.

7577 [ECONNABORTED] Connection aborted.

7578 [ECONNREFUSED] Connection refused.

7579 [ECONNRESET] Connection reset.

7580 [EDEADLK] Resource deadlock would occur.

7581 [EDESTADDRREQ] Destination address required.

7582 [EDOM] Mathematics argument out of domain of function.

7583 [EDQUOT] Reserved.

7584		[EEXIST]	File exists.
7585		[EFAULT]	Bad address.
7586		[EFBIG]	File too large.
7587		[EHOSTUNREACH]	Host is unreachable.
7588		[EIDRM]	Identifier removed.
7589		[EILSEQ]	Illegal byte sequence.
7590		[EINPROGRESS]	Operation in progress.
7591		[EINTR]	Interrupted function.
7592		[EINVAL]	Invalid argument.
7593		[EIO]	I/O error.
7594		[EISCONN]	Socket is connected.
7595		[EISDIR]	Is a directory.
7596		[ELOOP]	Too many levels of symbolic links.
7597		[EMFILE]	File descriptor value too large.
7598		[EMLINK]	Too many links.
7599		[EMSGSIZE]	Message too large.
7600		[EMULTIHOP]	Reserved.
7601		[ENAMETOOLONG]	Filename too long.
7602		[ENETDOWN]	Network is down.
7603		[ENETRESET]	Connection aborted by network.
7604		[ENETUNREACH]	Network unreachable.
7605		[ENFILE]	Too many files open in system.
7606		[ENOBUFS]	No buffer space available.
7607	OB XSR	[ENODATA]	No message is available on the STREAM head read queue.
7608		[ENODEV]	No such device.
7609		[ENOENT]	No such file or directory.
7610		[ENOEXEC]	Executable file format error.
7611		[ENOLCK]	No locks available.
7612		[ENOLINK]	Reserved.
7613		[ENOMEM]	Not enough space.
7614		[ENOMSG]	No message of the desired type.
7615		[ENOPROTOPT]	Protocol not available.
7616		[ENOSPC]	No space left on device.
7617	OB XSR	[ENOSR]	No STREAM resources.
7618	OB XSR	[ENOSTR]	Not a STREAM.

7619	[ENOSYS]	Function not supported.
7620	[ENOTCONN]	The socket is not connected.
7621	[ENOTDIR]	Not a directory.
7622	[ENOTEMPTY]	Directory not empty.
7623	[ENOTRECOVERABLE]	
7624		State not recoverable.
7625	[ENOTSOCK]	Not a socket.
7626	[ENOTSUP]	Not supported (may be the same value as [EOPNOTSUPP]).
7627	[ENOTTY]	Inappropriate I/O control operation.
7628	[ENXIO]	No such device or address.
7629	[EOPNOTSUPP]	Operation not supported on socket (may be the same value as
7630		[ENOTSUP]).
7631	[EOVERFLOW]	Value too large to be stored in data type.
7632	[EOWNERDEAD]	Previous owner died.
7633	[EPERM]	Operation not permitted.
7634	[EPIPE]	Broken pipe.
7635	[EPROTO]	Protocol error.
7636	[EPROTONOSUPPORT]	
7637		Protocol not supported.
7638	[EPROTOTYPE]	Protocol wrong type for socket.
7639	[ERANGE]	Result too large.
7640	[EROFS]	Read-only file system.
7641	[ESPIPE]	Invalid seek.
7642	[ESRCH]	No such process.
7643	[ESTALE]	Reserved.
7644	OB XSR [ETIME]	Stream <i>ioctl()</i> timeout.
7645	[ETIMEDOUT]	Connection timed out.
7646	[ETXTBSY]	Text file busy.
7647	[EWOULDBLOCK]	Operation would block (may be the same value as [EAGAIN]).
7648	[EXDEV]	Cross-device link.

APPLICATION USAGE

Additional error numbers may be defined on conforming systems; see the System Interfaces volume of POSIX.1-200x.

RATIONALE

None.

FUTURE DIRECTIONS

None.

7656

SEE ALSO

7657

XSH [Section 2.3](#) (on page 456) |

7658

CHANGE HISTORY

7659

First released in Issue 1. Derived from Issue 1 of the SVID.

7660

Issue 5

7661

Updated for alignment with the POSIX Realtime Extension.

7662

Issue 6

7663

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

7664

7665

- The majority of the error conditions previously marked as extensions are now mandatory, except for the STREAMS-related error conditions.

7666

7667

Values for *errno* are now required to be distinct positive values rather than non-zero values. This change is for alignment with the ISO/IEC 9899: 1999 standard.

7668

7669

Issue 7

7670

Austin Group Interpretation 1003.1-2001 #050 is applied.

7671

The [ENOTRECOVERABLE] and [EOWNERDEAD] errors are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

7672

7673

Functionality relating to the XSI STREAMS option is marked obsolescent.

7674

Functionality relating to the Threads option is moved to the Base.

7675

This reference page is clarified with respect to macros and symbolic constants. +

7676 NAME

7677 fcntl.h — file control options

7678 SYNOPSIS

7679 #include <fcntl.h>

7680 DESCRIPTION

7681 The <fcntl.h> header shall define the following symbolic constants for the *cmd* argument used
 7682 by *fcntl()*. The values shall be unique and shall be suitable for use in *#if* preprocessing
 7683 directives.

7684 F_DUPFD Duplicate file descriptor.

7685 F_GETFD Get file descriptor flags.

7686 F_SETFD Set file descriptor flags.

7687 F_GETFL Get file status flags and file access modes.

7688 F_SETFL Set file status flags.

7689 F_GETLK Get record locking information.

7690 F_SETLK Set record locking information.

7691 F_SETLKW Set record locking information; wait if blocked.

7692 F_GETOWN Get process or process group ID to receive SIGURG signals.

7693 F_SETOWN Set process or process group ID to receive SIGURG signals.

7694 The <fcntl.h> header shall define the following symbolic constant used for the *fcntl()* file
 7695 descriptor flags, which shall be suitable for use in *#if* preprocessing directives.

7696 FD_CLOEXEC Close the file descriptor upon execution of an *exec* family function.

7697 The <fcntl.h> header shall also define the following symbolic constants for the *l_type* argument
 7698 used for record locking with *fcntl()*. The values shall be unique and shall be suitable for use in
 7699 *#if* preprocessing directives.

7700 F_RDLCK Shared or read lock.

7701 F_UNLCK Unlock.

7702 F_WRLCK Exclusive or write lock.

7703 The <fcntl.h> header shall define the values used for *l_whence*, *SEEK_SET*, *SEEK_CUR*, and
 7704 *SEEK_END* as described in <unistd.h>.

7705 The <fcntl.h> header shall define the following symbolic constants as file creation flags for use
 7706 in the *oflag* value to *open()* and *openat()*. The values shall be bitwise-distinct and shall be
 7707 suitable for use in *#if* preprocessing directives.

7708 O_CREAT Create file if it does not exist.

7709 O_EXCL Exclusive use flag.

7710 O_NOCTTY Do not assign controlling terminal.

7711 O_TRUNC Truncate flag.

7712 The *fcntl.h()* header shall define the following symbolic constants for use as file status flags for
 7713 *open()*, *openat()*, and *fcntl()*. The values shall be suitable for use in *#if* preprocessing directives.

7714		O_APPEND	Set append mode.
7715	SIO	O_DSYNC	Write according to synchronized I/O data integrity completion.
7716		O_NONBLOCK	Non-blocking mode.
7717	SIO	O_RSYNC	Synchronized read I/O operations.
7718		O_SYNC	Write according to synchronized I/O file integrity completion.
7719		The <fcntl.h> header shall define the following symbolic constant for use as the mask for file access modes. The value shall be suitable for use in #if preprocessing directives.	
7720			
7721		O_ACCMODE	Mask for file access modes.
7722		The <fcntl.h> header shall define the following symbolic constants for use as the file access modes for <i>open()</i> , <i>openat()</i> , and <i>fcntl()</i> . The values shall be suitable for use in #if preprocessing directives.	
7723			
7724			
7725		O_EXEC	Open for execute only (non-directory files). Use of this flag on directories is currently unspecified.
7726			
7727		O_RDONLY	Open for reading only.
7728		O_RDWR	Open for reading and writing.
7729		O_WRONLY	Open for writing only.
7730		The <fcntl.h> header shall define the symbolic constants for file modes for use as values of mode_t as described in <sys/stat.h>.	
7731			
7732		The <fcntl.h> header shall define the following symbolic constant as a special value used in place of a file descriptor for the <i>*at()</i> functions which take a directory file descriptor as a parameter:	
7733			
7734			
7735		AT_FDCWD	Use the current working directory to determine the target of relative file paths.
7736			
7737		The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>faccessat()</i> :	
7738			
7739		AT_EACCESS	Check access using effective user and group ID.
7740		The <fcntl.h> header shall define the following symbolic constant as a value for the <i>flag</i> used by <i>fstatat()</i> , <i>fchmodat()</i> , <i>fchownat()</i> , and <i>utimensat()</i> :	
7741			
7742		AT_SYMLINK_NOFOLLOW	
7743			Do not follow symbolic links.
7744		The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by <i>linkat()</i> :	
7745			
7746		AT_SYMLINK_FOLLOW	
7747			Follow symbolic link.
7748		The <fcntl.h> header shall define the following symbolic constants as value for the flag used by <i>open()</i> and <i>openat()</i> :	
7749			
7750		O_DIRECTORY	Fail if not a directory.
7751		O_NOFOLLOW	Do not follow symbolic links.
7752		The <fcntl.h> header shall define the following symbolic constant as a value for the flag used by <i>unlinkat()</i> :	
7753			

7754 AT_REMOVEDIR Remove directory instead of file.

7755 ADV The <fcntl.h> header shall define the following symbolic constants for the *advice* argument used |
7756 by *posix_fadvise()*:

7757 POSIX_FADV_DONTNEED
7758 The application expects that it will not access the specified data in the near future. +

7759 POSIX_FADV_NOREUSE
7760 The application expects to access the specified data once and then not reuse it thereafter. +

7761 POSIX_FADV_NORMAL
7762 The application has no advice to give on its behavior with respect to the specified data. It is
7763 the default characteristic if no advice is given for an open file. -

7764 POSIX_FADV_RANDOM
7765 The application expects to access the specified data in a random order.

7766 POSIX_FADV_SEQUENTIAL
7767 The application expects to access the specified data sequentially from lower offsets to higher
7768 offsets. +

7769 POSIX_FADV_WILLNEED
7770 The application expects to access the specified data in the near future. -

7771 The <fcntl.h> header shall define the **flock** structure describing a file lock. It shall include the
7772 following members:

7773 short l_type Type of lock; F_RDLCK, F_WRLCK, F_UNLCK.
7774 short l_whence Flag for starting offset.
7775 off_t l_start Relative offset in bytes.
7776 off_t l_len Size; if 0 then until EOF.
7777 pid_t l_pid Process ID of the process holding the lock; returned with F_GETLK.

7778 The <fcntl.h> header shall define the **mode_t**, **off_t**, and **pid_t** types as described in
7779 <sys/types.h>.

7780 The following shall be declared as functions and may also be defined as macros. Function
7781 prototypes shall be provided.

7782 int creat(const char *, mode_t);
7783 int fcntl(int, int, ...);
7784 int open(const char *, int, ...);
7785 int openat(int, const char *, int, ...);
7786 ADV int posix_fadvise(int, off_t, off_t, int);
7787 int posix_fallocate(int, off_t, off_t);

7788 Inclusion of the <fcntl.h> header may also make visible all symbols from <sys/stat.h> and
7789 <unistd.h>.

7790
7791
7792
7793
7794
7795
7796
7797
7798
7799
7800
7801
7802
7803
7804
7805
7806
7807
7808
7809
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829

APPLICATION USAGE

Although no existing implementation defines `AT_SYMLINK_FOLLOW` and `AT_SYMLINK_NOFOLLOW` as the same numeric value, POSIX.1-200x does not prohibit that as the two constants are not used with the same interfaces.

RATIONALE

While many of the symbolic constants introduced in the `<fcntl.h>` header do not strictly need to be used in `#if` preprocessor directives, widespread historic practice has defined them as macros that are usable in such constructs, and examination of existing applications has shown that they are occasionally used in such a way. Therefore it was decided to retain this requirement on an implementation in POSIX.1-200x.

FUTURE DIRECTIONS

The meaning of the `O_EXEC` flag on directories may be specified in a future version.

SEE ALSO

[<sys/stat.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

XSH *creat()*, *exec*, *fcntl()*, *futimens()*, *open()*, *posix_fadvise()*, *posix_fallocate()*, *posix_madvise()*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- `O_DSYNC` and `O_RSYNC` are marked as part of the Synchronized Input and Output option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The definition of the `mode_t`, `off_t`, and `pid_t` types is mandated.

The `F_GETOWN` and `F_SETOWN` values are added for sockets.

The *posix_fadvise()*, *posix_fallocate()*, and *posix_madvise()* functions are added for alignment with IEEE Std 1003.1d-1999.

IEEE PASC Interpretation 1003.1 #102 is applied, moving the prototype for *posix_madvise()* to `<sys/mman.h>`.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/18 is applied, updating the prototypes for *posix_fadvise()* and *posix_fallocate()* to be large file-aware, using `off_t` instead of `size_t`.

Issue 7

The *openat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Additional flags are added to support *faccessat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*, *open()*, *openat()*, and *unlinkat()*.

This reference page is clarified with respect to macros and symbolic constants. +

Changes are made related to support for finegrained timestamps.

7830 NAME

7831 fenv.h — floating-point environment

7832 SYNOPSIS

7833 #include <fenv.h>

7834 DESCRIPTION

7835 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7836 conflict between the requirements described here and the ISO C standard is unintentional. This
 7837 volume of POSIX.1-200x defers to the ISO C standard.

7838 The <fenv.h> header shall define the following data types through **typedef**:

7839 **fenv_t** Represents the entire floating-point environment. The floating-point environment
 7840 refers collectively to any floating-point status flags and control modes supported
 7841 by the implementation.

7842 **feexcept_t** Represents the floating-point status flags collectively, including any status the
 7843 implementation associates with the flags. A floating-point status flag is a system
 7844 variable whose value is set (but never cleared) when a floating-point exception is
 7845 raised, which occurs as a side effect of exceptional floating-point arithmetic to
 7846 provide auxiliary information. A floating-point control mode is a system variable
 7847 whose value may be set by the user to affect the subsequent behavior of floating-
 7848 point arithmetic.

7849 The <fenv.h> header shall define each of the following macros if and only if the implementation
 7850 supports the floating-point exception by means of the floating-point functions *feclearexcept()*,
 7851 *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, and *fetestexcept()*. The defined macros shall
 7852 expand to integer constant expressions with values such that bitwise ORs of all combinations of
 7853 the macros result in distinct values.

7854 FE_DIVBYZERO
 7855 FE_INEXACT
 7856 FE_INVALID
 7857 FE_OVERFLOW
 7858 FE_UNDERFLOW

7859 MX If the implementation supports the IEC 60559 Floating-Point option, all five macros shall be
 7860 defined. Additional implementation-defined floating-point exceptions with macros beginning
 7861 with FE_ and an uppercase letter may also be specified by the implementation.

7862 The <fenv.h> header shall define the macro FE_ALL_EXCEPT as the bitwise-inclusive OR of all
 7863 floating-point exception macros defined by the implementation, if any. If no such macros are
 7864 defined, then the macro FE_ALL_EXCEPT shall be defined as zero.

7865 The <fenv.h> header shall define each of the following macros if and only if the implementation
 7866 supports getting and setting the represented rounding direction by means of the *fegetround()*
 7867 and *fesetround()* functions. The defined macros shall expand to integer constant expressions
 7868 whose values are distinct non-negative values.

7869 FE_DOWNWARD
 7870 FE_TONEAREST
 7871 FE_TOWARDZERO
 7872 FE_UPWARD

7873 MX If the implementation supports the IEC 60559 Floating-Point option, all four macros shall be
 7874 defined. Additional implementation-defined rounding directions with macros beginning with

7875 FE_ and an uppercase letter may also be specified by the implementation.

7876 The <fenv.h> header shall define the following macro, which represents the default floating-
7877 point environment (that is, the one installed at program startup) and has type pointer to const-
7878 qualified `fenv_t`. It can be used as an argument to the functions within the <fenv.h> header that
7879 manage the floating-point environment.

7880 `FE_DFL_ENV`

7881 The following shall be declared as functions and may also be defined as macros. Function
7882 prototypes shall be provided.

```
7883 int feclearexcept(int);
7884 int fegetenv(fenv_t *);
7885 int fegetexceptflag(fexcept_t *, int);
7886 int fegetround(void);
7887 int feholdexcept(fenv_t *);
7888 int feraiseexcept(int);
7889 int fesetenv(const fenv_t *);
7890 int fesetexceptflag(const fexcept_t *, int);
7891 int fesetround(int);
7892 int fetestexcept(int);
7893 int feupdateenv(const fenv_t *);
```

7894 The `FENV_ACCESS` pragma provides a means to inform the implementation when an
7895 application might access the floating-point environment to test floating-point status flags or run
7896 under non-default floating-point control modes. The pragma shall occur either outside external
7897 declarations or preceding all explicit declarations and statements inside a compound statement.
7898 When outside external declarations, the pragma takes effect from its occurrence until another
7899 `FENV_ACCESS` pragma is encountered, or until the end of the translation unit. When inside a
7900 compound statement, the pragma takes effect from its occurrence until another `FENV_ACCESS`
7901 pragma is encountered (including within a nested compound statement), or until the end of the
7902 compound statement; at the end of a compound statement the state for the pragma is restored to
7903 its condition just before the compound statement. If this pragma is used in any other context, the
7904 behavior is undefined. If part of an application tests floating-point status flags, sets floating-
7905 point control modes, or runs under non-default mode settings, but was translated with the state
7906 for the `FENV_ACCESS` pragma off, the behavior is undefined. The default state (on or off) for
7907 the pragma is implementation-defined. (When execution passes from a part of the application
7908 translated with `FENV_ACCESS` off to a part translated with `FENV_ACCESS` on, the state of the
7909 floating-point status flags is unspecified and the floating-point control modes have their default
7910 settings.)

7911 APPLICATION USAGE

7912 This header is designed to support the floating-point exception status flags and directed-
7913 rounding control modes required by the IEC 60559:1989 standard, and other similar floating-
7914 point state information. Also it is designed to facilitate code portability among all systems.

7915 Certain application programming conventions support the intended model of use for the
7916 floating-point environment:

- 7917 • A function call does not alter its caller's floating-point control modes, clear its caller's
7918 floating-point status flags, nor depend on the state of its caller's floating-point status flags
7919 unless the function is so documented.
- 7920 • A function call is assumed to require default floating-point control modes, unless its
7921 documentation promises otherwise.

- A function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

With these conventions, an application can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the application that does so explicitly.

Even though the rounding direction macros may expand to constants corresponding to the values of `FLT_ROUNDS`, they are not required to do so.

For example:

```
#include <fenv.h>
void f(double x)
{
    #pragma STDC FENV_ACCESS ON
    void g(double);
    void h(double);
    /* ... */
    g(x + 1);
    h(x + 1);
    /* ... */
}
```

If the function `g()` might depend on status flags set as a side effect of the first `x+1`, or if the second `x+1` might depend on control modes set as a side effect of the call to function `g()`, then the application shall contain an appropriately placed invocation as follows:

```
#pragma STDC FENV_ACCESS ON
```

RATIONALE

The `fexcept_t` Type

`fexcept_t` does not have to be an integer type. Its values must be obtained by a call to `fegetexceptflag()`, and cannot be created by logical operations from the exception macros. An implementation might simply implement `fexcept_t` as an `int` and use the representations reflected by the exception macros, but is not required to; other representations might contain extra information about the exceptions. `fexcept_t` might be a `struct` with a member for each exception (that might hold the address of the first or last floating-point instruction that caused that exception). The ISO/IEC 9899:1999 standard makes no claims about the internals of an `fexcept_t`, and so the user cannot inspect it.

Exception and Rounding Macros

Macros corresponding to unsupported modes and rounding directions are not defined by the implementation and must not be defined by the application. An application might use `#ifndef` to test for this.

FUTURE DIRECTIONS

None.

SEE ALSO

XSH [feclearexcept\(\)](#), [fegetenv\(\)](#), [fegetexceptflag\(\)](#), [fegetround\(\)](#), [feholdexcept\(\)](#), [feraiseexcept\(\)](#), [fetestexcept\(\)](#), [feupdateenv\(\)](#)

7964

CHANGE HISTORY

7965

First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

7966

7967

7968

The return types for *feclearexcept()*, *fegetexceptflag()*, *feraiseexcept()*, *fesetexceptflag()*, *fegetenv()*, *fesetenv()*, and *feupdateenv()* are changed from **void** to **int** for alignment with the ISO/IEC 9899:1999 standard, Defect Report 202.

7969

Issue 7

7970

SD5-XBD-ERN-48 and ISO C TC2 #37 (SD5-XBD-ERN-49) are applied.

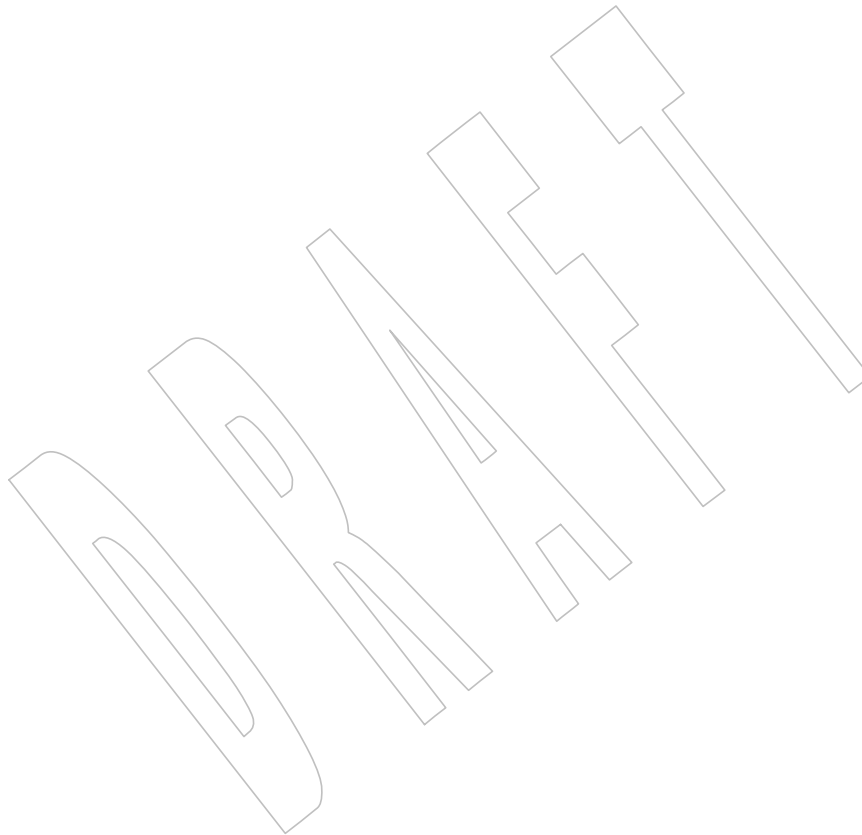
7971

SD5-XBD-ERN-69 is applied.

7972

This reference page is clarified with respect to macros and symbolic constants.

+



7973 NAME

7974 float.h — floating types

7975 SYNOPSIS

7976 #include <float.h>

7977 DESCRIPTION

7978 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 7979 conflict between the requirements described here and the ISO C standard is unintentional. This
 7980 volume of POSIX.1-200x defers to the ISO C standard.

7981 The characteristics of floating types are defined in terms of a model that describes a
 7982 representation of floating-point numbers and values that provide information about an
 7983 implementation's floating-point arithmetic.

7984 The following parameters are used to define the model for each floating-point type:

7985 *s* Sign (± 1).

7986 *b* Base or radix of exponent representation (an integer > 1).

7987 *e* Exponent (an integer between a minimum e_{\min} and a maximum e_{\max}).

7988 *p* Precision (the number of base-*b* digits in the significand).

7989 f_k Non-negative integers less than *b* (the significand digits).

7990 A floating-point number *x* is defined by the following model:

$$x = sb^e \sum_{k=1}^p f_k b^{-k}, e_{\min} \leq e \leq e_{\max}$$

7991 In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to
 7992 contain other kinds of floating-point numbers, such as subnormal floating-point numbers ($x \neq 0$,
 7993 $e = e_{\min}$, $f_1 = 0$) and unnormalized floating-point numbers ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are
 7994 not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-
 7995 Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a
 7996 floating-point exception; a *signaling NaN* generally raises a floating-point exception when
 7997 occurring as an arithmetic operand.

7998 An implementation may give zero and non-numeric values, such as infinities and NaNs, a sign,
 7999 or may leave them unsigned. Wherever such values are unsigned, any requirement in
 8000 POSIX.1-200x to retrieve the sign shall produce an unspecified sign and any requirement to set
 8001 the sign shall be ignored.

8002 The accuracy of the floating-point operations ('+', '-', '*', '/') and of the functions in
 8003 <math.h> and <complex.h> that return floating-point results is implementation-defined, as is
 8004 the accuracy of the conversion between floating-point internal representations and string
 8005 representations performed by the functions in <stdio.h>, <stdlib.h>, and <wchar.h>. The
 8006 implementation may state that the accuracy is unknown.

8007 All integer values in the <float.h> header, except FLT_ROUNDS, shall be constant expressions
 8008 suitable for use in #if preprocessing directives; all floating values shall be constant expressions.
 8009 All except DECIMAL_DIG, FLT_EVAL_METHOD, FLT_RADIX, and FLT_ROUNDS have
 8010 separate names for all three floating-point types. The floating-point model representation is
 8011 provided for all values except FLT_EVAL_METHOD and FLT_ROUNDS.

8012 The rounding mode for floating-point addition is characterized by the implementation-defined

8013 value of FLT_ROUNDS:

- 8014 -1 Indeterminable.
- 8015 0 Toward zero.
- 8016 1 To nearest.
- 8017 2 Toward positive infinity.
- 8018 3 Toward negative infinity.

8019 All other values for FLT_ROUNDS characterize implementation-defined rounding behavior.

8020 The values of operations with floating operands and values subject to the usual arithmetic
8021 conversions and of floating constants are evaluated to a format whose range and precision may
8022 be greater than required by the type. The use of evaluation formats is characterized by the
8023 implementation-defined value of FLT_EVAL_METHOD:

- 8024 -1 Indeterminable.
- 8025 0 Evaluate all operations and constants just to the range and precision of the type.
- 8026 1 Evaluate operations and constants of type **float** and **double** to the range and precision of
8027 the **double** type; evaluate **long double** operations and constants to the range and precision
8028 of the **long double** type.
- 8029 2 Evaluate all operations and constants to the range and precision of the **long double** type.

8030 All other negative values for FLT_EVAL_METHOD characterize implementation-defined
8031 behavior.

8032 The values given in the following list shall be defined as constant expressions with
8033 implementation-defined values that are greater or equal in magnitude (absolute value) to those
8034 shown, with the same sign.

- 8035 • Radix of exponent representation, b .
8036 FLT_RADIX 2
- 8037 • Number of base-FLT_RADIX digits in the floating-point significand, p .
8038 FLT_MANT_DIG
8039 DBL_MANT_DIG
8040 LDBL_MANT_DIG
- 8041 • Number of decimal digits, n , such that any floating-point number in the widest supported
8042 floating type with p_{\max} radix b digits can be rounded to a floating-point number with n
8043 decimal digits and back again without change to the value.

$$\left\{ \begin{array}{ll} p_{\max} \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \left\lceil 1 + p_{\max} \log_{10} b \right\rceil & \text{otherwise} \end{array} \right.$$

8044 DECIMAL_DIG 10

8045
8046
8047

- Number of decimal digits, q , such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits.

$$\begin{cases} p \log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil (p - 1) \log_{10} b \rceil & \text{otherwise} \end{cases}$$

8048
8049
8050

FLT_DIG 6
DBL_DIG 10
LDBL_DIG 10

8051
8052
8053
8054
8055

- Minimum negative integer such that FLT_RADIX raised to that power minus 1 is a normalized floating-point number, e_{\min} .

FLT_MIN_EXP
DBL_MIN_EXP
LDBL_MIN_EXP

8056
8057

- Minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers.

$$\lceil \log_{10} b^{e_{\min} - 1} \rceil$$

8058
8059
8060

FLT_MIN_10_EXP -37
DBL_MIN_10_EXP -37
LDBL_MIN_10_EXP -37

8061
8062
8063
8064
8065

- Maximum integer such that FLT_RADIX raised to that power minus 1 is a representable finite floating-point number, e_{\max} .

FLT_MAX_EXP
DBL_MAX_EXP
LDBL_MAX_EXP

8066
8067

- Maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers.

$$\lceil \log_{10}((1 - b^{-p}) b^{e_{\max}}) \rceil$$

8068
8069
8070

FLT_MAX_10_EXP +37
DBL_MAX_10_EXP +37
LDBL_MAX_10_EXP +37

8071
8072

The values given in the following list shall be defined as constant expressions with implementation-defined values that are greater than or equal to those shown:

- 8073
- Maximum representable finite floating-point number.

$$(1 - b^{-p}) b^{\ell_{\max}}$$

8074 FLT_MAX 1E+37

8075 DBL_MAX 1E+37

8076 LDBL_MAX 1E+37

8077 The values given in the following list shall be defined as constant expressions with
8078 implementation-defined (positive) values that are less than or equal to those shown:

- 8079
- The difference between 1 and the least value greater than 1 that is representable in the
8080 given floating-point type, b^{1-p} .

8081 FLT_EPSILON 1E-5

8082 DBL_EPSILON 1E-9

8083 LDBL_EPSILON 1E-9

- 8084
- Minimum normalized positive floating-point number, $b^{\ell_{\min}-1}$.

8085 FLT_MIN 1E-37

8086 DBL_MIN 1E-37

8087 LDBL_MIN 1E-37

8088 **APPLICATION USAGE**

8089 None.

8090 **RATIONALE**

8091 None.

8092 **FUTURE DIRECTIONS**

8093 None.

8094 **SEE ALSO**

8095 <complex.h>, <math.h>, <stdio.h>, <stdlib.h>, <wchar.h>

8096 **CHANGE HISTORY**

8097 First released in Issue 4. Derived from the ISO C standard.

8098 **Issue 6**

8099 The description of the operations with floating-point values is updated for alignment with the
8100 ISO/IEC 9899:1999 standard.

8101 **Issue 7**

8102 ISO C TC2 #4 (SD5-XBD-ERN-50) and ISO C TC2 #5 (SD5-XBD-ERN-51) are applied.

8103 **NAME**8104 `fmtmsg.h` — message display structures8105 **SYNOPSIS**8106 XSI `#include <fmtmsg.h>`8107 **DESCRIPTION**

8108 The <fmtmsg.h> header shall define the following symbolic constants:

8109	MM_HARD	Source of the condition is hardware.
8110	MM_SOFT	Source of the condition is software.
8111	MM_FIRM	Source of the condition is firmware.
8112	MM_APPL	Condition detected by application.
8113	MM_UTIL	Condition detected by utility.
8114	MM_OPSYS	Condition detected by operating system.
8115	MM_RECOVER	Recoverable error.
8116	MM_NRECOV	Non-recoverable error.
8117	MM_HALT	Error causing application to halt.
8118	MM_ERROR	Application has encountered a non-fatal fault.
8119	MM_WARNING	Application has detected unusual non-error condition.
8120	MM_INFO	Informative message.
8121	MM_NOSEV	No severity level provided for the message.
8122	MM_PRINT	Display message on standard error.
8123	MM_CONSOLE	Display message on system console.

8124 The table below indicates the null values and identifiers for *fmtmsg()* arguments. The
 8125 <fmtmsg.h> header shall define the symbolic constants in the **Identifier** column, which shall
 8126 have the type indicated in the **Type** column:

Argument	Type	Null-Value	Identifier
<i>label</i>	char *	(char*)0	MM_NULLLBL
<i>severity</i>	int	0	MM_NULLSEV
<i>class</i>	long	0L	MM_NULLMC
<i>text</i>	char *	(char*)0	MM_NULLTXT
<i>action</i>	char *	(char*)0	MM_NULLACT
<i>tag</i>	char *	(char*)0	MM_NULLTAG

8134 The <fmtmsg.h> header shall also define the following symbolic constants for use as return
 8135 values for *fmtmsg()*:

8136	MM_OK	The function succeeded.
8137	MM_NOTOK	The function failed completely.
8138	MM_NOMSG	The function was unable to generate a message on standard error, but 8139 otherwise succeeded.

8140 MM_NOCON The function was unable to generate a console message, but otherwise
8141 succeeded.

8142 The following shall be declared as a function and may also be defined as a macro. A function
8143 prototype shall be provided.

```
8144 int fmtmsg(long, const char *, int,  
8145 const char *, const char *, const char *);
```

8146 APPLICATION USAGE

8147 None.

8148 RATIONALE

8149 None.

8150 FUTURE DIRECTIONS

8151 None.

8152 SEE ALSO

8153 XSH *fmtmsg()*

8154 CHANGE HISTORY

8155 First released in Issue 4, Version 2.

8156 Issue 7

8157 This reference page is clarified with respect to macros and symbolic constants.

8158 **NAME**8159 `fnmatch.h` — filename-matching types8160 **SYNOPSIS**8161 `#include <fnmatch.h>`8162 **DESCRIPTION**

8163 The <fnmatch.h> header shall define the following symbolic constants: |

8164 `FNM_NOMATCH` The string does not match the specified pattern.8165 `FNM_PATHNAME` Slash in *string* only matches slash in *pattern*.8166 `FNM_PERIOD` Leading period in *string* must be exactly matched by period in *pattern*.8167 `FNM_NOESCAPE` Disable backslash escaping.8168 The following shall be declared as a function and may also be defined as a macro. A function
8169 prototype shall be provided.8170 `int fnmatch(const char *, const char *, int);`8171 **APPLICATION USAGE**

8172 None.

8173 **RATIONALE**

8174 None.

8175 **FUTURE DIRECTIONS**

8176 None.

8177 **SEE ALSO**8178 XSH *fnmatch()* |8179 **CHANGE HISTORY**

8180 First released in Issue 4. Derived from the ISO POSIX-2 standard.

8181 **Issue 6**8182 The `FNM_NOSYS` constant is marked obsolescent.8183 **Issue 7**8184 The obsolescent `FNM_NOSYS` constant is removed.

8185 This reference page is clarified with respect to macros and symbolic constants. +

8186 **NAME**

8187 ftw.h — file tree traversal

8188 **SYNOPSIS**8189 XSI `#include <ftw.h>`8190 **DESCRIPTION**8191 The <ftw.h> header shall define the **FTW** structure that includes at least the following members:8192 `int base`
8193 `int level`8194 The <ftw.h> header shall define the following symbolic constants for use as values of the third
8195 argument to the application-supplied function that is passed as the second argument to *ftw()*
8196 and *nftw()*:8197 **FTW_F** File.
8198 **FTW_D** Directory.
8199 **FTW_DNR** Directory without read permission.
8200 **FTW_DP** Directory with subdirectories visited.
8201 **FTW_NS** Unknown type; *stat()* failed.
8202 **FTW_SL** Symbolic link.
8203 **FTW_SLN** Symbolic link that names a nonexistent file.8204 The <ftw.h> header shall define the following symbolic constants for use as values of the fourth
8205 argument to *nftw()*:8206 **FTW_PHYS** Physical walk, does not follow symbolic links. Otherwise, *nftw()* follows
8207 links but does not walk down any path that crosses itself.
8208 **FTW_MOUNT** The walk does not cross a mount point.
8209 **FTW_DEPTH** All subdirectories are visited before the directory itself.
8210 **FTW_CHDIR** The walk changes to each directory before reading it.8211 The following shall be declared as functions and may also be defined as macros. Function
8212 prototypes shall be provided.8213 `int ftw(const char *, int (*)(const char *, const struct stat *,`
8214 `int), int);`
8215 `int nftw(const char *, int (*)(const char *, const struct stat *,`
8216 `int, struct FTW *), int, int);`8217 The <ftw.h> header shall define the **stat** structure and the symbolic names for *st_mode* and the
8218 file type test macros as described in <sys/stat.h>.

8219 Inclusion of the <ftw.h> header may also make visible all symbols from <sys/stat.h>.

8220 **APPLICATION USAGE**

8221 None.

8222 **RATIONALE**

8223 None.

8224 **FUTURE DIRECTIONS**

8225 None.

8226 **SEE ALSO**8227 [<sys/stat.h>](#)8228 XSH *ftw()*, *nftw()* |8229 **CHANGE HISTORY**

8230 First released in Issue 1. Derived from Issue 1 of the SVID.

8231 **Issue 5**

8232 A description of FTW_DP is added.

8233 **Issue 7**8234 The *ftw()* function is marked obsolescent.

8235 This reference page is clarified with respect to macros and symbolic constants. +

DRAFT

8236 **NAME**

8237 glob.h — pathname pattern-matching types

8238 **SYNOPSIS**

8239 #include <glob.h>

8240 **DESCRIPTION**8241 The <glob.h> header shall define the structures and symbolic constants used by the *glob()*
8242 function.8243 The structure type **glob_t** shall contain at least the following members:8244 `size_t gl_pathc` Count of paths matched by *pattern*.
8245 `char **gl_pathv` Pointer to a list of matched pathnames.
8246 `size_t gl_offs` Slots to reserve at the beginning of *gl_pathv*.8247 The **size_t** type shall be defined as described in <sys/types.h>.8248 The <glob.h> header shall define the following symbolic constants as values for the *flags* |
8249 argument:8250 **GLOB_APPEND** Append generated pathnames to those previously obtained.
8251 **GLOB_DOOFFS** Specify how many null pointers to add to the beginning of *gl_pathv*.
8252 **GLOB_ERR** Cause *glob()* to return on error.
8253 **GLOB_MARK** Each pathname that is a directory that matches *pattern* has a slash
8254 appended.
8255 **GLOB_NOCHECK** If *pattern* does not match any pathname, then return a list consisting of
8256 only *pattern*.
8257 **GLOB_NOESCAPE** Disable backslash escaping.
8258 **GLOB_NOSORT** Do not sort the pathnames returned.

8259 The <glob.h> header shall define the following symbolic constants as error return values: |

8260 **GLOB_ABORTED** The scan was stopped because **GLOB_ERR** was set or *(*errfunc)()*
8261 returned non-zero.
8262 **GLOB_NOMATCH** The *pattern* does not match any existing pathname, and
8263 **GLOB_NOCHECK** was not set in *flags*.
8264 **GLOB_NOSPACE** An attempt to allocate memory failed.8265 The following shall be declared as functions and may also be defined as macros. Function
8266 prototypes shall be provided.8267 `int glob(const char *restrict, int, int (*)(const char *, int),` |
8268 `glob_t *restrict);`
8269 `void globfree(glob_t *);`

8270	APPLICATION USAGE	-
8271	None.	
8272	RATIONALE	
8273	None.	
8274	FUTURE DIRECTIONS	
8275	None.	
8276	SEE ALSO	
8277	<sys/types.h>	
8278	XSH glob()	
8279	CHANGE HISTORY	
8280	First released in Issue 4. Derived from the ISO POSIX-2 standard.	
8281	Issue 6	
8282	The restrict keyword is added to the prototype for glob() .	
8283	The GLOB_NOSYS constant is marked obsolescent.	
8284	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/8 is applied, correcting the glob()	
8285	prototype definition by removing the restrict qualifier from the function pointer argument.	
8286	Issue 7	
8287	SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the size_t	
8288	The obsolescent GLOB_NOSYS constant is removed.	
8289	This reference page is clarified with respect to macros and symbolic constants.	+

8290 **NAME**

8291 grp.h — group structure

8292 **SYNOPSIS**

8293 #include <grp.h>

8294 **DESCRIPTION**8295 The <grp.h> header shall declare the structure **group** which shall include the following
8296 members:

8297 char *gr_name The name of the group.
 8298 gid_t gr_gid Numerical group ID.
 8299 char **gr_mem Pointer to a null-terminated array of character
 8300 pointers to member names.

8301 The **gid_t** and **size_t** types shall be defined as described in <sys/types.h>.8302 The following shall be declared as functions and may also be defined as macros. Function
8303 prototypes shall be provided.

```
8304 struct group *getgrgid(gid_t);
8305 struct group *getgrnam(const char *);
8306 int getgrgid_r(gid_t, struct group *, char *,
8307               size_t, struct group **);
8308 int getgrnam_r(const char *, struct group *, char *,
8309               size_t, struct group **);
8310 XSI struct group *getgrent(void);
8311 void endgrent(void);
8312 void setgrent(void);
```

8313 **APPLICATION USAGE**

8314 None.

8315 **RATIONALE**

8316 None.

8317 **FUTURE DIRECTIONS**

8318 None.

8319 **SEE ALSO**

8320 <sys/types.h>

8321 XSH *endgrent()*, *getgrgid()*, *getgrnam()*8322 **CHANGE HISTORY**

8323 First released in Issue 1.

8324 **Issue 5**

8325 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

8326 **Issue 6**8327 The following new requirements on POSIX implementations derive from alignment with the
8328 Single UNIX Specification:

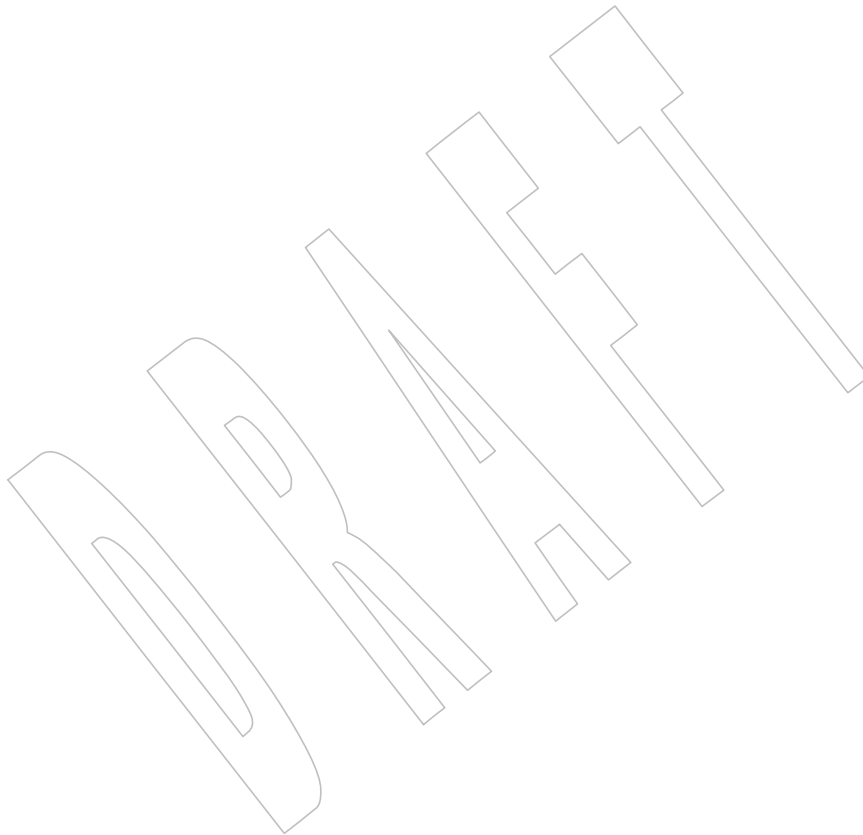
- 8329 • The definition of **gid_t** is mandated.
- 8330 • The *getgrgid_r()* and *getgrnam_r()* functions are marked as part of the Thread-Safe
8331 Functions option.

8332

Issue 7

8333

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.



8334 **NAME**

8335 iconv.h — codeset conversion facility

8336 **SYNOPSIS**

8337 #include <iconv.h>

8338 **DESCRIPTION**

8339 The <iconv.h> header shall define the following types:

8340 **iconv_t** Identifies the conversion from one codeset to another.8341 **size_t** As described in <sys/types.h>.8342 The following shall be declared as functions and may also be defined as macros. Function
8343 prototypes shall be provided.8344 size_t iconv(iconv_t, char **restrict, size_t *restrict,
8345 char **restrict, size_t *restrict);
8346 int iconv_close(iconv_t);
8347 iconv_t iconv_open(const char *, const char *);8348 **APPLICATION USAGE**

8349 None.

8350 **RATIONALE**

8351 None.

8352 **FUTURE DIRECTIONS**

8353 None.

8354 **SEE ALSO**

8355 <sys/types.h>

8356 XSH *iconv*, *iconv_close()*, *iconv_open()*8357 **CHANGE HISTORY**

8358 First released in Issue 4.

8359 **Issue 6**8360 The **restrict** keyword is added to the prototype for *iconv()*.8361 **Issue 7**8362 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

8363 The <iconv.h> header is moved from the XSI option to the Base.

8364 **NAME**

8365 inttypes.h — fixed size integer types

8366 **SYNOPSIS**

8367 #include <inttypes.h>

8368 **DESCRIPTION**

8369 CX Some of the functionality described on this reference page extends the ISO C standard.
 8370 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 8371 enable the visibility of these symbols in this header.

8372 The <inttypes.h> header shall include the <stdint.h> header.

8373 The <inttypes.h> header shall include a definition of at least the following type:

8374 **imaxdiv_t** Structure type that is the type of the value returned by the *imaxdiv()* function.

8375 The following macros shall be defined. Each expands to a character string literal containing a
 8376 conversion specifier, possibly modified by a length modifier, suitable for use within the *format*
 8377 argument of a formatted input/output function when converting the corresponding integer
 8378 type. These macros have the general form of PRI (character string literals for the *fprintf()* and
 8379 *fwprintf()* family of functions) or SCN (character string literals for the *fscanf()* and *fwscanf()*
 8380 family of functions), followed by the conversion specifier, followed by a name corresponding to
 8381 a similar type name in <stdint.h>. In these names, *N* represents the width of the type as
 8382 described in <stdint.h>. For example, *PRIdFAST32* can be used in a format string to print the
 8383 value of an integer of type *int_fast32_t*.

8384 The *fprintf()* macros for signed integers are:

8385	PRId <i>N</i>	PRIdLEAST <i>N</i>	PRIdFAST <i>N</i>	PRIdMAX	PRIdPTR
8386	PRIdi <i>N</i>	PRIdiLEAST <i>N</i>	PRIdiFAST <i>N</i>	PRIdiMAX	PRIdiPTR

8387 The *fprintf()* macros for unsigned integers are:

8388	PRIo <i>N</i>	PRIoLEAST <i>N</i>	PRIoFAST <i>N</i>	PRIoMAX	PRIoPTR
8389	PRIU <i>N</i>	PRIULEAST <i>N</i>	PRIUFAST <i>N</i>	PRIUMAX	PRIUPTR
8390	PRIx <i>N</i>	PRIXLEAST <i>N</i>	PRIXFAST <i>N</i>	PRIXMAX	PRIXPTR
8391	PRIXN	PRIXLEASTN	PRIXFASTN	PRIXMAX	PRIXPTR

8392 The *fscanf()* macros for signed integers are:

8393	SCNd <i>N</i>	SCNdLEAST <i>N</i>	SCNdFAST <i>N</i>	SCNdMAX	SCNdPTR
8394	SCNi <i>N</i>	SCNiLEAST <i>N</i>	SCNiFAST <i>N</i>	SCNiMAX	SCNiPTR

8395 The *fscanf()* macros for unsigned integers are:

8396	SCNo <i>N</i>	SCNoLEAST <i>N</i>	SCNoFAST <i>N</i>	SCNoMAX	SCNoPTR
8397	SCNu <i>N</i>	SCNuLEAST <i>N</i>	SCNuFAST <i>N</i>	SCNuMAX	SCNuPTR
8398	SCNx <i>N</i>	SCNxLEAST <i>N</i>	SCNxFAST <i>N</i>	SCNxMAX	SCNxPTR

8399 For each type that the implementation provides in <stdint.h>, the corresponding *fprintf()* and
 8400 *fwprintf()* macros shall be defined and the corresponding *fscanf()* and *fwscanf()* macros shall be
 8401 defined unless the implementation does not have a suitable modifier for the type.

8402 The following shall be declared as functions and may also be defined as macros. Function
 8403 prototypes shall be provided.

8404 `intmax_t imaxabs(intmax_t);`

```

8405     imaxdiv_t imaxdiv(intmax_t, intmax_t);
8406     intmax_t  strtouimax(const char *restrict, char **restrict, int);
8407     uintmax_t strtoumax(const char *restrict, char **restrict, int);
8408     intmax_t  wcstouimax(const wchar_t *restrict, wchar_t **restrict, int);
8409     uintmax_t wcstoumax(const wchar_t *restrict, wchar_t **restrict, int);

```

EXAMPLES

```

8410     #include <inttypes.h>
8411     #include <wchar.h>
8412     int main(void)
8413     {
8414         uintmax_t i = UINTMAX_MAX; // This type always exists.
8415         wprintf(L"The largest integer value is %020"
8416             PRIxMAX "\n", i);
8417         return 0;
8418     }
8419

```

APPLICATION USAGE

8420 The purpose of <inttypes.h> is to provide a set of integer types whose definitions are consistent
8421 across machines and independent of operating systems and other implementation
8422 idiosyncrasies. It defines, via **typedef**, integer types of various sizes. Implementations are free to
8423 **typedef** them as ISO C standard integer types or extensions that they support. Consistent use of
8424 this header will greatly increase the portability of applications across platforms.
8425

RATIONALE

8426 The ISO/IEC 9899:1990 standard specified that the language should support four signed and
8427 unsigned integer data types—**char**, **short**, **int**, and **long**—but placed very little requirement on
8428 their size other than that **int** and **short** be at least 16 bits and **long** be at least as long as **int** and
8429 not smaller than 32 bits. For 16-bit systems, most implementations assigned 8, 16, 16, and 32 bits
8430 to **char**, **short**, **int**, and **long**, respectively. For 32-bit systems, the common practice has been to
8431 assign 8, 16, 32, and 32 bits to these types. This difference in **int** size can create some problems
8432 for users who migrate from one system to another which assigns different sizes to integer types,
8433 because the ISO C standard integer promotion rule can produce silent changes unexpectedly.
8434 The need for defining an extended integer type increased with the introduction of 64-bit
8435 systems.
8436

FUTURE DIRECTIONS

8437 Macro names beginning with PRI or SCN followed by any lowercase letter or 'x' may be added
8438 to the macros defined in the <inttypes.h> header.
8439

SEE ALSO

8440 XSH [Section 2.2](#) (on page 448), *imaxdiv()*

CHANGE HISTORY

8441 First released in Issue 5.

Issue 6

8442 The Open Group Base Resolution bwg97-006 is applied.

8443 This reference page is updated to align with the ISO/IEC 9899:1999 standard.
8444
8445
8446

8447 **NAME**

8448 iso646.h — alternative spellings

8449 **SYNOPSIS**

8450 #include <iso646.h>

8451 **DESCRIPTION**

8452 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 8453 conflict between the requirements described here and the ISO C standard is unintentional. This
 8454 volume of POSIX.1-200x defers to the ISO C standard.

8455 The **<iso646.h>** header shall define the following eleven macros (on the left) that expand to the
 8456 corresponding tokens (on the right):

8457	and	&&
8458	and_eq	&=
8459	bitand	&
8460	bitor	
8461	compl	~
8462	not	!
8463	not_eq	!=
8464	or	
8465	or_eq	=
8466	xor	^
8467	xor_eq	^=

8468 **APPLICATION USAGE**

8469 None.

8470 **RATIONALE**

8471 None.

8472 **FUTURE DIRECTIONS**

8473 None.

8474 **SEE ALSO**

8475 None.

8476 **CHANGE HISTORY**

8477 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

8478 **NAME**
8479 langinfo.h — language information constants

8480 **SYNOPSIS**
8481 #include <langinfo.h>

8482 **DESCRIPTION**
8483 The <langinfo.h> header defines the symbolic constants used to identify items of *langinfo* data |
8484 (see *nl_langinfo()*). -

8485 The **nl_item** type shall be defined as described in <nl/types.h>. |

8486 The <langinfo.h> header shall define the following symbolic constants with type **nl_item**. The |
8487 entries under **Category** indicate in which *setlocale()* category each item is defined.

Constant	Category	Meaning
8488 CODESET	<i>LC_CTYPE</i>	Codeset name.
8489 D_T_FMT	<i>LC_TIME</i>	String for formatting date and time.
8490 D_FMT	<i>LC_TIME</i>	Date format string.
8491 T_FMT	<i>LC_TIME</i>	Time format string.
8492 T_FMT_AMPM	<i>LC_TIME</i>	a.m. or p.m. time format string.
8493 AM_STR	<i>LC_TIME</i>	Ante-meridiem affix.
8494 PM_STR	<i>LC_TIME</i>	Post-meridiem affix.
8495 DAY_1	<i>LC_TIME</i>	Name of the first day of the week (for example, Sunday).
8496 DAY_2	<i>LC_TIME</i>	Name of the second day of the week (for example, Monday).
8497 DAY_3	<i>LC_TIME</i>	Name of the third day of the week (for example, Tuesday).
8498 DAY_4	<i>LC_TIME</i>	Name of the fourth day of the week
8499		(for example, Wednesday).
8500 DAY_5	<i>LC_TIME</i>	Name of the fifth day of the week (for example, Thursday).
8501 DAY_6	<i>LC_TIME</i>	Name of the sixth day of the week (for example, Friday).
8502 DAY_7	<i>LC_TIME</i>	Name of the seventh day of the week
8503		(for example, Saturday).
8504 ABDAY_1	<i>LC_TIME</i>	Abbreviated name of the first day of the week.
8505 ABDAY_2	<i>LC_TIME</i>	Abbreviated name of the second day of the week.
8506 ABDAY_3	<i>LC_TIME</i>	Abbreviated name of the third day of the week.
8507 ABDAY_4	<i>LC_TIME</i>	Abbreviated name of the fourth day of the week.
8508 ABDAY_5	<i>LC_TIME</i>	Abbreviated name of the fifth day of the week.
8509 ABDAY_6	<i>LC_TIME</i>	Abbreviated name of the sixth day of the week.
8510 ABDAY_7	<i>LC_TIME</i>	Abbreviated name of the seventh day of the week.
8511 MON_1	<i>LC_TIME</i>	Name of the first month of the year.
8512 MON_2	<i>LC_TIME</i>	Name of the second month.
8513 MON_3	<i>LC_TIME</i>	Name of the third month.
8514 MON_4	<i>LC_TIME</i>	Name of the fourth month.
8515 MON_5	<i>LC_TIME</i>	Name of the fifth month.
8516 MON_6	<i>LC_TIME</i>	Name of the sixth month.
8517 MON_7	<i>LC_TIME</i>	Name of the seventh month.
8518 MON_8	<i>LC_TIME</i>	Name of the eighth month.
8519 MON_9	<i>LC_TIME</i>	Name of the ninth month.
8520 MON_10	<i>LC_TIME</i>	Name of the tenth month.
8521 MON_11	<i>LC_TIME</i>	Name of the eleventh month.
8522 MON_12	<i>LC_TIME</i>	Name of the twelfth month.
8523 ABMON_1	<i>LC_TIME</i>	Abbreviated name of the first month.
8524 ABMON_2	<i>LC_TIME</i>	Abbreviated name of the second month.

Constant	Category	Meaning
8526 ABMON_3	LC_TIME	Abbreviated name of the third month.
8527 ABMON_4	LC_TIME	Abbreviated name of the fourth month.
8528 ABMON_5	LC_TIME	Abbreviated name of the fifth month.
8529 ABMON_6	LC_TIME	Abbreviated name of the sixth month.
8530 ABMON_7	LC_TIME	Abbreviated name of the seventh month.
8531 ABMON_8	LC_TIME	Abbreviated name of the eighth month.
8532 ABMON_9	LC_TIME	Abbreviated name of the ninth month.
8533 ABMON_10	LC_TIME	Abbreviated name of the tenth month.
8534 ABMON_11	LC_TIME	Abbreviated name of the eleventh month.
8535 ABMON_12	LC_TIME	Abbreviated name of the twelfth month.
8536 ERA	LC_TIME	Era description segments.
8537 ERA_D_FMT	LC_TIME	Era date format string.
8538 ERA_D_T_FMT	LC_TIME	Era date and time format string.
8539 ERA_T_FMT	LC_TIME	Era time format string.
8540 ALT_DIGITS	LC_TIME	Alternative symbols for digits.
8541 RADIXCHAR	LC_NUMERIC	Radix character.
8542 THOUSEP	LC_NUMERIC	Separator for thousands.
8543 YESEXPR	LC_MESSAGES	Affirmative response expression.
8544 NOEXPR	LC_MESSAGES	Negative response expression.
8545 CRNCYSTR	LC_MONETARY	Local currency symbol, preceded by '-' if the symbol should appear before the value, '+' if the symbol should appear after the value, or '.' if the symbol should replace the radix character. If the local currency symbol is the empty string, implementations may return the empty string ("").

8551 If the locale's values for **p_cs_precedes** and **n_cs_precedes** do not match, the value of
 8552 *nl_langinfo*(CRNCYSTR) and *nl_langinfo_l*(CRNCYSTR,loc) is unspecified. +

8553 The following shall be declared as a function and may also be defined as a macro. A function
 8554 prototype shall be provided.

```
8555 char *nl_langinfo(nl_item);
8556 char *nl_langinfo_l(nl_item, locale_t);
```

8557 Inclusion of the <langinfo.h> header may also make visible all symbols from <nl_types.h>.

8558 APPLICATION USAGE

8559 Wherever possible, users are advised to use functions compatible with those in the ISO C
 8560 standard to access items of *langinfo* data. In particular, the *strftime*() function should be used to
 8561 access date and time information defined in category *LC_TIME*. The *localeconv*() function
 8562 should be used to access information corresponding to *RADIXCHAR*, *THOUSEP*, and
 8563 *CRNCYSTR*.

8564 RATIONALE

8565 None.

8566 FUTURE DIRECTIONS

8567 None.

8568 SEE ALSO

8569 [Chapter 7](#) (on page 121), <nl_types.h>

8570 XSH *nl_langinfo*(), *localeconv*(), *strfmon*(), *strftime*() |

8571
8572
8573
8574
8575
8576
8577
8578
8579
8580
8581
8582
8583
8584**CHANGE HISTORY**

First released in Issue 2.

Issue 5

The constants YESSTR and NOSTR are marked LEGACY.

Issue 6

The constants YESSTR and NOSTR are removed.

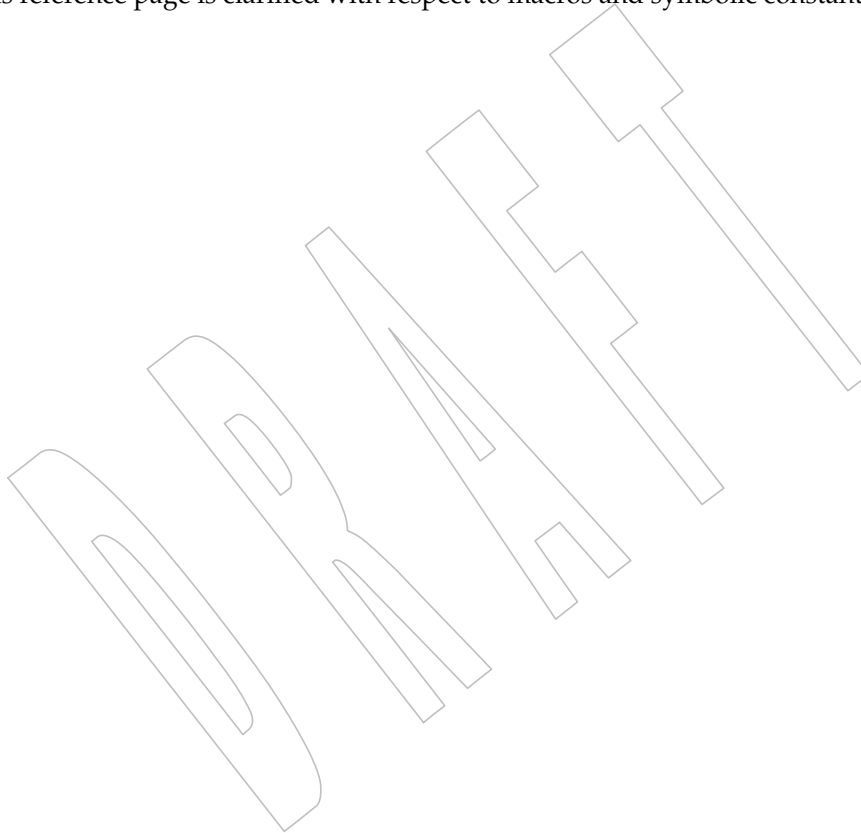
IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/9 is applied, adding a sentence to the "Meaning" column entry for the CRNCYSTR constant. This change is to accommodate historic practice.

Issue 7

The <langinfo.h> header is moved from the XSI option to the Base.

The *nl_langinfo_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

This reference page is clarified with respect to macros and symbolic constants. +



8585 **NAME**
 8586 libgen.h — definitions for pattern matching functions

8587 **SYNOPSIS**
 8588 XSI `#include <libgen.h>`

8589 **DESCRIPTION**
 8590 The following shall be declared as functions and may also be defined as macros. Function
 8591 prototypes shall be provided.

8592 `char *basename(char *);`
 8593 `char *dirname(char *);`

8594 **APPLICATION USAGE**
 8595 None.

8596 **RATIONALE**
 8597 None.

8598 **FUTURE DIRECTIONS**
 8599 None.

8600 **SEE ALSO**
 8601 XSH *basename*, *dirname*

8602 **CHANGE HISTORY**
 8603 First released in Issue 4, Version 2.

8604 **Issue 5**
 8605 The function prototypes for *basename()* and *dirname()* are changed to indicate that the first
 8606 argument is of type **char *** rather than **const char ***.

8607 **Issue 6**
 8608 The `__loc1` symbol and the *regcmp()* and *regex()* functions are removed.

8609 **NAME**
 8610 limits.h — implementation-defined constants

8611 **SYNOPSIS**
 8612 #include <limits.h>

8613 **DESCRIPTION**

8614 CX Some of the functionality described on this reference page extends the ISO C standard.
 8615 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 8616 enable the visibility of these symbols in this header.

8617 Many of the symbols listed here are not defined by the ISO/IEC 9899:1999 standard. Such
 8618 symbols are not shown as CX shaded.

8619 The <limits.h> header shall define macros and symbolic constants for various limits. Different
 8620 categories of limits are described below, representing various limits on resources that the
 8621 implementation imposes on applications. All macros and symbolic constants defined in this
 8622 header shall be suitable for use in #if preprocessing directives.

8623 Implementations may choose any appropriate value for each limit, provided it is not more
 8624 restrictive than the Minimum Acceptable Values listed below. Symbolic constant names
 8625 beginning with _POSIX may be found in <unistd.h>.

8626 Applications should not assume any particular value for a limit. To achieve maximum
 8627 portability, an application should not require more resource than the Minimum Acceptable
 8628 Value quantity. However, an application wishing to avail itself of the full amount of a resource
 8629 available on an implementation may make use of the value given in <limits.h> on that
 8630 particular implementation, by using the macros and symbolic constants listed below. It should
 8631 be noted, however, that many of the listed limits are not invariant, and at runtime, the value of
 8632 the limit may differ from those given in this header, for the following reasons:

- 8633 • The limit is pathname-dependent.
- 8634 • The limit differs between the compile and runtime machines.

8635 For these reasons, an application may use the *fpathconf()*, *pathconf()*, and *sysconf()* functions to
 8636 determine the actual value of a limit at runtime.

8637 The items in the list ending in _MIN give the most negative values that the mathematical types
 8638 are guaranteed to be capable of representing. Numbers of a more negative value may be
 8639 supported on some implementations, as indicated by the <limits.h> header on the
 8640 implementation, but applications requiring such numbers are not guaranteed to be portable to
 8641 all implementations. For positive constants ending in _MIN, this indicates the minimum
 8642 acceptable value.

8643 **Runtime Invariant Values (Possibly Indeterminate)**

8644 A definition of one of the symbolic constants in the following list shall be omitted from
 8645 <limits.h> on specific implementations where the corresponding value is equal to or greater
 8646 than the stated minimum, but is unspecified.

8647 This indetermination might depend on the amount of available memory space on a specific
 8648 instance of a specific implementation. The actual value supported by a specific instance shall be
 8649 provided by the *sysconf()* function.

8650 {AIO_LISTIO_MAX}
 8651 Maximum number of I/O operations in a single list I/O call supported by the
 8652 implementation.

8653 Minimum Acceptable Value: {_POSIX_AIO_LISTIO_MAX}

8654 {AIO_MAX}

8655 Maximum number of outstanding asynchronous I/O operations supported by the

8656 implementation.

8657 Minimum Acceptable Value: {_POSIX_AIO_MAX}

8658 {AIO_PRIO_DELTA_MAX}

8659 The maximum amount by which a process can decrease its asynchronous I/O priority level

8660 from its own scheduling priority.

8661 Minimum Acceptable Value: 0

8662 {ARG_MAX}

8663 Maximum length of argument to the *exec* functions including environment data.

8664 Minimum Acceptable Value: {_POSIX_ARG_MAX}

8665 {ATEXIT_MAX}

8666 Maximum number of functions that may be registered with *atexit()*.

8667 Minimum Acceptable Value: 32

8668 {CHILD_MAX}

8669 Maximum number of simultaneous processes per real user ID.

8670 Minimum Acceptable Value: {_POSIX_CHILD_MAX}

8671 {DELAYTIMER_MAX}

8672 Maximum number of timer expiration overruns.

8673 Minimum Acceptable Value: {_POSIX_DELAYTIMER_MAX}

8674 {HOST_NAME_MAX}

8675 Maximum length of a host name (not including the terminating null) as returned from the

8676 *gethostname()* function.

8677 Minimum Acceptable Value: {_POSIX_HOST_NAME_MAX}

8678 XSI {IOV_MAX}

8679 Maximum number of *iovec* structures that one process has available for use with *readv()* or

8680 *writev()*.

8681 Minimum Acceptable Value: {_XOPEN_IOV_MAX}

8682 {LOGIN_NAME_MAX}

8683 Maximum length of a login name.

8684 Minimum Acceptable Value: {_POSIX_LOGIN_NAME_MAX}

8685 MSG {MQ_OPEN_MAX}

8686 The maximum number of open message queue descriptors a process may hold.

8687 Minimum Acceptable Value: {_POSIX_MQ_OPEN_MAX}

8688 MSG {MQ_PRIO_MAX}

8689 The maximum number of message priorities supported by the implementation.

8690 Minimum Acceptable Value: {_POSIX_MQ_PRIO_MAX}

8691 {OPEN_MAX}

8692 A value one greater than the maximum value that the system may assign to a newly-created

8693 file descriptor.

8694 Minimum Acceptable Value: {_POSIX_OPEN_MAX}

8695 {PAGESIZE}

8696 Size in bytes of a page.

8697 Minimum Acceptable Value: 1

8698	XSI	{PAGE_SIZE}
8699		Equivalent to {PAGESIZE}. If either {PAGESIZE} or {PAGE_SIZE} is defined, the other is
8700		defined with the same value.
8701		{PTHREAD_DESTRUCTOR_ITERATIONS}
8702		Maximum number of attempts made to destroy a thread's thread-specific data values on
8703		thread exit.
8704		Minimum Acceptable Value: {_POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8705		{PTHREAD_KEYS_MAX}
8706		Maximum number of data keys that can be created by a process.
8707		Minimum Acceptable Value: {_POSIX_THREAD_KEYS_MAX}
8708		{PTHREAD_STACK_MIN}
8709		Minimum size in bytes of thread stack storage.
8710		Minimum Acceptable Value: 0
8711		{PTHREAD_THREADS_MAX}
8712		Maximum number of threads that can be created per process.
8713		Minimum Acceptable Value: {_POSIX_THREAD_THREADS_MAX}
8714		{RE_DUP_MAX}
8715		The number of repeated occurrences of a BRE or ERE interval expression; see Section 9.3.6
8716		(on page 172) and Section 9.4.6 (on page 175).
8717		Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}
8718		{RTSIG_MAX}
8719		Maximum number of realtime signals reserved for application use in this implementation.
8720		Minimum Acceptable Value: {_POSIX_RTSIG_MAX}
8721		{SEM_NSEMS_MAX}
8722		Maximum number of semaphores that a process may have.
8723		Minimum Acceptable Value: {_POSIX_SEM_NSEMS_MAX}
8724		{SEM_VALUE_MAX}
8725		The maximum value a semaphore may have.
8726		Minimum Acceptable Value: {_POSIX_SEM_VALUE_MAX}
8727		{SIGQUEUE_MAX}
8728		Maximum number of queued signals that a process may send and have pending at the
8729		receiver(s) at any time.
8730		Minimum Acceptable Value: {_POSIX_SIGQUEUE_MAX}
8731	SSI/TSP	{SS_REPL_MAX}
8732		The maximum number of replenishment operations that may be simultaneously pending
8733		for a particular sporadic server scheduler.
8734		Minimum Acceptable Value: {_POSIX_SS_REPL_MAX}
8735		{STREAM_MAX}
8736		The number of streams that one process can have open at one time. If defined, it has the
8737		same value as {FOPEN_MAX} (see <stdio.h>).
8738		Minimum Acceptable Value: {_POSIX_STREAM_MAX}
8739		{SYMLOOP_MAX}
8740		Maximum number of symbolic links that can be reliably traversed in the resolution of a
8741		pathname in the absence of a loop.
8742		Minimum Acceptable Value: {_POSIX_SYMLOOP_MAX}
8743		{TIMER_MAX}
8744		Maximum number of timers per process supported by the implementation.
8745		Minimum Acceptable Value: {_POSIX_TIMER_MAX}

8746 OB TRC {TRACE_EVENT_NAME_MAX}
 8747 Maximum length of the trace event name.
 8748 Minimum Acceptable Value: {_POSIX_TRACE_EVENT_NAME_MAX}

8749 OB TRC {TRACE_NAME_MAX}
 8750 Maximum length of the trace generation version string or of the trace stream name.
 8751 Minimum Acceptable Value: {_POSIX_TRACE_NAME_MAX}

8752 OB TRC {TRACE_SYS_MAX}
 8753 Maximum number of trace streams that may simultaneously exist in the system.
 8754 Minimum Acceptable Value: {_POSIX_TRACE_SYS_MAX}

8755 OB TRC {TRACE_USER_EVENT_MAX}
 8756 Maximum number of user trace event type identifiers that may simultaneously exist in a
 8757 traced process, including the predefined user trace event
 8758 POSIX_TRACE_UNNAMED_USER_EVENT.
 8759 Minimum Acceptable Value: {_POSIX_TRACE_USER_EVENT_MAX}

8760 {TTY_NAME_MAX}
 8761 Maximum length of terminal device name.
 8762 Minimum Acceptable Value: {_POSIX_TTY_NAME_MAX}

8763 {TZNAME_MAX}
 8764 Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).
 8765 Minimum Acceptable Value: {_POSIX_TZNAME_MAX}

8766 **Note:** The length given by {TZNAME_MAX} does not include the quoting characters mentioned in
 8767 Section 8.3 (on page 163).

8768 Pathname Variable Values

8769 The values in the following list may be constants within an implementation or may vary from
 8770 one pathname to another. For example, file systems or directories may have different
 8771 characteristics.

8772 A definition of one of the symbolic constants in the following list shall be omitted from the
 8773 <limits.h> header on specific implementations where the corresponding value is equal to or
 8774 greater than the stated minimum, but where the value can vary depending on the file to which it
 8775 is applied. The actual value supported for a specific pathname shall be provided by the
 8776 *pathconf()* function.

8777 {FILESIZEBITS}
 8778 Minimum number of bits needed to represent, as a signed integer value, the maximum size
 8779 of a regular file allowed in the specified directory.
 8780 Minimum Acceptable Value: 32

8781 {LINK_MAX}
 8782 Maximum number of links to a single file.
 8783 Minimum Acceptable Value: {_POSIX_LINK_MAX}

8784 {MAX_CANON}
 8785 Maximum number of bytes in a terminal canonical input line.
 8786 Minimum Acceptable Value: {_POSIX_MAX_CANON}

8787 {MAX_INPUT}
 8788 Minimum number of bytes for which space is available in a terminal input queue; therefore,
 8789 the maximum number of bytes a conforming application may require to be typed as input
 8790 before reading them.
 8791 Minimum Acceptable Value: {_POSIX_MAX_INPUT}

8792		{NAME_MAX}
8793		Maximum number of bytes in a filename (not including terminating null).
8794		Minimum Acceptable Value: {_POSIX_NAME_MAX}
8795	XSI	Minimum Acceptable Value: {_XOPEN_NAME_MAX}
8796		{PATH_MAX}
8797		Maximum number of bytes in a pathname, including the terminating null character.
8798		Minimum Acceptable Value: {_POSIX_PATH_MAX}
8799	XSI	Minimum Acceptable Value: {_XOPEN_PATH_MAX}
8800		{PIPE_BUF}
8801		Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
8802		Minimum Acceptable Value: {_POSIX_PIPE_BUF}
8803	ADV	{POSIX_ALLOC_SIZE_MIN}
8804		Minimum number of bytes of storage actually allocated for any portion of a file.
8805		Minimum Acceptable Value: Not specified.
8806	ADV	{POSIX_REC_INCR_XFER_SIZE}
8807		Recommended increment for file transfer sizes between the
8808		{POSIX_REC_MIN_XFER_SIZE} and {POSIX_REC_MAX_XFER_SIZE} values.
8809		Minimum Acceptable Value: Not specified.
8810	ADV	{POSIX_REC_MAX_XFER_SIZE}
8811		Maximum recommended file transfer size.
8812		Minimum Acceptable Value: Not specified.
8813	ADV	{POSIX_REC_MIN_XFER_SIZE}
8814		Minimum recommended file transfer size.
8815		Minimum Acceptable Value: Not specified.
8816	ADV	{POSIX_REC_XFER_ALIGN}
8817		Recommended file transfer buffer alignment.
8818		Minimum Acceptable Value: Not specified.
8819		{SYMLINK_MAX}
8820		Maximum number of bytes in a symbolic link.
8821		Minimum Acceptable Value: {_POSIX_SYMLINK_MAX}

8822 Runtime Inceasable Values

8823 The magnitude limitations in the following list shall be fixed by specific implementations. An
 8824 application should assume that the value of the symbolic constant defined by <limits.h> in a
 8825 specific implementation is the minimum that pertains whenever the application is run under
 8826 that implementation. A specific instance of a specific implementation may increase the value
 8827 relative to that supplied by <limits.h> for that implementation. The actual value supported by a
 8828 specific instance shall be provided by the *sysconf()* function.

8829		{BC_BASE_MAX}
8830		Maximum <i>obase</i> values allowed by the <i>bc</i> utility.
8831		Minimum Acceptable Value: {_POSIX2_BC_BASE_MAX}
8832		{BC_DIM_MAX}
8833		Maximum number of elements permitted in an array by the <i>bc</i> utility.
8834		Minimum Acceptable Value: {_POSIX2_BC_DIM_MAX}
8835		{BC_SCALE_MAX}
8836		Maximum <i>scale</i> value allowed by the <i>bc</i> utility.
8837		Minimum Acceptable Value: {_POSIX2_BC_SCALE_MAX}

8838 {BC_STRING_MAX}
 8839 Maximum length of a string constant accepted by the *bc* utility.
 8840 Minimum Acceptable Value: {_POSIX2_BC_STRING_MAX}

8841 {CHARCLASS_NAME_MAX}
 8842 Maximum number of bytes in a character class name.
 8843 Minimum Acceptable Value: {_POSIX2_CHARCLASS_NAME_MAX}

8844 {COLL_WEIGHTS_MAX}
 8845 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
 8846 keyword in the locale definition file; see [Chapter 7](#) (on page 121).
 8847 Minimum Acceptable Value: {_POSIX2_COLL_WEIGHTS_MAX}

8848 {EXPR_NEST_MAX}
 8849 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
 8850 Minimum Acceptable Value: {_POSIX2_EXPR_NEST_MAX}

8851 {LINE_MAX}
 8852 Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
 8853 standard input or another file), when the utility is described as processing text files. The
 8854 length includes room for the trailing <newline>.
 8855 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

8856 {NGROUPS_MAX}
 8857 Maximum number of simultaneous supplementary group IDs per process.
 8858 Minimum Acceptable Value: {_POSIX2_NGROUPS_MAX}

8859 {RE_DUP_MAX}
 8860 Maximum number of repeated occurrences of a regular expression permitted when using
 8861 the interval notation $\{m,n\}$; see [Chapter 9](#) (on page 167).
 8862 Minimum Acceptable Value: {_POSIX2_RE_DUP_MAX}

Maximum Values

8863
 8864 The symbolic constants in the following list shall be defined in <limits.h> with the values
 8865 shown. These are the most restrictive values for certain features on an implementation. A
 8866 conforming implementation shall provide values no larger than these values. A conforming
 8867 application must not require a smaller value for correct operation.

8868 {_POSIX_CLOCKRES_MIN}
 8869 The resolution of the CLOCK_REALTIME clock, in nanoseconds.
 8870 Value: 20 000 000

8871 MON If the Monotonic Clock option is supported, the resolution of the CLOCK_MONOTONIC
 8872 clock, in nanoseconds, is represented by {_POSIX_CLOCKRES_MIN}.

Minimum Values

8873
 8874 The symbolic constants in the following list shall be defined in <limits.h> with the values
 8875 shown. These are the most restrictive values for certain features on an implementation
 8876 conforming to this volume of POSIX.1-200x. Related symbolic constants are defined elsewhere in
 8877 this volume of POSIX.1-200x which reflect the actual implementation and which need not be as
 8878 restrictive. A conforming implementation shall provide values at least this large. A strictly
 8879 conforming application must not require a larger value for correct operation.

8880 {_POSIX_AIO_LISTIO_MAX}
 8881 The number of I/O operations that can be specified in a list I/O call.
 8882 Value: 2

8883	{_POSIX_AIO_MAX}	
8884		The number of outstanding asynchronous I/O operations.
8885		Value: 1
8886	{_POSIX_ARG_MAX}	
8887		Maximum length of argument to the <i>exec</i> functions including environment data.
8888		Value: 4 096
8889	{_POSIX_CHILD_MAX}	
8890		Maximum number of simultaneous processes per real user ID.
8891		Value: 25
8892	{_POSIX_DELAYTIMER_MAX}	
8893		The number of timer expiration overruns.
8894		Value: 32
8895	{_POSIX_HOST_NAME_MAX}	
8896		Maximum length of a host name (not including the terminating null) as returned from the <i>gethostname()</i> function.
8897		Value: 255
8899	{_POSIX_LINK_MAX}	
8900		Maximum number of links to a single file.
8901		Value: 8
8902	{_POSIX_LOGIN_NAME_MAX}	
8903		The size of the storage required for a login name, in bytes, including the terminating null.
8904		Value: 9
8905	{_POSIX_MAX_CANON}	
8906		Maximum number of bytes in a terminal canonical input queue.
8907		Value: 255
8908	{_POSIX_MAX_INPUT}	
8909		Maximum number of bytes allowed in a terminal input queue.
8910		Value: 255
8911	MSG	{_POSIX_MQ_OPEN_MAX}
8912		The number of message queues that can be open for a single process.
8913		Value: 8
8914	MSG	{_POSIX_MQ_PRIO_MAX}
8915		The maximum number of message priorities supported by the implementation.
8916		Value: 32
8917		{_POSIX_NAME_MAX}
8918		Maximum number of bytes in a filename (not including terminating null).
8919		Value: 14
8920		{_POSIX_NGROUPS_MAX}
8921		Maximum number of simultaneous supplementary group IDs per process.
8922		Value: 8
8923		{_POSIX_OPEN_MAX}
8924		Maximum number of files that one process can have open at any one time.
8925		Value: 20
8926		{_POSIX_PATH_MAX}
8927		Maximum number of bytes in a pathname.
8928		Value: 256

8929 { _POSIX_PIPE_BUF}
8930 Maximum number of bytes that is guaranteed to be atomic when writing to a pipe.
8931 Value: 512

8932 { _POSIX_RE_DUP_MAX}
8933 The number of repeated occurrences of a BRE permitted by the *regex*(*)* and *regcomp*(*)*
8934 functions when using the interval notation $\{m,n\}$; see [Section 9.3.6](#) (on page 172).
8935 Value: 255

8936 { _POSIX_RTSIG_MAX}
8937 The number of realtime signal numbers reserved for application use.
8938 Value: 8

8939 { _POSIX_SEM_NSEMS_MAX}
8940 The number of semaphores that a process may have.
8941 Value: 256

8942 { _POSIX_SEM_VALUE_MAX}
8943 The maximum value a semaphore may have.
8944 Value: 32 767

8945 { _POSIX_SIGQUEUE_MAX}
8946 The number of queued signals that a process may send and have pending at the receiver(s)
8947 at any time.
8948 Value: 32

8949 { _POSIX_SSIZE_MAX}
8950 The value that can be stored in an object of type **ssize_t**.
8951 Value: 32 767

8952 SS | TSP { _POSIX_SS_REPL_MAX}
8953 The number of replenishment operations that may be simultaneously pending for a
8954 particular sporadic server scheduler.
8955 Value: 4

8956 { _POSIX_STREAM_MAX}
8957 The number of streams that one process can have open at one time.
8958 Value: 8

8959 { _POSIX_SYMLINK_MAX}
8960 The number of bytes in a symbolic link.
8961 Value: 255

8962 { _POSIX_SYMLINK_MAX}
8963 The number of symbolic links that can be traversed in the resolution of a pathname in the
8964 absence of a loop.
8965 Value: 8

8966 { _POSIX_THREAD_DESTRUCTOR_ITERATIONS}
8967 The number of attempts made to destroy a thread's thread-specific data values on thread
8968 exit.
8969 Value: 4

8970 { _POSIX_THREAD_KEYS_MAX}
8971 The number of data keys per process.
8972 Value: 128

8973 { _POSIX_THREAD_THREADS_MAX}
8974 The number of threads per process.
8975 Value: 64

8976 { _POSIX_TIMER_MAX}
8977 The per-process number of timers.
8978 Value: 32

8979 OB TRC { _POSIX_TRACE_EVENT_NAME_MAX}
8980 The length in bytes of a trace event name.
8981 Value: 30

8982 OB TRC { _POSIX_TRACE_NAME_MAX}
8983 The length in bytes of a trace generation version string or a trace stream name.
8984 Value: 8

8985 OB TRC { _POSIX_TRACE_SYS_MAX}
8986 The number of trace streams that may simultaneously exist in the system.
8987 Value: 8

8988 OB TRC { _POSIX_TRACE_USER_EVENT_MAX}
8989 The number of user trace event type identifiers that may simultaneously exist in a traced
8990 process, including the predefined user trace event
8991 POSIX_TRACE_UNNAMED_USER_EVENT.
8992 Value: 32

8993 { _POSIX_TTY_NAME_MAX}
8994 The size of the storage required for a terminal device name, in bytes, including the
8995 terminating null.
8996 Value: 9

8997 { _POSIX_TZNAME_MAX}
8998 Maximum number of bytes supported for the name of a timezone (not of the *TZ* variable).
8999 Value: 6

9000 **Note:** The length given by { _POSIX_TZNAME_MAX} does not include the quoting characters
9001 mentioned in [Section 8.3](#) (on page 163).

9002 { _POSIX2_BC_BASE_MAX}
9003 Maximum *obase* values allowed by the *bc* utility.
9004 Value: 99

9005 { _POSIX2_BC_DIM_MAX}
9006 Maximum number of elements permitted in an array by the *bc* utility.
9007 Value: 2 048

9008 { _POSIX2_BC_SCALE_MAX}
9009 Maximum *scale* value allowed by the *bc* utility.
9010 Value: 99

9011 { _POSIX2_BC_STRING_MAX}
9012 Maximum length of a string constant accepted by the *bc* utility.
9013 Value: 1 000

9014 { _POSIX2_CHARCLASS_NAME_MAX}
9015 Maximum number of bytes in a character class name.
9016 Value: 14

9017 { _POSIX2_COLL_WEIGHTS_MAX}
9018 Maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order**
9019 keyword in the locale definition file; see [Chapter 7](#) (on page 121).
9020 Value: 2

9021 { _POSIX2_EXPR_NEST_MAX}
9022 Maximum number of expressions that can be nested within parentheses by the *expr* utility.
9023 Value: 32

9024		{_POSIX2_LINE_MAX}
9025		Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either
9026		standard input or another file), when the utility is described as processing text files. The
9027		length includes room for the trailing <newline>.
9028		Value: 2 048
9029		{_POSIX2_RE_DUP_MAX}
9030		Maximum number of repeated occurrences of a regular expression permitted when using
9031		the interval notation $\{m,n\}$; see Chapter 9 (on page 167).
9032		Value: 255
9033	XSI	{_XOPEN_IOV_MAX}
9034		Maximum number of iovec structures that one process has available for use with <i>readv()</i> or
9035		<i>writev()</i> .
9036		Value: 16
9037	XSI	{_XOPEN_NAME_MAX}
9038		Maximum number of bytes in a filename (not including the terminating null).
9039		Value: 255
9040	XSI	{_XOPEN_PATH_MAX}
9041		Maximum number of bytes in a pathname.
9042		Value: 1 024

9043 Numerical Limits

9044 The macros in the following list shall be defined in <limits.h> and, except for {CHAR_BIT}, |
 9045 {LONG_BIT}, {MB_LEN_MAX}, and {WORD_BIT}, shall be replaced by expressions that have |
 9046 the same type as would an expression that is an object of the corresponding type converted |
 9047 according to the integer promotions.

9048 If the value of an object of type **char** is treated as a signed integer when used in an expression,
 9049 the value of {CHAR_MIN} is the same as that of {SCHAR_MIN} and the value of {CHAR_MAX}
 9050 is the same as that of {SCHAR_MAX}. Otherwise, the value of {CHAR_MIN} is 0 and the value
 9051 of {CHAR_MAX} is the same as that of {UCHAR_MAX}.

9052		{CHAR_BIT}
9053		Number of bits in a type char .
9054	CX	Value: 8
9055		{CHAR_MAX}
9056		Maximum value of type char .
9057		Value: {UCHAR_MAX} or {SCHAR_MAX}
9058		{CHAR_MIN}
9059		Minimum value of type char .
9060		Value: {SCHAR_MIN} or 0
9061		{INT_MAX}
9062		Maximum value of an int .
9063	CX	Minimum Acceptable Value: 2 147 483 647
9064		{INT_MIN}
9065		Minimum value of type int .
9066	CX	Maximum Acceptable Value: -2 147 483 647
9067		{LLONG_MAX}
9068		Maximum value of type long long .
9069		Minimum Acceptable Value: +9 223 372 036 854 775 807

9070 {LLONG_MIN}
 9071 Minimum value of type **long long**.
 9072 Maximum Acceptable Value: -9 223 372 036 854 775 807

9073 {LONG_BIT}
 9074 Number of bits in a **long**.
 9075 Minimum Acceptable Value: 32

9076 {LONG_MAX}
 9077 Maximum value of a **long**.
 9078 Minimum Acceptable Value: +2 147 483 647

9079 {LONG_MIN}
 9080 Minimum value of type **long**.
 9081 Maximum Acceptable Value: -2 147 483 647

9082 {MB_LEN_MAX}
 9083 Maximum number of bytes in a character, for any supported locale.
 9084 Minimum Acceptable Value: 1

9085 {SCHAR_MAX}
 9086 Maximum value of type **signed char**.
 9087 CX Value: +127

9088 {SCHAR_MIN}
 9089 Minimum value of type **signed char**.
 9090 CX Value: -128

9091 {SHRT_MAX}
 9092 Maximum value of type **short**.
 9093 Minimum Acceptable Value: +32 767

9094 {SHRT_MIN}
 9095 Minimum value of type **short**.
 9096 Maximum Acceptable Value: -32 767

9097 {SSIZE_MAX}
 9098 Maximum value of an object of type **ssize_t**.
 9099 Minimum Acceptable Value: {_POSIX_SSIZE_MAX}

9100 {UCHAR_MAX}
 9101 Maximum value of type **unsigned char**.
 9102 CX Value: 255

9103 {UINT_MAX}
 9104 Maximum value of type **unsigned**.
 9105 CX Minimum Acceptable Value: 4 294 967 295

9106 {ULLONG_MAX}
 9107 Maximum value of type **unsigned long long**.
 9108 Minimum Acceptable Value: 18 446 744 073 709 551 615

9109 {ULONG_MAX}
 9110 Maximum value of type **unsigned long**.
 9111 Minimum Acceptable Value: 4 294 967 295

9112 {USHRT_MAX}
 9113 Maximum value for a type **unsigned short**.
 9114 Minimum Acceptable Value: 65 535

9115 {WORD_BIT}
 9116 Number of bits in a type `int`.
 9117 Minimum Acceptable Value: 32

9118 Other Invariant Values

9119 The following symbolic constants shall be defined in <limits.h>:

9120 {NL_ARGMAX}
 9121 Maximum value of *n* in conversion specifications using the "%n\$" sequence in calls to the
 9122 `printf()` and `scanf()` families of functions.
 9123 Minimum Acceptable Value: 9

9124 XSI {NL_LANGMAX}
 9125 Maximum number of bytes in a *LANG* name.
 9126 Minimum Acceptable Value: 14

9127 {NL_MSGMAX}
 9128 Maximum message number.
 9129 Minimum Acceptable Value: 32 767

9130 {NL_SETMAX}
 9131 Maximum set number.
 9132 Minimum Acceptable Value: 255

9133 {NL_TEXTMAX}
 9134 Maximum number of bytes in a message string.
 9135 Minimum Acceptable Value: {_POSIX2_LINE_MAX}

9136 XSI {NZERO}
 9137 Default process priority.
 9138 Minimum Acceptable Value: 20

9139 APPLICATION USAGE

9140 None.

9141 RATIONALE

9142 A request was made to reduce the value of {_POSIX_LINK_MAX} from the value of 8 specified
 9143 for it in the POSIX.1-1990 standard to 2. The standard developers decided to deny this request
 9144 for several reasons:

- 9145 • They wanted to avoid making any changes to the standard that could break conforming
 9146 applications, and the requested change could have that effect.
- 9147 • The use of multiple hard links to a file cannot always be replaced with use of symbolic
 9148 links. Symbolic links are semantically different from hard links in that they associate a
 9149 pathname with another pathname rather than a pathname with a file. This has
 9150 implications for access control, file permanence, and transparency.
- 9151 • The original standard developers had considered the issue of allowing for
 9152 implementations that did not in general support hard links, and decided that this would
 9153 reduce consensus on the standard.

9154 Systems that support historical versions of the development option of the ISO POSIX-2 standard
 9155 retain the name {_POSIX2_RE_DUP_MAX} as an alias for {_POSIX_RE_DUP_MAX}.

9156 {PATH_MAX}
 9157 IEEE PASC Interpretation 1003.1 #15 addressed the inconsistency in the standard with the
 9158 definition of pathname and the description of {PATH_MAX}, allowing application
 9159 developers to allocate either {PATH_MAX} or {PATH_MAX}+1 bytes. The inconsistency has
 9160 been removed by correction to the {PATH_MAX} definition to include the null character.

9161 With this change, applications that previously allocated {PATH_MAX} bytes will continue to
 9162 succeed.

9163 {SYMLINK_MAX}

9164 This symbol refers to space for data that is stored in the file system, as opposed to
 9165 {PATH_MAX} which is the length of a name that can be passed to a function. In some
 9166 existing implementations, the filenames pointed to by symbolic links are stored in the
 9167 inodes of the links, so it is important that {SYMLINK_MAX} not be constrained to be as
 9168 large as {PATH_MAX}.

9169 FUTURE DIRECTIONS

9170 None.

9171 SEE ALSO

9172 [Chapter 7](#) (on page 121), [<stdio.h>](#), [<unistd.h>](#)

9173 [XSH Section 2.2](#) (on page 448), [fpathconf\(\)](#), [sysconf\(\)](#) |

9174 CHANGE HISTORY

9175 First released in Issue 1.

9176 Issue 5

9177 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 9178 Threads Extension.

9179 {FILESIZEBITS} is added for the Large File Summit extensions.

9180 The minimum acceptable values for {INT_MAX}, {INT_MIN}, and {UINT_MAX} are changed to
 9181 make 32-bit values the minimum requirement.

9182 The entry is restructured to improve readability.

9183 Issue 6

9184 The Open Group Corrigendum U033/4 is applied. The wording is made clear for {CHAR_MIN},
 9185 {INT_MIN}, {LONG_MIN}, {SCHAR_MIN}, and {SHRT_MIN} that these are maximum
 9186 acceptable values.

9187 The following new requirements on POSIX implementations derive from alignment with the
 9188 Single UNIX Specification:

- 9189 • The minimum value for {CHILD_MAX} is 25. This is a FIPS requirement.
- 9190 • The minimum value for {OPEN_MAX} is 20. This is a FIPS requirement.
- 9191 • The minimum value for {NGROUPS_MAX} is 8. This is also a FIPS requirement.

9192 Symbolic constants are added for {_POSIX_SYMLINK_MAX}, {_POSIX_SYMLOOP_MAX},
 9193 {_POSIX_RE_DUP_MAX}, {RE_DUP_MAX}, {SYMLOOP_MAX}, and {SYMLINK_MAX}.

9194 The following values are added for alignment with IEEE Std 1003.1d-1999:

9195 {_POSIX_SS_REPL_MAX}
 9196 {SS_REPL_MAX}
 9197 {POSIX_ALLOC_SIZE_MIN}
 9198 {POSIX_REC_INCR_XFER_SIZE}
 9199 {POSIX_REC_MAX_XFER_SIZE}
 9200 {POSIX_REC_MIN_XFER_SIZE}
 9201 {POSIX_REC_XFER_ALIGN}

9202 Reference to CLOCK_MONOTONIC is added in the description of {_POSIX_CLOCKRES_MIN}
 9203 for alignment with IEEE Std 1003.1j-2000.

9204 The constants {LLONG_MIN}, {LLONG_MAX}, and {ULLONG_MAX} are added for alignment

9205 with the ISO/IEC 9899:1999 standard.

9206 The following values are added for alignment with IEEE Std 1003.1q-2000:

```
9207 { _POSIX_TRACE_EVENT_NAME_MAX }
9208 { _POSIX_TRACE_NAME_MAX }
9209 { _POSIX_TRACE_SYS_MAX }
9210 { _POSIX_TRACE_USER_EVENT_MAX }
9211 { TRACE_EVENT_NAME_MAX }
9212 { TRACE_NAME_MAX }
9213 { TRACE_SYS_MAX }
9214 { TRACE_USER_EVENT_MAX }
```

9215 The new limits {_XOPEN_NAME_MAX} and {_XOPEN_PATH_MAX} are added as minimum
9216 values for {PATH_MAX} and {NAME_MAX} limits on XSI-conformant systems.

9217 The LEGACY symbols {PASS_MAX} and {TMP_MAX} are removed.

9218 The values for the limits {CHAR_BIT}, {SCHAR_MAX}, and {UCHAR_MAX} are now required
9219 to be 8, +127, and 255, respectively.

9220 The value for the limit {CHAR_MAX} is now {UCHAR_MAX} or {SCHAR_MAX}.

9221 The value for the limit {CHAR_MIN} is now {SCHAR_MIN} or zero.

9222 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/10 is applied, correcting the value of
9223 {_POSIX_CHILD_MAX} from 6 to 25. This is for FIPS 151-2 alignment.

9224 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/19 is applied, updating the values for
9225 {INT_MAX}, {UINT_MAX}, and {INT_MIN} to be CX extensions over the ISO C standard, and
9226 correcting {WORD_BIT} from 16 to 32.

9227 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/20 is applied, removing
9228 {CHARCLASS_NAME_MAX} from the "Other Invariant Values" section (it also occurs under
9229 "Runtime Inceasable Values").

9230 Issue 7

9231 SD5-XBD-ERN-36 is applied, changing the description of {RE_DUP_MAX}.

9232 {NL_NMAX} is removed; it should have been removed in Issue 6.

9233 The Trace option values are marked obsolescent.

9234 The {ATEXIT_MAX}, {LONG_BIT}, {NL_MSGMAX}, {NL_SETMAX}, {NL_TEXTMAX}, and
9235 {WORD_BIT} values are moved from the XSI option to the Base.

9236 The AIO_* and _POSIX_AIO_* values are moved from the Asynchronous Input and Output
9237 option to the Base.

9238 The {_POSIX_RT_SIG_MAX}, {_POSIX_SIGQUEUE_MAX}, {RTSIG_MAX}, and
9239 {SIGQUEUE_MAX} values are moved from the Realtime Signals Extension option to the Base.

9240 Functionality relating to the Threads and Timers options is moved to the Base.

9241 This reference page is clarified with respect to macros and symbolic constants. +

9242 **NAME**

9243 locale.h — category macros

9244 **SYNOPSIS**

9245 #include <locale.h>

9246 **DESCRIPTION**

9247 CX Some of the functionality described on this reference page extends the ISO C standard. Any
 9248 conflict between the requirements described here and the ISO C standard is unintentional. This
 9249 volume of POSIX.1-200x defers to the ISO C standard.

9250 The <locale.h> header shall provide a definition for **lconv** structure, which shall include at least
 9251 the following members. (See the definitions of *LC_MONETARY* in [Section 7.3.3](#) (on page 139)
 9252 and [Section 7.3.4](#) (on page 143).)

```

9253 char    *currency_symbol
9254 char    *decimal_point
9255 char    frac_digits
9256 char    *grouping
9257 char    *int_curr_symbol
9258 char    int_frac_digits
9259 char    int_n_cs_precedes
9260 char    int_n_sep_by_space
9261 char    int_n_sign_posn
9262 char    int_p_cs_precedes
9263 char    int_p_sep_by_space
9264 char    int_p_sign_posn
9265 char    *mon_decimal_point
9266 char    *mon_grouping
9267 char    *mon_thousands_sep
9268 char    *negative_sign
9269 char    n_cs_precedes
9270 char    n_sep_by_space
9271 char    n_sign_posn
9272 char    *positive_sign
9273 char    p_cs_precedes
9274 char    p_sep_by_space
9275 char    p_sign_posn
9276 char    *thousands_sep
  
```

9277 The <locale.h> header shall define NULL (as defined in <stddef.h>) and at least the following
 9278 as macros:

```

9279 LC_ALL
9280 LC_COLLATE
9281 LC_CTYPE
9282 CX LC_MESSAGES
9283 LC_MONETARY
9284 LC_NUMERIC
9285 LC_TIME
  
```

9286 which shall expand to integer constant expressions with distinct values for use as the first
 9287 argument to the *setlocale()* function.

9288 Implementations may add additional masks using the form *LC_** and an uppercase letter.

9289 CX The <locale.h> header shall contain at least the following macros representing bitmasks for use
 9290 with the *newlocale()* function for each supported locale category:

```

9291 LC_COLLATE_MASK
9292 LC_CTYPE_MASK
9293 LC_MESSAGES_MASK
9294 LC_MONETARY_MASK
9295 LC_NUMERIC_MASK
9296 LC_TIME_MASK

```

9297 Implementations may add additional masks using the form `LC_*_MASK`.

9298 In addition, a macro to set the bits for all categories set shall be defined:

```

9299 LC_ALL_MASK

```

9300 The <locale.h> header shall define `LC_GLOBAL_LOCALE`, a special locale object descriptor
 9301 used by the *uselocale()* function.

9302 The <locale.h> header shall provide a definition for a type `locale_t` representing a locale object.

9303 The following shall be declared as functions and may also be defined as macros. Function
 9304 prototypes shall be provided for use with ISO C standard compilers.

```

9305 struct lconv *localeconv(void);
9306 CX locale_t duplocale(locale_t);
9307 void freelocale(locale_t);
9308 locale_t newlocale(int, const char *, locale_t);
9309 char *setlocale(int, const char *);
9310 CX locale_t uselocale (locale_t);

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO[Chapter 8](#) (on page 159), [<stddef.h>](#)XSH [localeconv\(\)](#), [setlocale\(\)](#)**CHANGE HISTORY**

First released in Issue 3.

Included for alignment with the ISO C standard.

Issue 6

9324 The `lconv` structure is expanded with new members (`int_n_cs_precedes`, `int_n_sep_by_space`,
 9325 `int_n_sign_posn`, `int_p_cs_precedes`, `int_p_sep_by_space`, and `int_p_sign_posn`) for alignment
 9326 with the ISO/IEC 9899:1999 standard.

Extensions beyond the ISO C standard are marked.

9328

Issue 7

9329

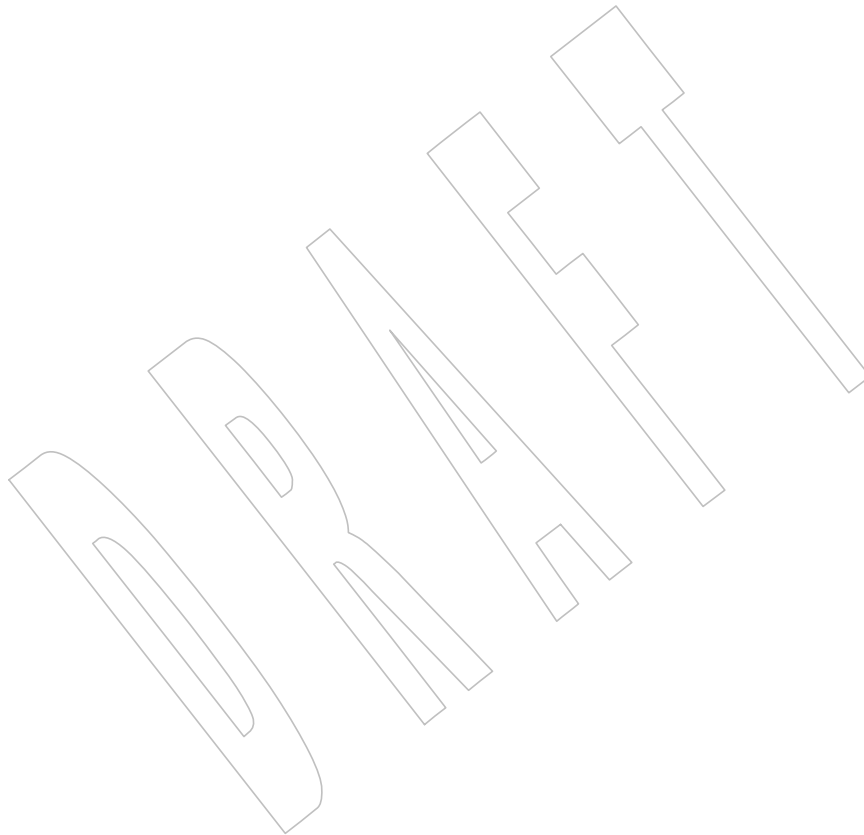
The *duplocale()*, *freelocale()*, *newlocale()*, and *uselocale()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

9330

9331

This reference page is clarified with respect to macros and symbolic constants.

+



9332 **NAME**9333 `math.h` — mathematical declarations9334 **SYNOPSIS**9335 `#include <math.h>`9336 **DESCRIPTION**

9337 CX Some of the functionality described on this reference page extends the ISO C standard.
 9338 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 9339 enable the visibility of these symbols in this header.

9340 The <math.h> header shall include definitions for at least the following types:

9341 **float_t** A real-floating type at least as wide as **float**.9342 **double_t** A real-floating type at least as wide as **double**, and at least as wide as **float_t**.

9343 If FLT_EVAL_METHOD equals 0, **float_t** and **double_t** shall be **float** and **double**, respectively; if
 9344 FLT_EVAL_METHOD equals 1, they shall both be **double**; if FLT_EVAL_METHOD equals 2,
 9345 they shall both be **long double**; for other values of FLT_EVAL_METHOD, they are otherwise
 9346 implementation-defined.

9347 The <math.h> header shall define the following macros, where real-floating indicates that the
 9348 argument shall be an expression of real-floating type:

```
9349 int fpclassify(real-floating x);
9350 int isfinite(real-floating x);
9351 int isinf(real-floating x);
9352 int isnan(real-floating x);
9353 int isnormal(real-floating x);
9354 int signbit(real-floating x);
9355 int isgreater(real-floating x, real-floating y);
9356 int isgreaterequal(real-floating x, real-floating y);
9357 int isless(real-floating x, real-floating y);
9358 int islessequal(real-floating x, real-floating y);
9359 int islessgreater(real-floating x, real-floating y);
9360 int isunordered(real-floating x, real-floating y);
```

9361 The <math.h> header shall define the following symbolic constants. The values shall have type
 9362 **double** and shall be accurate within the precision of the **double** type.

9363	XSI	M_E	Value of e
9364	XSI	M_LOG2E	Value of $\log_2 e$
9365	XSI	M_LOG10E	Value of $\log_{10} e$
9366	XSI	M_LN2	Value of $\log_e 2$
9367	XSI	M_LN10	Value of $\log_e 10$
9368	XSI	M_PI	Value of π
9369	XSI	M_PI_2	Value of $\pi/2$
9370	XSI	M_PI_4	Value of $\pi/4$
9371	XSI	M_1_PI	Value of $1/\pi$

9372	XSI	M_2_PI	Value of $2/\pi$		
9373	XSI	M_2_SQRTPI	Value of $2/\sqrt{\pi}$		
9374	XSI	M_SQRT2	Value of $\sqrt{2}$		
9375	XSI	M_SQRT1_2	Value of $1/\sqrt{2}$		
9378		The <math.h> header shall define the following symbolic constant:			
9379	OB XSI	MAXFLOAT	Same value as FLT_MAX in <float.h>.		
9380		The <math.h> header shall define the following macros:		+	
9381		HUGE_VAL	A positive double constant expression, not necessarily representable as a		
9382			float . Used as an error value returned by the mathematics library.		
9383			HUGE_VAL evaluates to +infinity on systems supporting IEEE Std 754-1985.		
9384		HUGE_VALF	A positive float constant expression. Used as an error value returned by the		
9385			mathematics library. HUGE_VALF evaluates to +infinity on systems		
9386			supporting IEEE Std 754-1985.		
9387		HUGE_VALL	A positive long double constant expression. Used as an error value returned		
9388			by the mathematics library. HUGE_VALL evaluates to +infinity on systems		
9389			supporting IEEE Std 754-1985.		
9390		INFINITY	A constant expression of type float representing positive or unsigned infinity,		
9391			if available; else a positive constant of type float that overflows at translation		
9392			time.		
9393		NAN	A constant expression of type float representing a quiet NaN. This macro is		
9394			only defined if the implementation supports quiet NaNs for the float type.		
9395		The following macros shall be defined for number classification. They represent the mutually-			
9396		exclusive kinds of floating-point values. They expand to integer constant expressions with			
9397		distinct values. Additional implementation-defined floating-point classifications, with macro			
9398		definitions beginning with FP_ and an uppercase letter, may also be specified by the			
9399		implementation.			
9400		FP_INFINITE			
9401		FP_NAN			
9402		FP_NORMAL			
9403		FP_SUBNORMAL			
9404		FP_ZERO			
9405		The following optional macros indicate whether the <i>fma()</i> family of functions are fast compared			
9406		with direct code:			
9407		FP_FAST_FMA			
9408		FP_FAST_FMAF			
9409		FP_FAST_FMAL			
9410		If defined, the FP_FAST_FMA macro shall expand to the integer constant 1 and shall indicate			
9411		that the <i>fma()</i> function generally executes about as fast as, or faster than, a multiply and an add			
9412		of double operands. If undefined, the speed of execution is unspecified. The other macros have			
9413		the equivalent meaning for the float and long double versions.			
9414		The following macros shall expand to integer constant expressions whose values are returned by			
9415		<i>ilogb(x)</i> if <i>x</i> is zero or NaN, respectively. The value of FP_ILOGB0 shall be either { INT_MIN } or			
9416		-{ INT_MAX }. The value of FP_ILOGBNAN shall be either { INT_MAX } or { INT_MIN }.			

9417 FP_ILOGB0
 9418 FP_ILOGBNAN

9419 The following macros shall expand to the integer constants 1 and 2, respectively;

9420 MATH_ERRNO
 9421 MATH_ERREXCEPT

9422 The following macro shall expand to an expression that has type **int** and the value
 9423 MATH_ERRNO, MATH_ERREXCEPT, or the bitwise-inclusive OR of both:

9424 math_errhandling

9425 The value of `math_errhandling` is constant for the duration of the program. It is unspecified
 9426 whether `math_errhandling` is a macro or an identifier with external linkage. If a macro definition
 9427 is suppressed or a program defines an identifier with the name `math_errhandling`, the behavior
 9428 is undefined. If the expression `(math_errhandling & MATH_ERREXCEPT)` can be non-zero, the
 9429 implementation shall define the macros `FE_DIVBYZERO`, `FE_INVALID`, and `FE_OVERFLOW` in
 9430 <fenv.h>.

9431 The following shall be declared as functions and may also be defined as macros. Function
 9432 prototypes shall be provided.

```

9433 double      acos(double);
9434 float       acosf(float);
9435 double      acosh(double);
9436 float       acoshf(float);
9437 long double acoshl(long double);
9438 long double acosl(long double);
9439 double      asin(double);
9440 float       asinf(float);
9441 double      asinh(double);
9442 float       asinhf(float);
9443 long double asinhl(long double);
9444 long double asinl(long double);
9445 double      atan(double);
9446 double      atan2(double, double);
9447 float       atan2f(float, float);
9448 long double atan2l(long double, long double);
9449 float       atanf(float);
9450 double      atanh(double);
9451 float       atanhf(float);
9452 long double atanh1(long double);
9453 long double atanl(long double);
9454 double      cbrt(double);
9455 float       cbrtf(float);
9456 long double cbrtl(long double);
9457 double      ceil(double);
9458 float       ceilf(float);
9459 long double ceill(long double);
9460 double      copysign(double, double);
9461 float       copysignf(float, float);
9462 long double copysignl(long double, long double);
9463 double      cos(double);
9464 float       cosf(float);
9465 double      cosh(double);
  
```



```

9466     float      coshf(float);
9467     long double coshl(long double);
9468     long double cosl(long double);
9469     double      erf(double);
9470     double      erfc(double);
9471     float       erfcf(float);
9472     long double erfcl(long double);
9473     float       erff(float);
9474     long double erfl(long double);
9475     double      exp(double);
9476     double      exp2(double);
9477     float       exp2f(float);
9478     long double exp2l(long double);
9479     float       expf(float);
9480     long double expl(long double);
9481     double      expml(double);
9482     float       expmlf(float);
9483     long double expmll(long double);
9484     double      fabs(double);
9485     float       fabsf(float);
9486     long double fabsl(long double);
9487     double      fdim(double, double);
9488     float       fdimf(float, float);
9489     long double fdiml(long double, long double);
9490     double      floor(double);
9491     float       floorf(float);
9492     long double floorl(long double);
9493     double      fma(double, double, double);
9494     float       fmaf(float, float, float);
9495     long double fmal(long double, long double, long double);
9496     double      fmax(double, double);
9497     float       fmaxf(float, float);
9498     long double fmaxl(long double, long double);
9499     double      fmin(double, double);
9500     float       fminf(float, float);
9501     long double fminl(long double, long double);
9502     double      fmod(double, double);
9503     float       fmodf(float, float);
9504     long double fmodl(long double, long double);
9505     double      frexp(double, int *);
9506     float       frexpf(float, int *);
9507     long double frexpl(long double, int *);
9508     double      hypot(double, double);
9509     float       hypotf(float, float);
9510     long double hypotl(long double, long double);
9511     int         ilogb(double);
9512     int         ilogbf(float);
9513     int         ilogbl(long double);
9514     double      j0(double);
9515     double      j1(double);
9516     double      jn(int, double);
9517     double      ldexp(double, int);
9518     float       ldexpf(float, int);
9519     long double ldexpl(long double, int);

```

```

9520     double      lgamma(double);
9521     float       lgammaf(float);
9522     long double  lgammal(long double);
9523     long long    llrint(double);
9524     long long    llrintf(float);
9525     long long    llrintl(long double);
9526     long long    llround(double);
9527     long long    llroundf(float);
9528     long long    llroundl(long double);
9529     double      log(double);
9530     double      log10(double);
9531     float       log10f(float);
9532     long double  log10l(long double);
9533     double      log1p(double);
9534     float       log1pf(float);
9535     long double  log1pl(long double);
9536     double      log2(double);
9537     float       log2f(float);
9538     long double  log2l(long double);
9539     double      logb(double);
9540     float       logbf(float);
9541     long double  logbl(long double);
9542     float       logf(float);
9543     long double  logl(long double);
9544     long        lrint(double);
9545     long        lrintf(float);
9546     long        lrintl(long double);
9547     long        lround(double);
9548     long        lroundf(float);
9549     long        lroundl(long double);
9550     double      modf(double, double *);
9551     float       modff(float, float *);
9552     long double  modfl(long double, long double *);
9553     double      nan(const char *);
9554     float       nanf(const char *);
9555     long double  nanl(const char *);
9556     double      nearbyint(double);
9557     float       nearbyintf(float);
9558     long double  nearbyintl(long double);
9559     double      nextafter(double, double);
9560     float       nextafterf(float, float);
9561     long double  nextafterl(long double, long double);
9562     double      nexttoward(double, long double);
9563     float       nexttowardf(float, long double);
9564     long double  nexttowardl(long double, long double);
9565     double      pow(double, double);
9566     float       powf(float, float);
9567     long double  powl(long double, long double);
9568     double      remainder(double, double);
9569     float       remainderf(float, float);
9570     long double  remainderl(long double, long double);
9571     double      remquo(double, double, int *);
9572     float       remquof(float, float, int *);
9573     long double  remquol(long double, long double, int *);

```

```

9574     double    rint(double);
9575     float     rintf(float);
9576     long double rintl(long double);
9577     double    round(double);
9578     float     roundf(float);
9579     long double roundl(long double);
9580     double    scalbln(double, long);
9581     float     scalblnf(float, long);
9582     long double scalblnl(long double, long);
9583     double    scalbn(double, int);
9584     float     scalbnf(float, int);
9585     long double scalbnl(long double, int);
9586     double    sin(double);
9587     float     sinf(float);
9588     double    sinh(double);
9589     float     sinhf(float);
9590     long double sinhl(long double);
9591     long double sinl(long double);
9592     double    sqrt(double);
9593     float     sqrtf(float);
9594     long double sqrtl(long double);
9595     double    tan(double);
9596     float     tanf(float);
9597     double    tanh(double);
9598     float     tanhf(float);
9599     long double tanhl(long double);
9600     long double tanl(long double);
9601     double    tgamma(double);
9602     float     tgammaf(float);
9603     long double tgamma1(long double);
9604     double    trunc(double);
9605     float     truncf(float);
9606     long double trunc1(long double);
9607     XSI     double    y0(double);
9608     double    y1(double);
9609     double    yn(int, double);

```

9610 The following external variable shall be defined:

```

9611     XSI     extern int signgam;

```

9612 The behavior of each of the functions defined in <math.h> is specified in the System Interfaces
9613 volume of POSIX.1-200x for all representable values of its input arguments, except where stated
9614 otherwise. Each function shall execute as if it were a single operation without generating any
9615 externally visible exceptional conditions.

APPLICATION USAGE

The FP_CONTRACT pragma can be used to allow (if the state is on) or disallow (if the state is off) the implementation to contract expressions. Each pragma can occur either outside external declarations or preceding all explicit declarations and statements inside a compound statement. When outside external declarations, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered, or until the end of the translation unit. When inside a compound statement, the pragma takes effect from its occurrence until another FP_CONTRACT pragma is encountered (including within a nested compound statement), or until the end of the compound statement; at the end of a compound statement the state for the pragma is restored to its condition just before the compound statement. If this pragma is used in any other context, the behavior is undefined. The default state (on or off) for the pragma is implementation-defined.

Applications should use FLT_MAX as defined in <float.h> instead of the obsolescent MAXFLOAT.

RATIONALE

Before the ISO/IEC 9899:1999 standard, the math library was defined only for the floating type **double**. All the names formed by appending 'f' or 'l' to a name in <math.h> were reserved to allow for the definition of **float** and **long double** libraries; and the ISO/IEC 9899:1999 standard provides for all three versions of math functions.

The functions *ecvt()*, *fcvt()*, and *gcvt()* have been dropped from the ISO C standard since their capability is available through *sprintf()*.

FUTURE DIRECTIONS

None.

SEE ALSO

<float.h>, <stddef.h>, <sys/types.h>

XSH Section 2.2 (on page 448), *acos()*, *acosh()*, *asin()*, *atan()*, *atan2()*, *cbrt()*, *ceil()*, *cos()*, *cosh()*, *erf()*, *exp()*, *expm1()*, *fabs()*, *floor()*, *fmod()*, *frexp()*, *hypot()*, *ilogb()*, *isnan()*, *j0()*, *ldexp()*, *lgamma()*, *log()*, *log10()*, *log1p()*, *logb()*, *modf()*, *nextafter()*, *pow()*, *remainder()*, *rint()*, *scalbln()*, *sin()*, *sinh()*, *sqrt()*, *tan()*, *tanh()*, *y0()*

CHANGE HISTORY

First released in Issue 1.

Issue 6

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/21 is applied, making it clear that the meaning of the FP_FAST_FMA macro is unspecified if the macro is undefined.

Issue 7

ISO C TC2 #47 (SD5-XBD-ERN-52) is applied, clarifying the wording of the FP_FAST_FMA macro.

The MAXFLOAT constant is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

9655 **NAME**

9656 monetary.h — monetary types

9657 **SYNOPSIS**

9658 #include <monetary.h>

9659 **DESCRIPTION**

9660 The <monetary.h> header shall define the following types:

9661 **size_t** As described in <stddef.h>.9662 **ssize_t** As described in <sys/types.h>.9663 The following shall be declared as functions and may also be defined as macros. Function
9664 prototypes shall be provided for use with ISO C standard compilers.9665 `ssize_t strfmon(char *restrict, size_t, const char *restrict, ...);`9666 `ssize_t strfmon_l(char *restrict, size_t, locale_t,`9667 `const char *restrict, ...);`9668 **APPLICATION USAGE**

9669 None.

9670 **RATIONALE**

9671 None.

9672 **FUTURE DIRECTIONS**

9673 None.

9674 **SEE ALSO**

9675 <stddef.h>, <sys/types.h>

9676 XSH *strfmon()*9677 **CHANGE HISTORY**

9678 First released in Issue 4.

9679 **Issue 6**9680 The **restrict** keyword is added to the prototype for *strfmon()*.9681 **Issue 7**

9682 The <monetary.h> header is moved from the XSI option to the Base.

9683 The *strmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API
9684 Set Part 4.

9685 **NAME**9686 mqqueue.h — message queues (**REALTIME**)9687 **SYNOPSIS**

9688 MSG #include <mqqueue.h>

9689 **DESCRIPTION**9690 The <mqqueue.h> header shall define the **mqd_t** type, which is used for message queue
9691 descriptors. This is not an array type.9692 The <mqqueue.h> header shall define the **sigevent** structure (as described in <signal.h>) and the
9693 **mq_attr** structure, which is used in getting and setting the attributes of a message queue.
9694 Attributes are initially set when the message queue is created. An **mq_attr** structure shall have at
9695 least the following fields:9696 long mq_flags Message queue flags.
9697 long mq_maxmsg Maximum number of messages.
9698 long mq_msgsize Maximum message size.
9699 long mq_curmsgs Number of messages currently queued.9700 The following shall be declared as functions and may also be defined as macros. Function
9701 prototypes shall be provided.9702 int mq_close(mqd_t);
9703 int mq_getattr(mqd_t, struct mq_attr *);
9704 int mq_notify(mqd_t, const struct sigevent *);
9705 mqd_t mq_open(const char *, int, ...);
9706 ssize_t mq_receive(mqd_t, char *, size_t, unsigned *);
9707 int mq_send(mqd_t, const char *, size_t, unsigned);
9708 int mq_setattr(mqd_t, const struct mq_attr *restrict,
9709 struct mq_attr *restrict);
9710 ssize_t mq_timedreceive(mqd_t, char *restrict, size_t,
9711 unsigned *restrict, const struct timespec *restrict);
9712 int mq_timedsend(mqd_t, const char *, size_t, unsigned,
9713 const struct timespec *);
9714 int mq_unlink(const char *);9715 Inclusion of the <mqqueue.h> header may make visible symbols defined in the headers
9716 <fcntl.h>, <signal.h>, <sys/types.h>, and <time.h>.9717 **APPLICATION USAGE**

9718 None.

9719 **RATIONALE**

9720 None.

9721 **FUTURE DIRECTIONS**

9722 None.

9723 **SEE ALSO**

9724 <fcntl.h>, <signal.h>, <sys/types.h>, <time.h>

9725 XSH *mq_close()*, *mq_getattr()*, *mq_notify()*, *mq_open()*, *mq_receive()*, *mq_send()*, *mq_setattr()*,
9726 *mq_unlink()*

9727

CHANGE HISTORY

9728

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

9729

Issue 6

9730

The <mqqueue.h> header is marked as part of the Message Passing option.

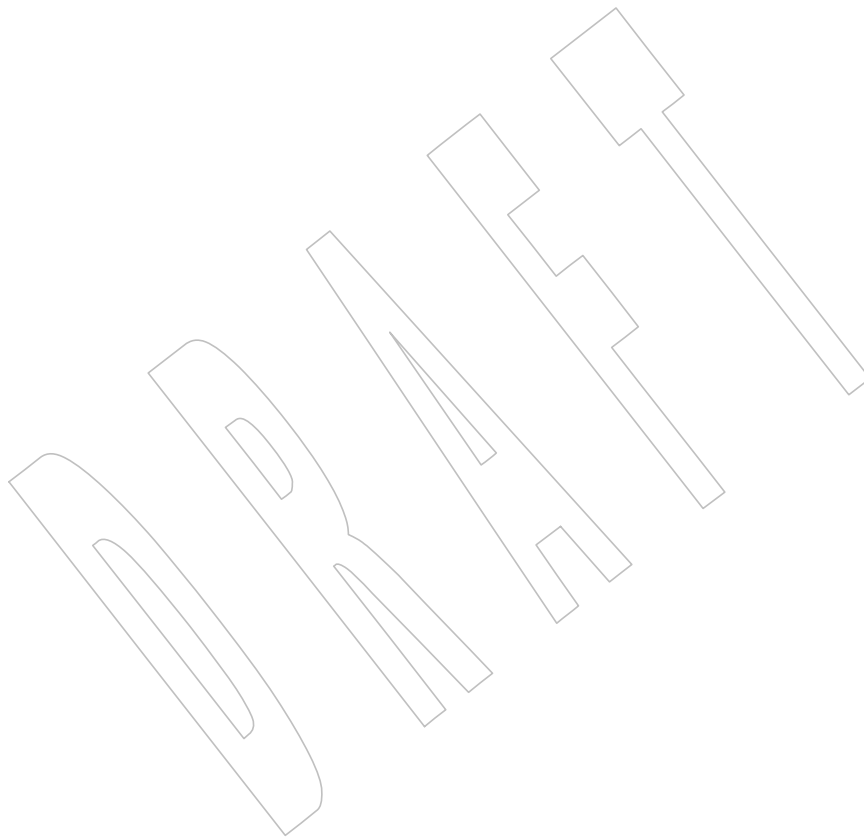
9731

The *mq_timedreceive()* and *mq_timedsend()* functions are added for alignment with IEEE Std 1003.1d-1999.

9732

9733

The **restrict** keyword is added to the prototypes for *mq_setattr()* and *mq_timedreceive()*.



9734 **NAME**
 9735 ndbm.h — definitions for ndbm database operations

9736 **SYNOPSIS**
 9737 XSI `#include <ndbm.h>`

9738 **DESCRIPTION**
 9739 The <ndbm.h> header shall define the **datum** type as a structure that includes at least the
 9740 following members:

9741 `void *dptr` A pointer to the application's data.
 9742 `size_t dsize` The size of the object pointed to by *dptr*.

9743 The **size_t** type shall be defined as described in <stddef.h>.

9744 The <ndbm.h> header shall define the **DBM** type.

9745 The <ndbm.h> header shall define the following symbolic constants as possible values for the
 9746 *store_mode* argument to *dbm_store()*:

9747 **DBM_INSERT** Insertion of new entries only.

9748 **DBM_REPLACE** Allow replacing existing entries.

9749 The following shall be declared as functions and may also be defined as macros. Function
 9750 prototypes shall be provided.

9751 `int dbm_clearerr(DBM *);`
 9752 `void dbm_close(DBM *);`
 9753 `int dbm_delete(DBM *, datum);`
 9754 `int dbm_error(DBM *);`
 9755 `datum dbm_fetch(DBM *, datum);`
 9756 `datum dbm_firstkey(DBM *);`
 9757 `datum dbm_nextkey(DBM *);`
 9758 `DBM *dbm_open(const char *, int, mode_t);`
 9759 `int dbm_store(DBM *, datum, datum, int);`

9760 The **mode_t** type shall be defined through **typedef** as described in <sys/types.h>.

9761 APPLICATION USAGE

9762 None.

9763 RATIONALE

9764 None.

9765 FUTURE DIRECTIONS

9766 None.

9767 SEE ALSO

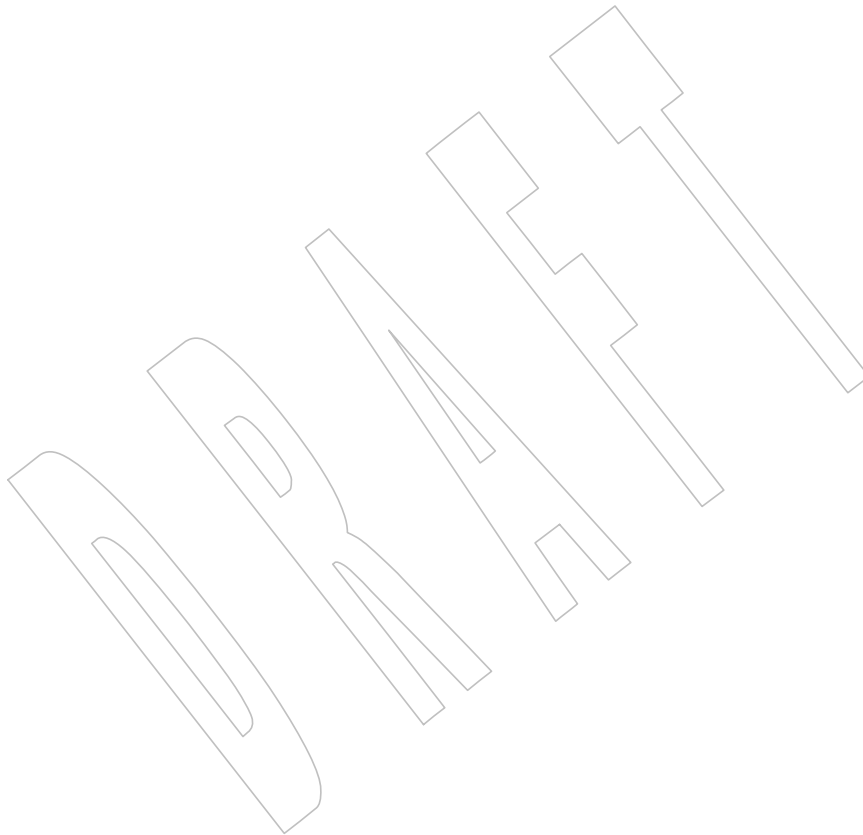
9768 [<stddef.h>](#), [<sys/types.h>](#)

9769 XSH [dbm_clearerr\(\)](#)

9770 CHANGE HISTORY

9771 First released in Issue 4, Version 2.

- 9772 **Issue 5**
9773 References to the definitions of **size_t** and **mode_t** are added to the DESCRIPTION.
- 9774 **Issue 7**
9775 This reference page is clarified with respect to macros and symbolic constants. +



9776 **NAME**

9777 net/if.h — sockets local interfaces

9778 **SYNOPSIS**

9779 #include <net/if.h>

9780 **DESCRIPTION**9781 The <net/if.h> header shall define the **if_nameindex** structure that includes at least the
9782 following members:9783 unsigned if_index Numeric index of the interface.
9784 char *if_name Null-terminated name of the interface.9785 The <net/if.h> header shall define the following symbolic constant for the length of a buffer |
9786 containing an interface name (including the terminating NULL character):

9787 IF_NAMESIZE Interface name length.

9788 The following shall be declared as functions and may also be defined as macros. Function
9789 prototypes shall be provided.9790 unsigned if_nametoindex(const char *);
9791 char *if_indextoname(unsigned, char *);
9792 struct if_nameindex *if_nameindex(void);
9793 void if_freenameindex(struct if_nameindex *);9794 **APPLICATION USAGE**

9795 None.

9796 **RATIONALE**

9797 None.

9798 **FUTURE DIRECTIONS**

9799 None.

9800 **SEE ALSO**9801 XSH [if_freenameindex\(\)](#), [if_indextoname\(\)](#), [if_nameindex\(\)](#), [if_nametoindex\(\)](#) |9802 **CHANGE HISTORY**

9803 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9804 **Issue 7**

9805 This reference page is clarified with respect to macros and symbolic constants. +

9806 **NAME**
 9807 netdb.h — definitions for network database operations

9808 **SYNOPSIS**
 9809 #include <netdb.h>

9810 **DESCRIPTION**
 9811 The <netdb.h> header may define the **in_port_t** type and the **in_addr_t** type as described in
 9812 <netinet/in.h>.

9813 The <netdb.h> header shall define the **hostent** structure that includes at least the following
 9814 members:

9815	char	*h_name	Official name of the host.
9816	char	**h_aliases	A pointer to an array of pointers to 9817 alternative host names, terminated by a 9818 null pointer.
9819	int	h_addrtype	Address type.
9820	int	h_length	The length, in bytes, of the address.
9821	char	**h_addr_list	A pointer to an array of pointers to network 9822 addresses (in network byte order) for the host, 9823 terminated by a null pointer.

9824 The <netdb.h> header shall define the **netent** structure that includes at least the following
 9825 members:

9826	char	*n_name	Official, fully-qualified (including the 9827 domain) name of the host.
9828	char	**n_aliases	A pointer to an array of pointers to 9829 alternative network names, terminated by a 9830 null pointer.
9831	int	n_addrtype	The address type of the network.
9832	uint32_t	n_net	The network number, in host byte order.

9833 The **uint32_t** type shall be defined as described in <inttypes.h>.

9834 The <netdb.h> header shall define the **protoent** structure that includes at least the following
 9835 members:

9836	char	*p_name	Official name of the protocol.
9837	char	**p_aliases	A pointer to an array of pointers to 9838 alternative protocol names, terminated by 9839 a null pointer.
9840	int	p_proto	The protocol number.

9841 The <netdb.h> header shall define the **servent** structure that includes at least the following
 9842 members:

9843	char	*s_name	Official name of the service.
9844	char	**s_aliases	A pointer to an array of pointers to 9845 alternative service names, terminated by 9846 a null pointer.
9847	int	s_port	A value which, when converted to uint16_t , 9848 yields the port number in network byte order 9849 at which the service resides.
9850	char	*s_proto	The name of the protocol to use when 9851 contacting the service.

9852 The <netdb.h> header shall define the IPPORT_RESERVED symbolic constant with the value of
9853 the highest reserved Internet port number.

9854 Address Information Structure

9855 The <netdb.h> header shall define the **addrinfo** structure that includes at least the following
9856 members:

9857	int	ai_flags	Input flags.
9858	int	ai_family	Address family of socket.
9859	int	ai_socktype	Socket type.
9860	int	ai_protocol	Protocol of socket.
9861	socklen_t	ai_addrlen	Length of socket address.
9862	struct sockaddr	*ai_addr	Socket address of socket.
9863	char	*ai_canonname	Canonical name of service location.
9864	struct addrinfo	*ai_next	Pointer to next in list.

9865 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
9866 distinct integer constants for use in the *flags* field of the **addrinfo** structure:

9867	AI_PASSIVE	Socket address is intended for <i>bind()</i> .
9868	AI_CANONNAME	Request for canonical name.
9869		
9870	AI_NUMERICHOST	Return numeric host address as name.
9871		
9872	AI_NUMERICSERV	Inhibit service name resolution.
9873		
9874	AI_V4MAPPED	If no IPv6 addresses are found, query for IPv4 addresses and return them to 9875 the caller as IPv4-mapped IPv6 addresses.
9876	AI_ALL	Query for both IPv4 and IPv6 addresses.
9877	AI_ADDRCONFIG	Query for IPv4 addresses only when an IPv4 address is configured; query for 9878 IPv6 addresses only when an IPv6 address is configured. 9879

9880 The <netdb.h> header shall define the following symbolic constants that evaluate to bitwise-
9881 distinct integer constants for use in the *flags* argument to *getnameinfo()*:

9882	NI_NOFQDN	Only the nodename portion of the FQDN is returned for local hosts.
9883	NI_NUMERICHOST	The numeric form of the node's address is returned instead of its name.
9884		
9885	NI_NAMEREQD	Return an error if the node's name cannot be located in the database.
9886	NI_NUMERICSERV	The numeric form of the service address is returned instead of its name.
9887		
9888	NI_NUMERICSERVICE	
9889	NI_NUMERICSERVICE	For IPv6 addresses, the numeric form of the scope identifier is returned 9890 instead of its name.
9891	NI_DGRAM	Indicates that the service is a datagram service (SOCK_DGRAM).

9892 **Address Information Errors**

9893 The <netdb.h> header shall define the following symbolic constants for use as error values for
 9894 *getaddrinfo()* and *getnameinfo()*. The values shall be suitable for use in #if preprocessing
 9895 directives.

9896 EAI_AGAIN The name could not be resolved at this time. Future attempts may succeed.

9897 EAI_BADFLAGS The flags had an invalid value.

9898 EAI_FAIL A non-recoverable error occurred.

9899 EAI_FAMILY The address family was not recognized or the address length was invalid for
 9900 the specified family.

9901 EAI_MEMORY There was a memory allocation failure.

9902 EAI_NONAME The name does not resolve for the supplied parameters.

9903 NI_NAMEREQD is set and the host's name cannot be located, or both
 9904 *nodename* and *servname* were null.

9905 EAI_SERVICE The service passed was not recognized for the specified socket type.

9906 EAI_SOCKTYPE The intended socket type was not recognized.

9907 EAI_SYSTEM A system error occurred. The error code can be found in *errno*.

9908 EAI_OVERFLOW
 9909 An argument buffer overflowed.

9910 The following shall be declared as functions and may also be defined as macros. Function
 9911 prototypes shall be provided.

```

9912 void          endhostent(void);
9913 void          endnetent(void);
9914 void          endprotoent(void);
9915 void          endservent(void);
9916 void          freeaddrinfo(struct addrinfo *);
9917 const char    *gai_strerror(int);
9918 int           getaddrinfo(const char *restrict, const char *restrict,
9919                          const struct addrinfo *restrict,
9920                          struct addrinfo **restrict);
9921 struct hostent *gethostent(void);
9922 int           getnameinfo(const struct sockaddr *restrict, socklen_t,
9923                          char *restrict, socklen_t, char *restrict,
9924                          socklen_t, int);
9925 struct netent *getnetbyaddr(uint32_t, int);
9926 struct netent *getnetbyname(const char *);
9927 struct netent *getnetent(void);
9928 struct protoent *getprotobyname(const char *);
9929 struct protoent *getprotobynumber(int);
9930 struct protoent *getprotoent(void);
9931 struct servent *getservbyname(const char *, const char *);
9932 struct servent *getservbyport(int, const char *);
9933 struct servent *getservent(void);
9934 void          sethostent(int);
9935 void          setnetent(int);
9936 void          setprotoent(int);
9937 void          setservent(int);

```

9938 The type **socklen_t** shall be defined through **typedef** as described in <sys/socket.h>.

9939 Inclusion of the <netdb.h> header may also make visible all symbols from <netinet/in.h>,
9940 <sys/socket.h>, and <inttypes.h>.

9941 APPLICATION USAGE

9942 None.

9943 RATIONALE

9944 None.

9945 FUTURE DIRECTIONS

9946 None.

9947 SEE ALSO

9948 <inttypes.h>, <netinet/in.h>, <sys/socket.h>

9949 XSH *bind()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*, *freeaddrinfo()*, *getnameinfo()*

9950 CHANGE HISTORY

9951 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

9952 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type for
9953 *gai_strerror()* from **char *** to **const char ***. This is for coordination with the IPnG Working Group.

9954 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/11 is applied, adding a description of the
9955 NI_NUMERICSCOPE macro and correcting the *getnameinfo()* function prototype. These changes
9956 are for alignment with IPv6.

9957 Issue 7

9958 SD5-XBD-ERN-14 is applied, changing the description of the *s_port* member of the **servent**
9959 structure.

9960 The obsolescent *h_errno* external integer, and the obsolescent *gethostbyaddr()*, and
9961 *gethostbyname()* functions are removed, along with the HOST_NOT_FOUND, NO_DATA,
9962 NO_RECOVERY, and TRY_AGAIN macros.

9963 This reference page is clarified with respect to macros and symbolic constants.

9964 **NAME**

9965 netinet/in.h — Internet address family

9966 **SYNOPSIS**

9967 #include <netinet/in.h>

9968 **DESCRIPTION**

9969 The <netinet/in.h> header shall define the following types:

9970 **in_port_t** Equivalent to the type **uint16_t** as defined in <inttypes.h>.9971 **in_addr_t** Equivalent to the type **uint32_t** as defined in <inttypes.h>.9972 The **sa_family_t** type shall be defined as described in <sys/socket.h>.9973 The **uint8_t** and **uint32_t** type shall be defined as described in <inttypes.h>. Inclusion of the
9974 <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>.9975 The <netinet/in.h> header shall define the **in_addr** structure that includes at least the following
9976 member:

9977 in_addr_t s_addr

9978 The <netinet/in.h> header shall define the **sockaddr_in** structure that includes at least the
9979 following members:9980 sa_family_t sin_family AF_INET.
9981 in_port_t sin_port Port number.
9982 struct in_addr sin_addr IP address.9983 The *sin_port* and *sin_addr* members shall be in network byte order.9984 The **sockaddr_in** structure is used to store addresses for the Internet address family. Values of
9985 this type shall be cast by applications to **struct sockaddr** for use with socket functions.9986 IP6 The <netinet/in.h> header shall define the **in6_addr** structure that contains at least the following
9987 member:

9988 uint8_t s6_addr[16]

9989 This array is used to contain a 128-bit IPv6 address, stored in network byte order.

9990 The <netinet/in.h> header shall define the **sockaddr_in6** structure that includes at least the
9991 following members:9992 sa_family_t sin6_family AF_INET6.
9993 in_port_t sin6_port Port number.
9994 uint32_t sin6_flowinfo IPv6 traffic class and flow information.
9995 struct in6_addr sin6_addr IPv6 address.
9996 uint32_t sin6_scope_id Set of interfaces for a scope.9997 The *sin6_port* and *sin6_addr* members shall be in network byte order.9998 The **sockaddr_in6** structure shall be set to zero by an application prior to using it, since
9999 implementations are free to have additional, implementation-defined fields in **sockaddr_in6**.10000 The *sin6_scope_id* field is a 32-bit integer that identifies a set of interfaces as appropriate for the
10001 scope of the address carried in the *sin6_addr* field. For a link scope *sin6_addr*, the application
10002 shall ensure that *sin6_scope_id* is a link index. For a site scope *sin6_addr*, the application shall
10003 ensure that *sin6_scope_id* is a site index. The mapping of *sin6_scope_id* to an interface or set of
10004 interfaces is implementation-defined.

10005		The <netinet/in.h> header shall declare the following external variable:
10006		<code>const struct in6_addr in6addr_any</code>
10007		This variable is initialized by the system to contain the wildcard IPv6 address. The
10008		<netinet/in.h> header also defines the IN6ADDR_ANY_INIT macro. This macro must be
10009		constant at compile time and can be used to initialize a variable of type struct in6_addr to the
10010		IPv6 wildcard address.
10011		The <netinet/in.h> header shall declare the following external variable:
10012		<code>const struct in6_addr in6addr_loopback</code>
10013		This variable is initialized by the system to contain the loopback IPv6 address. The
10014		<netinet/in.h> header also defines the IN6ADDR_LOOPBACK_INIT macro. This macro must be
10015		constant at compile time and can be used to initialize a variable of type struct in6_addr to the
10016		IPv6 loopback address.
10017		The <netinet/in.h> header shall define the ipv6_mreq structure that includes at least the
10018		following members:
10019		<code>struct in6_addr ipv6mr_multiaddr</code> IPv6 multicast address.
10020		<code>unsigned ipv6mr_interface</code> Interface index.
10021		The <netinet/in.h> header shall define the following symbolic constants for use as values of the
10022		<i>level</i> argument of <i>getsockopt()</i> and <i>setsockopt()</i> :
10023		IPPROTO_IP Internet protocol.
10024	IP6	IPPROTO_IPV6 Internet Protocol Version 6.
10025		IPPROTO_ICMP Control message protocol.
10026	RS	IPPROTO_RAW Raw IP Packets Protocol.
10027		IPPROTO_TCP Transmission control protocol.
10028		IPPROTO_UDP User datagram protocol.
10029		The <netinet/in.h> header shall define the following symbolic constants for use as destination
10030		addresses for <i>connect()</i> , <i>sendmsg()</i> , and <i>sendto()</i> :
10031		INADDR_ANY IPv4 local host address.
10032		INADDR_BROADCAST IPv4 broadcast address.
10033		The <netinet/in.h> header shall define the following symbolic constant, with the value
10034		specified, to help applications declare buffers of the proper size to store IPv4 addresses in string
10035		form:
10036		INET_ADDRSTRLEN 16. Length of the string form for IP.
10037		The <i>htonl()</i> , <i>htons()</i> , <i>ntohl()</i> , and <i>ntohs()</i> functions shall be available as defined in <arpa/inet.h>.
10038		Inclusion of the <netinet/in.h> header may also make visible all symbols from <arpa/inet.h>.
10039	IP6	The <netinet/in.h> header shall define the following symbolic constant, with the value
10040		specified, to help applications declare buffers of the proper size to store IPv6 addresses in string
10041		form:
10042		INET6_ADDRSTRLEN 46. Length of the string form for IPv6.

10043	IP6	The <netinet/in.h> header shall define the following symbolic constants, with distinct integer values, for use in the <i>option_name</i> argument in the <i>getsockopt()</i> or <i>setsockopt()</i> functions at protocol level IPPROTO_IPV6:
10044		
10045		
10046		IPV6_JOIN_GROUP Join a multicast group.
10047		IPV6_LEAVE_GROUP Quit a multicast group.
10048		IPV6_MULTICAST_HOPS
10049		Multicast hop limit.
10050		IPV6_MULTICAST_IF Interface to use for outgoing multicast packets.
10051		IPV6_MULTICAST_LOOP
10052		Multicast packets are delivered back to the local application.
10053		IPV6_UNICAST_HOPS Unicast hop limit.
10054		IPV6_V6ONLY Restrict AF_INET6 socket to IPv6 communications only.
10055		The <netinet/in.h> header shall define the following macros that test for special IPv6 addresses.
10056		Each macro is of type int and takes a single argument of type const struct in6_addr * :
10057		IN6_IS_ADDR_UNSPECIFIED
10058		Unspecified address.
10059		IN6_IS_ADDR_LOOPBACK
10060		Loopback address.
10061		IN6_IS_ADDR_MULTICAST
10062		Multicast address.
10063		IN6_IS_ADDR_LINKLOCAL
10064		Unicast link-local address.
10065		IN6_IS_ADDR_SITELOCAL
10066		Unicast site-local address.
10067		IN6_IS_ADDR_V4MAPPED
10068		IPv4 mapped address.
10069		IN6_IS_ADDR_V4COMPAT
10070		IPv4-compatible address.
10071		IN6_IS_ADDR_MC_NODELOCAL
10072		Multicast node-local address.
10073		IN6_IS_ADDR_MC_LINKLOCAL
10074		Multicast link-local address.
10075		IN6_IS_ADDR_MC_SITELOCAL
10076		Multicast site-local address.
10077		IN6_IS_ADDR_MC_ORGLOCAL
10078		Multicast organization-local address.
10079		IN6_IS_ADDR_MC_GLOBAL
10080		Multicast global address.

10081 **APPLICATION USAGE**

10082 None.

10083 **RATIONALE**

10084 None.

10085 **FUTURE DIRECTIONS**

10086 None.

10087 **SEE ALSO**10088 [Section 4.9](#) (on page 98), [<arpa/inet.h>](#), [<inttypes.h>](#), [<sys/socket.h>](#)10089 XSH [connect\(\)](#), [getsockopt\(\)](#), [htonl\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#) |10090 **CHANGE HISTORY**

10091 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10092 The *sin_zero* member was removed from the **sockaddr_in** structure as per The Open Group Base
10093 Resolution bwg2001-004.10094 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/12 is applied, adding **const** qualifiers to
10095 the *in6addr_any* and *in6addr_loopback* external variables.10096 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/22 is applied, making it clear which
10097 structure members are in network byte order.10098 **Issue 7**

10099 This reference page is clarified with respect to macros and symbolic constants. +

DRAFT

10100 **NAME**

10101 netinet/tcp.h — definitions for the Internet Transmission Control Protocol (TCP)

10102 **SYNOPSIS**

10103 #include <netinet/tcp.h>

10104 **DESCRIPTION**10105 The <netinet/tcp.h> header shall define the following symbolic constant for use as a socket |
10106 option at the IPPROTO_TCP level:

10107 TCP_NODELAY Avoid coalescing of small segments.

10108 The implementation need not allow the value of the option to be set via *setsockopt()* or retrieved |
10109 via *getsockopt()*.10110 **APPLICATION USAGE**

10111 None.

10112 **RATIONALE**

10113 None.

10114 **FUTURE DIRECTIONS**

10115 None.

10116 **SEE ALSO**

10117 <sys/socket.h>

10118 XSH *getsockopt()*, *setsockopt()* |10119 **CHANGE HISTORY**

10120 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

10121 **Issue 7**

10122 This reference page is clarified with respect to macros and symbolic constants. +

10123 **NAME**

10124 nl_types.h — data types

10125 **SYNOPSIS**

10126 #include <nl_types.h>

10127 **DESCRIPTION**

10128 The <nl_types.h> header shall contain definitions of at least the following types:

10129 **nl_catd** Used by the message catalog functions *catopen()*, *catgets()*, and *catclose()*
 10130 to identify a catalog descriptor.

10131 **nl_item** Used by *nl_langinfo()* to identify items of *langinfo* data. Values of objects
 10132 of type **nl_item** are defined in <langinfo.h>.

10133 The <nl_types.h> header shall contain definitions of at least the following symbolic constants:

10134 **NL_SETD** Used by *gencat* when no *\$set* directive is specified in a message text source
 10135 file. This constant can be passed as the value of *set_id* on subsequent calls
 10136 to *catgets()* (that is, to retrieve messages from the default message set).
 10137 The value of **NL_SETD** is implementation-defined.

10138 **NL_CAT_LOCALE** Value that must be passed as the *oflag* argument to *catopen()* to ensure that
 10139 message catalog selection depends on the *LC_MESSAGES* locale category,
 10140 rather than directly on the *LANG* environment variable.

10141 The following shall be declared as functions and may also be defined as macros. Function
10142 prototypes shall be provided.

```
10143 int      catclose(nl_catd);
10144 char    *catgets(nl_catd, int, int, const char *);
10145 nl_catd catopen(const char *, int);
```

10146 **APPLICATION USAGE**

10147 None.

10148 **RATIONALE**

10149 None.

10150 **FUTURE DIRECTIONS**

10151 None.

10152 **SEE ALSO**

10153 <langinfo.h>

10154 XSH *catclose()*, *catgets()*, *catopen()*, *nl_langinfo()*10155 XCU *gencat*10156 **CHANGE HISTORY**

10157 First released in Issue 2.

10158 **Issue 7**

10159 The <nl_types.h> header is moved from the XSI option to the Base.

10160 This reference page is clarified with respect to macros and symbolic constants. +

10161 **NAME**
 10162 poll.h — definitions for the poll() function

10163 **SYNOPSIS**
 10164 #include <poll.h>

10165 **DESCRIPTION**
 10166 The <poll.h> header shall define the **pollfd** structure that includes at least the following
 10167 members:

10168	int	fd	The following descriptor being polled.
10169	short	events	The input event flags (see below).
10170	short	revents	The output event flags (see below).

10171 The <poll.h> header shall define the following type through **typedef**:

10172 **nfds_t** An unsigned integer type used for the number of file descriptors.

10173 The implementation shall support one or more programming environments in which the width
 10174 of **nfds_t** is no greater than the width of type **long**. The names of these programming
 10175 environments can be obtained using the *confstr()* function or the *getconf* utility.

10176 The following symbolic constants shall be defined, zero or more of which may be OR'ed
 10177 together to form the *events* or *revents* members in the **pollfd** structure:

10178	POLLIN	Data other than high-priority data may be read without blocking.
10179	POLLRDNORM	Normal data may be read without blocking.
10180	POLLRDBAND	Priority data may be read without blocking.
10181	POLLPRI	High priority data may be read without blocking.
10182	POLLOUT	Normal data may be written without blocking.
10183	POLLWRNORM	Equivalent to POLLOUT.
10184	POLLWRBAND	Priority data may be written.
10185	POLLERR	An error has occurred (<i>revents</i> only).
10186	POLLHUP	Device has been disconnected (<i>revents</i> only).
10187	POLLNVAL	Invalid <i>fd</i> member (<i>revents</i> only).

10188 The significance and semantics of normal, priority, and high-priority data are file and device-
 10189 specific.

10190 The following shall be declared as a function and may also be defined as a macro. A function
 10191 prototype shall be provided.

10192 int poll(struct pollfd [], nfds_t, int);

10193 **APPLICATION USAGE**

10194 None.

10195 **RATIONALE**

10196 None.

10197 **FUTURE DIRECTIONS**

10198 None.

10199 **SEE ALSO**10200 XSH *confstr()*, *poll()*10201 XCU *getconf*10202 **CHANGE HISTORY**

10203 First released in Issue 4, Version 2.

10204 **Issue 6**10205 The description of the symbolic constants is updated to match the *poll()* function.10206 Text related to STREAMS has been moved to the *poll()* reference page.10207 A note is added to the DESCRIPTION regarding the significance and semantics of normal,
10208 priority, and high-priority data.10209 **Issue 7**

10210 The <poll.h> header is moved from the XSI option to the Base.

10211 **NAME**

10212 pthread.h — threads

10213 **SYNOPSIS**

10214 #include <pthread.h>

10215 **DESCRIPTION**

10216 The <pthread.h> header shall define the following symbolic constants: |

10217 PTHREAD_BARRIER_SERIAL_THREAD

10218 PTHREAD_CANCEL_ASYNCHRONOUS

10219 PTHREAD_CANCEL_ENABLE

10220 PTHREAD_CANCEL_DEFERRED

10221 PTHREAD_CANCEL_DISABLE

10222 PTHREAD_CANCELED

10223 PTHREAD_CREATE_DETACHED -

10224 PTHREAD_CREATE_JOINABLE

10225 TPS PTHREAD_EXPLICIT_SCHED

10226 PTHREAD_INHERIT_SCHED

10227 PTHREAD_MUTEX_DEFAULT

10228 PTHREAD_MUTEX_ERRORCHECK

10229 PTHREAD_MUTEX_NORMAL -

10230 PTHREAD_MUTEX_RECURSIVE

10231 PTHREAD_MUTEX_ROBUST

10232 PTHREAD_MUTEX_STALLED

10233 PTHREAD_ONCE_INIT

10234 RPI | TPI PTHREAD_PRIO_INHERIT

10235 MC1 PTHREAD_PRIO_NONE

10236 RPP | TPP PTHREAD_PRIO_PROTECT

10237 PTHREAD_PROCESS_SHARED

10238 PTHREAD_PROCESS_PRIVATE

10239 TPS PTHREAD_SCOPE_PROCESS -

10240 PTHREAD_SCOPE_SYSTEM

10241 The <pthread.h> header shall define the following compile-time constant expressions valid as +
10242 initializers for the following types: +

Name	Initializer for Type
PTHREAD_COND_INITIALIZER	pthread_cond_t
PTHREAD_MUTEX_INITIALIZER	pthread_mutex_t
PTHREAD_RWLOCK_INITIALIZER	pthread_rwlock_t

10247 The following types shall be defined as described in <sys/types.h>:

10248 pthread_attr_t

10249 pthread_barrier_t

10250 pthread_barrierattr_t

10251 pthread_cond_t

10252 pthread_condattr_t

10253 pthread_key_t

10254 pthread_mutex_t

10255 pthread_mutexattr_t

10256 pthread_once_t

10257 pthread_rwlock_t

10258 **pthread_rwlockattr_t**
 10259 **pthread_spinlock_t**
 10260 **pthread_t**

10261 The following shall be declared as functions and may also be defined as macros. Function
 10262 prototypes shall be provided.

```

10263 int pthread_atfork(void (*)(void), void (*)(void),
10264                  void (*)(void));
10265 int pthread_attr_destroy(pthread_attr_t *);
10266 int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
10267 int pthread_attr_getguardsize(const pthread_attr_t *restrict,
10268                               size_t *restrict);
10269 TPS int pthread_attr_getinheritsched(const pthread_attr_t *restrict,
10270                                     int *restrict);
10271 int pthread_attr_getschedparam(const pthread_attr_t *restrict,
10272                               struct sched_param *restrict);
10273 TPS int pthread_attr_getschedpolicy(const pthread_attr_t *restrict,
10274                                    int *restrict);
10275 int pthread_attr_getscope(const pthread_attr_t *restrict,
10276                           int *restrict);
10277 TSA TSS int pthread_attr_getstack(const pthread_attr_t *restrict,
10278                                  void **restrict, size_t *restrict);
10279 TSS int pthread_attr_getstacksize(const pthread_attr_t *restrict,
10280                                  size_t *restrict);
10281 int pthread_attr_init(pthread_attr_t *);
10282 int pthread_attr_setdetachstate(pthread_attr_t *, int);
10283 int pthread_attr_setguardsize(pthread_attr_t *, size_t);
10284 TPS int pthread_attr_setinheritsched(pthread_attr_t *, int);
10285 int pthread_attr_setschedparam(pthread_attr_t *restrict,
10286                               const struct sched_param *restrict);
10287 TPS int pthread_attr_setschedpolicy(pthread_attr_t *, int);
10288 int pthread_attr_setscope(pthread_attr_t *, int);
10289 TSA TSS int pthread_attr_setstack(pthread_attr_t *, void *, size_t);
10290 TSS int pthread_attr_setstacksize(pthread_attr_t *, size_t);
10291 int pthread_barrier_destroy(pthread_barrier_t *);
10292 int pthread_barrier_init(pthread_barrier_t *restrict,
10293                         const pthread_barrierattr_t *restrict, unsigned);
10294 int pthread_barrier_wait(pthread_barrier_t *);
10295 int pthread_barrierattr_destroy(pthread_barrierattr_t *);
10296 TSH int pthread_barrierattr_getpshared(
10297     const pthread_barrierattr_t *restrict, int *restrict);
10298 int pthread_barrierattr_init(pthread_barrierattr_t *);
10299 TSH int pthread_barrierattr_setpshared(pthread_barrierattr_t *, int);
10300 int pthread_cancel(pthread_t);
10301 void pthread_cleanup_pop(int);
10302 void pthread_cleanup_push(void (*)(void*), void *);
10303 int pthread_cond_broadcast(pthread_cond_t *);
10304 int pthread_cond_destroy(pthread_cond_t *);
10305 int pthread_cond_init(pthread_cond_t *restrict,
10306                      const pthread_condattr_t *restrict);
10307 int pthread_cond_signal(pthread_cond_t *);
10308 int pthread_cond_timedwait(pthread_cond_t *restrict,
10309                            pthread_mutex_t *restrict, const struct timespec *restrict);
10310 int pthread_cond_wait(pthread_cond_t *restrict,

```



```

10311         pthread_mutex_t *restrict);
10312 int pthread_condattr_destroy(pthread_condattr_t *);
10313 int pthread_condattr_getclock(const pthread_condattr_t *restrict,
10314         clockid_t *restrict);
10315 TSH int pthread_condattr_getpshared(const pthread_condattr_t *restrict,
10316         int *restrict);
10317 int pthread_condattr_init(pthread_condattr_t *);
10318 int pthread_condattr_setclock(pthread_condattr_t *, clockid_t);
10319 TSH int pthread_condattr_setpshared(pthread_condattr_t *, int);
10320 int pthread_create(pthread_t *restrict, const pthread_attr_t *restrict,
10321         void *(*)(void*), void *restrict);
10322 int pthread_detach(pthread_t);
10323 int pthread_equal(pthread_t, pthread_t);
10324 void pthread_exit(void *);
10325 OB XSI int pthread_getconcurrency(void);
10326 TCT int pthread_getcpuclockid(pthread_t, clockid_t *);
10327 TPS int pthread_getschedparam(pthread_t, int *restrict,
10328         struct sched_param *restrict);
10329 void *pthread_getspecific(pthread_key_t);
10330 int pthread_join(pthread_t, void **);
10331 int pthread_key_create(pthread_key_t *, void (*)(void*));
10332 int pthread_key_delete(pthread_key_t);
10333 int pthread_mutex_consistent(pthread_mutex_t *);
10334 int pthread_mutex_destroy(pthread_mutex_t *);
10335 RPP|TPP int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict,
10336         int *restrict);
10337 int pthread_mutex_init(pthread_mutex_t *restrict,
10338         const pthread_mutexattr_t *restrict);
10339 int pthread_mutex_lock(pthread_mutex_t *);
10340 RPP|TPP int pthread_mutex_setprioceiling(pthread_mutex_t *restrict, int,
10341         int *restrict);
10342 int pthread_mutex_timedlock(pthread_mutex_t *restrict,
10343         const struct timespec *restrict);
10344 int pthread_mutex_trylock(pthread_mutex_t *);
10345 int pthread_mutex_unlock(pthread_mutex_t *);
10346 int pthread_mutexattr_destroy(pthread_mutexattr_t *);
10347 RPP|TPP int pthread_mutexattr_getprioceiling(
10348         const pthread_mutexattr_t *restrict, int *restrict);
10349 MC1 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *restrict,
10350         int *restrict);
10351 TSH int pthread_mutexattr_getpshared(const pthread_mutexattr_t *restrict,
10352         int *restrict);
10353 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict,
10354         int *restrict);
10355 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict,
10356         int *restrict);
10357 int pthread_mutexattr_init(pthread_mutexattr_t *);
10358 RPP|TPP int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *, int);
10359 MC1 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *, int);
10360 TSH int pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
10361 int pthread_mutexattr_setrobust(pthread_mutexattr_t *, int);
10362 int pthread_mutexattr_settype(pthread_mutexattr_t *, int);
10363 int pthread_once(pthread_once_t *, void (*)(void));
10364 int pthread_rwlock_destroy(pthread_rwlock_t *);

```

<pthread.h>

Headers

```

10365     int   pthread_rwlock_init(pthread_rwlock_t *restrict,
10366         const pthread_rwlockattr_t *restrict);
10367     int   pthread_rwlock_rdlock(pthread_rwlock_t *);
10368     int   pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict,
10369         const struct timespec *restrict);
10370     int   pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict,
10371         const struct timespec *restrict);
10372     int   pthread_rwlock_tryrdlock(pthread_rwlock_t *);
10373     int   pthread_rwlock_trywrlock(pthread_rwlock_t *);
10374     int   pthread_rwlock_unlock(pthread_rwlock_t *);
10375     int   pthread_rwlock_wrlock(pthread_rwlock_t *);
10376     int   pthread_rwlockattr_destroy(pthread_rwlockattr_t *);
10377     TSH   int   pthread_rwlockattr_getpshared(
10378         const pthread_rwlockattr_t *restrict, int *restrict);
10379     int   pthread_rwlockattr_init(pthread_rwlockattr_t *);
10380     TSH   int   pthread_rwlockattr_setpshared(pthread_rwlockattr_t *, int);
10381     pthread_t
10382         pthread_self(void);
10383     int   pthread_setcancelstate(int, int *);
10384     int   pthread_setcanceltype(int, int *);
10385     OB XSI int   pthread_setconcurrency(int);
10386     TPS   int   pthread_setschedparam(pthread_t, int,
10387         const struct sched_param *);
10388     int   pthread_setschedprio(pthread_t, int);
10389     int   pthread_setspecific(pthread_key_t, const void *);
10390     int   pthread_spin_destroy(pthread_spinlock_t *);
10391     int   pthread_spin_init(pthread_spinlock_t *, int);
10392     int   pthread_spin_lock(pthread_spinlock_t *);
10393     int   pthread_spin_trylock(pthread_spinlock_t *);
10394     int   pthread_spin_unlock(pthread_spinlock_t *);
10395     void  pthread_testcancel(void);

```

10396 Inclusion of the **<pthread.h>** header shall make symbols defined in the headers **<sched.h>** and
10397 **<time.h>** visible.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO**<sched.h>**, **<sys/types.h>**, **<time.h>**

```

10406     XSH pthread_attr_destroy(), pthread_attr_getguardsize(), pthread_attr_getscope(), |
10407     pthread_barrier_destroy(), pthread_barrier_wait(), pthread_barrierattr_destroy(), |
10408     pthread_barrierattr_getpshared(), pthread_cancel(), pthread_cleanup_pop(), pthread_cond_broadcast(), |
10409     pthread_cond_destroy(), pthread_cond_timedwait(), pthread_condattr_getclock(), |
10410     pthread_condattr_destroy(), pthread_create(), pthread_detach(), pthread_equal(), pthread_exit(), |
10411     pthread_getconcurrency(), pthread_getcpuclockid(), pthread_getschedparam(), pthread_getspecific(), |
10412     pthread_join(), pthread_key_create(), pthread_key_delete(), pthread_mutex_getprioceiling(), |
10413     pthread_mutex_destroy(), pthread_mutex_lock(), pthread_mutex_timedlock(), |
10414     pthread_mutexattr_destroy(), pthread_mutexattr_getprotocol(), pthread_mutexattr_gettype(), |
10415     pthread_once(), pthread_rwlock_destroy(), pthread_rwlock_rdlock(), pthread_rwlock_timedrdlock(), |

```

10416 *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*,
 10417 *pthread_rwlockattr_destroy()*, *pthread_rwlockattr_getpshared()*, *pthread_self()*,
 10418 *pthread_setcancelstate()*, *pthread_spin_destroy()*, *pthread_spin_lock()*, *pthread_spin_unlock()*

CHANGE HISTORY

10419 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

10420 The RTT margin markers are broken out into their POSIX options.

10421 The Open Group Corrigendum U021/9 is applied, correcting the prototype for the
 10422 *pthread_cond_wait()* function.

10423 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the
 10424 *pthread_setschedparam()* function so that its second argument is of type **int**.

10425 The *pthread_getcpuclockid()* and *pthread_mutex_timedlock()* functions are added for alignment
 10426 with IEEE Std 1003.1d-1999.

10427 The following functions are added for alignment with IEEE Std 1003.1j-2000:

10428 *pthread_barrier_destroy()*, *pthread_barrier_init()*, *pthread_barrier_wait()*,
 10429 *pthread_barrierattr_destroy()*, *pthread_barrierattr_getpshared()*, *pthread_barrierattr_init()*,
 10430 *pthread_barrierattr_setpshared()*, *pthread_condattr_getclock()*, *pthread_condattr_setclock()*,
 10431 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_spin_destroy()*,
 10432 *pthread_spin_init()*, *pthread_spin_lock()*, *pthread_spin_trylock()*, and *pthread_spin_unlock()*.

10433 PTHREAD_RWLOCK_INITIALIZER is removed for alignment with IEEE Std 1003.1j-2000.

10434 Functions previously marked as part of the Read-Write Locks option are now moved to the
 10435 Threads option.

10436 The **restrict** keyword is added to the prototypes for *pthread_attr_getguardsize()*,
 10437 *pthread_attr_getinheritsched()*, *pthread_attr_getschedparam()*, *pthread_attr_getschedpolicy()*,
 10438 *pthread_attr_getscope()*, *pthread_attr_getstackaddr()*, *pthread_attr_getstacksize()*,
 10439 *pthread_attr_getschedparam()*, *pthread_barrier_init()*, *pthread_barrierattr_getpshared()*,
 10440 *pthread_cond_init()*, *pthread_cond_signal()*, *pthread_cond_timedwait()*, *pthread_cond_wait()*,
 10441 *pthread_condattr_getclock()*, *pthread_condattr_getpshared()*, *pthread_create()*,
 10442 *pthread_getschedparam()*, *pthread_mutex_getprioceiling()*, *pthread_mutex_init()*,
 10443 *pthread_mutex_setprioceiling()*, *pthread_mutexattr_getprioceiling()*, *pthread_mutexattr_getprotocol()*,
 10444 *pthread_mutexattr_getpshared()*, *pthread_mutexattr_gettype()*, *pthread_rwlock_init()*,
 10445 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlockattr_getpshared()*, and
 10446 *pthread_sigmask()*.

10447 IEEE PASC Interpretation 1003.1 #86 is applied, allowing the symbols from <sched.h> and
 10448 <time.h> to be made visible when <pthread.h> is included. Previously this was an XSI option.

10449 IEEE PASC Interpretation 1003.1c #42 is applied, removing the requirement for prototypes for
 10450 the *pthread_kill()* and *pthread_sigmask()* functions. These are required to be in the <signal.h>
 10451 header. They are allowed here through the name space rules.

10452 IEEE PASC Interpretation 1003.1 #96 is applied, adding the *pthread_setschedprio()* function.

10453 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/13 is applied, correcting shading errors
 10454 that were in contradiction with the System Interfaces volume of POSIX.1-200x.

Issue 7

10455 SD5-XBD-ERN-55 is applied, adding the **restrict** keyword to the *pthread_mutex_timedlock()*
 10456 function prototype.

10457 SD5-XBD-ERN-62 is applied.

10458 Austin Group Interpretation 1003.1-2001 #048 is applied.

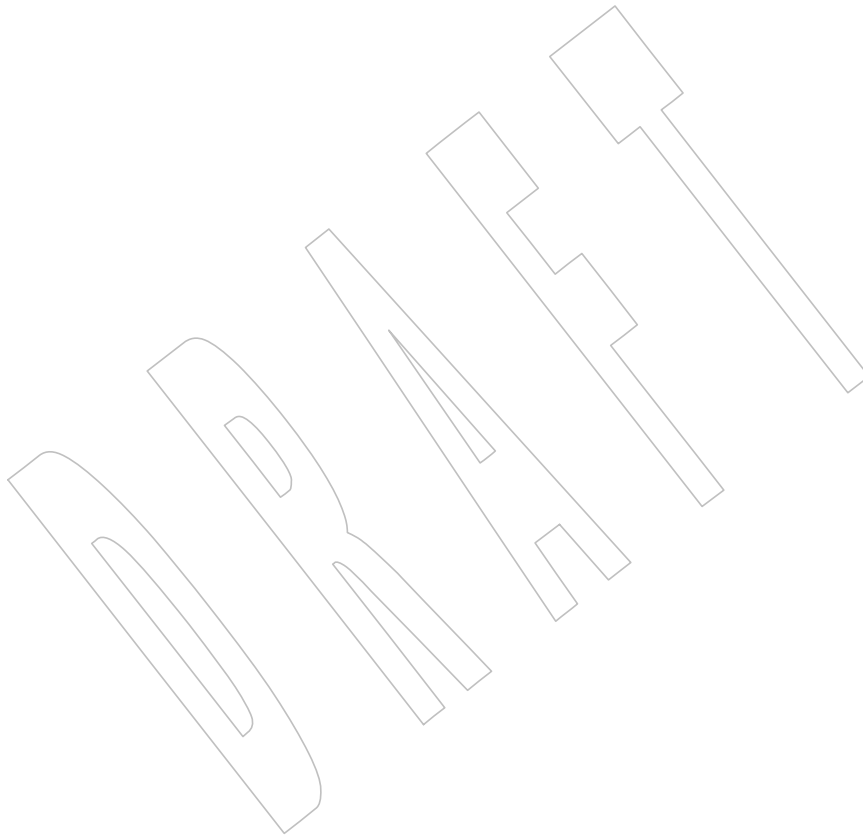
10462 The <pthread.h> header is moved from the Threads option to the Base.

10463 The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK,
10464 PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended mutex types
10465 are moved from the XSI option to the Base.

10466 The PTHREAD_MUTEX_ROBUST and PTHREAD_MUTEX_STALLED symbols and the
10467 *pthread_mutex_consistent()*, *pthread_mutexattr_getrobust()*, and *pthread_mutexattr_setrobust()*
10468 functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

10469 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
10470 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
10471 or Robust Mutex Priority Inheritance, respectively.

10472 This reference page is clarified with respect to macros and symbolic constants. +



10473 **NAME**
 10474 `pwd.h` — password structure

10475 **SYNOPSIS**
 10476 `#include <pwd.h>`

10477 **DESCRIPTION**
 10478 The `<pwd.h>` header shall provide a definition for **struct passwd**, which shall include at least
 10479 the following members:

10480	<code>char</code>	<code>*pw_name</code>	User's login name.
10481	<code>uid_t</code>	<code>pw_uid</code>	Numerical user ID.
10482	<code>gid_t</code>	<code>pw_gid</code>	Numerical group ID.
10483	<code>char</code>	<code>*pw_dir</code>	Initial working directory.
10484	<code>char</code>	<code>*pw_shell</code>	Program to use as shell.

10485 The `gid_t`, `uid_t`, and `size_t` types shall be defined as described in `<sys/types.h>`.

10486 The following shall be declared as functions and may also be defined as macros. Function
 10487 prototypes shall be provided.

```

10488 struct passwd *getpwnam(const char *);
10489 int           getpwnam_r(const char *, struct passwd *, char *,
10490                        size_t, struct passwd **);
10491 struct passwd *getpwuid(uid_t);
10492 int           getpwuid_r(uid_t, struct passwd *, char *,
10493                        size_t, struct passwd **);
10494 XSI void      endpwent(void);
10495 struct passwd *getpwent(void);
10496 void      setpwent(void);

```

10497 **APPLICATION USAGE**
 10498 None.

10499 **RATIONALE**
 10500 None.

10501 **FUTURE DIRECTIONS**
 10502 None.

10503 **SEE ALSO**
 10504 [<sys/types.h>](#)

10505 XSH [endpwent\(\)](#), [getpwnam\(\)](#), [getpwuid\(\)](#)

10506 **CHANGE HISTORY**
 10507 First released in Issue 1.

10508 **Issue 5**
 10509 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

10510 **Issue 6**
 10511 The following new requirements on POSIX implementations derive from alignment with the
 10512 Single UNIX Specification:

- 10513 • The `gid_t` and `uid_t` types are mandated.

10514

10515

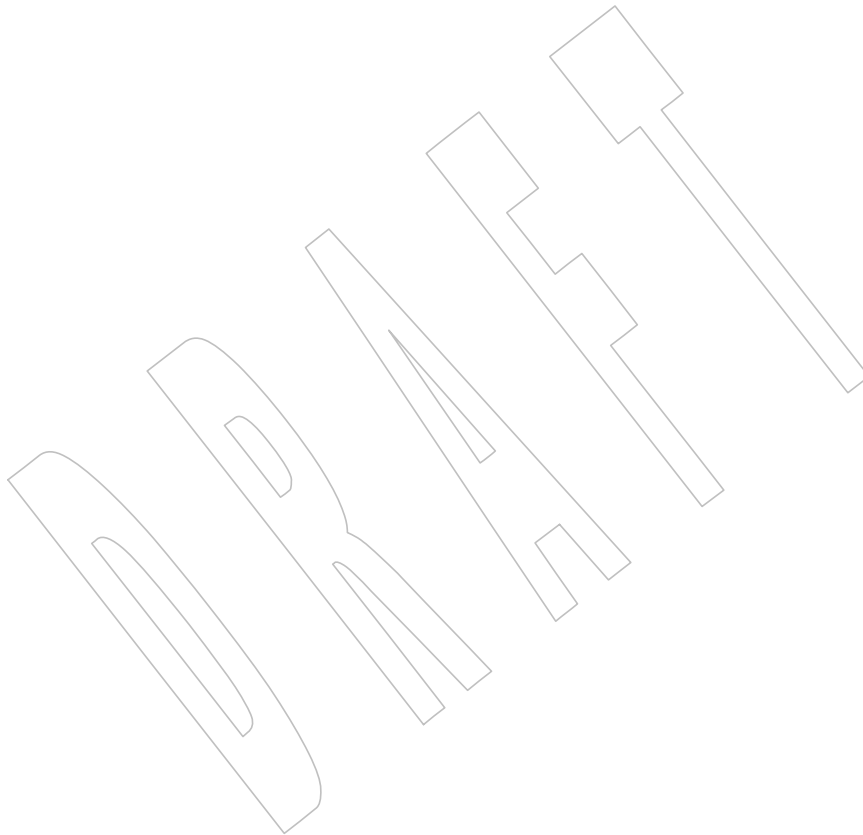
- The *getpwnam_r()* and *getpwuid_r()* functions are marked as part of the Thread-Safe Functions option.

10516

Issue 7

10517

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.



10518 **NAME**10519 `regex.h` — regular expression matching types10520 **SYNOPSIS**10521 `#include <regex.h>`10522 **DESCRIPTION**10523 The <regex.h> header shall define the structures and symbolic constants used by the `regcomp()`,
10524 `regexexec()`, `regerror()`, and `regfree()` functions.10525 The structure type `regex_t` shall contain at least the following member:10526 `size_t re_nsub` Number of parenthesized subexpressions.10527 The type `size_t` shall be defined as described in <sys/types.h>.10528 The type `regoff_t` shall be defined as a signed integer type that can hold the largest value that
10529 can be stored in either a type `ptrdiff_t` or a type `ssize_t`. The structure type `regmatch_t` shall
10530 contain at least the following members:10531 `regoff_t rm_so` Byte offset from start of string
10532 to start of substring.
10533 `regoff_t rm_eo` Byte offset from start of string of the
10534 first character after the end of substring.10535 The <regex.h> header shall define the following symbolic constants for the `cflags` parameter to
10536 the `regcomp()` function: |10537 `REG_EXTENDED` Use Extended Regular Expressions.10538 `REG_ICASE` Ignore case in match.10539 `REG_NOSUB` Report only success or fail in `regexexec()`.10540 `REG_NEWLINE` Change the handling of <newline>.10541 The <regex.h> header shall define the following symbolic constants for the `eflags` parameter to
10542 the `regexexec()` function: |10543 `REG_NOTBOL` The circumflex character ('`^`'), when taken as a special character, does
10544 not match the beginning of *string*.10545 `REG_NOTEOL` The dollar sign ('`$`'), when taken as a special character, does not match
10546 the end of *string*.

10547 The <regex.h> header shall define the following symbolic constants as error return values: |

10548 `REG_NOMATCH` `regexexec()` failed to match.10549 `REG_BADPAT` Invalid regular expression.10550 `REG_ECOLLATE` Invalid collating element referenced.10551 `REG_ECTYPE` Invalid character class type referenced.10552 `REG_EESCAPE` Trailing '`\`' in pattern.10553 `REG_ESUBREG` Number in `\digit` invalid or in error.10554 `REG_EBRACK` "`[]`" imbalance.10555 `REG_EPAREN` "`\(\)`" or "`()`" imbalance.

<regex.h>

Headers

10556	REG_EBRACE	"\{\}" imbalance.
10557	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than
10558		two numbers, first larger than second.
10559	REG_ERANGE	Invalid endpoint in range expression.
10560	REG_ESPACE	Out of memory.
10561	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.

10562 The following shall be declared as functions and may also be defined as macros. Function
10563 prototypes shall be provided.

```
10564 int regcomp(regex_t *restrict, const char *restrict, int);
10565 size_t regerror(int, const regex_t *restrict, char *restrict, size_t);
10566 int regexec(const regex_t *restrict, const char *restrict, size_t,
10567             regmatch_t [restrict], int);
10568 void regfree(regex_t *);
```

10569 The implementation may define additional macros or constants using names beginning with
10570 REG_.

10571 APPLICATION USAGE

10572 None.

10573 RATIONALE

10574 None.

10575 FUTURE DIRECTIONS

10576 None.

10577 SEE ALSO

10578 [<sys/types.h>](#)

10579 XSH *regcomp()*

10580 CHANGE HISTORY

10581 First released in Issue 4.

10582 Originally derived from the ISO POSIX-2 standard.

10583 Issue 6

10584 The REG_ENOSYS constant is marked obsolescent.

10585 The **restrict** keyword is added to the prototypes for *regcomp()*, *regerror()*, and *regexec()*.

10586 A statement is added that the **size_t** type is defined as described in [<sys/types.h>](#).

10587 Issue 7

10588 SD5-XBD-ERN-60 is applied.

10589 The obsolescent REG_ENOSYS constant is removed.

10590 This reference page is clarified with respect to macros and symbolic constants.

10591 **NAME**
 10592 sched.h — execution scheduling

10593 **SYNOPSIS**
 10594 #include <sched.h>

10595 **DESCRIPTION**
 10596 The <sched.h> header shall define the **sched_param** structure, which contains the scheduling
 10597 parameters required for implementation of each supported scheduling policy. This structure
 10598 shall contain at least the following member:

10599 int sched_priority Process or thread execution scheduling priority.

10600 SS|TSP The **sched_param** structure defined in <sched.h> shall contain the following members in
 10601 addition to those specified above:

10602	int	sched_ss_low_priority	Low scheduling priority for
10603			sporadic server.
10604	struct timespec	sched_ss_repl_period	Replenishment period for
10605			sporadic server.
10606	struct timespec	sched_ss_init_budget	Initial budget for sporadic server.
10607	int	sched_ss_max_repl	Maximum pending replenishments for
10608			sporadic server.

10609 Each process or thread is controlled by an associated scheduling policy and priority. Associated
 10610 with each policy is a priority range. Each policy definition specifies the minimum priority range
 10611 for that policy. The priority ranges for each policy may overlap the priority ranges of other
 10612 policies.

10613 Four scheduling policies are defined; others may be defined by the implementation. The four
 10614 standard policies are indicated by the values of the following symbolic constants:

10615 PS|TPS **SCHED_FIFO** First in-first out (FIFO) scheduling policy.

10616 PS|TPS **SCHED_RR** Round robin scheduling policy.

10617 SS|TSP **SCHED_SPORADIC** Sporadic server scheduling policy.

10618 PS|TPS **SCHED_OTHER** Another scheduling policy.

10619 The values of these constants are distinct.

10620 The following shall be declared as functions and may also be defined as macros. Function
 10621 prototypes shall be provided.

10622	PS TPS	int	sched_get_priority_max(int);
10623		int	sched_get_priority_min(int);
10624	PS	int	sched_getparam(pid_t, struct sched_param *);
10625		int	sched_getscheduler(pid_t);
10626	PS TPS	int	sched_rr_get_interval(pid_t, struct timespec *);
10627	PS	int	sched_setparam(pid_t, const struct sched_param *);
10628		int	sched_setscheduler(pid_t, int, const struct sched_param *);
10629		int	sched_yield(void);

10630 Inclusion of the <sched.h> header may make visible all symbols from the <time.h> header.

10631 **APPLICATION USAGE**

10632 None.

10633 **RATIONALE**

10634 None.

10635 **FUTURE DIRECTIONS**

10636 None.

10637 **SEE ALSO**10638 [<time.h>](#)10639 **CHANGE HISTORY**

10640 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10641 **Issue 6**

10642 The <sched.h> header is marked as part of the Process Scheduling option.

10643 Sporadic server members are added to the **sched_param** structure, and the SCHED_SPORADIC
10644 scheduling policy is added for alignment with IEEE Std 1003.1d-1999.10645 IEEE PASC Interpretation 1003.1 #108 is applied, correcting the **sched_param** structure whose
10646 members *sched_ss_repl_period* and *sched_ss_init_budget* should be type **struct timespec** and not
10647 **timespec**.

10648 Symbols from <time.h> may be made visible when <sched.h> is included.

10649 IEEE Std 1003.1-2001/Cor 1-2002, items XSH/TC1/D6/52 and XSH/TC1/D6/53 are applied,
10650 aligning the function prototype shading and margin codes with the System Interfaces volume of
10651 IEEE Std 1003.1-2001.10652 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/23 is applied, updating the
10653 DESCRIPTION to differentiate between thread and process execution.10654 **Issue 7**

10655 SD5-XBD-ERN-13 is applied.

10656 Austin Group Interpretation 1003.1-2001 #064 is applied.

10657 The <sched.h> headers is moved from the Threads option to the Base.

10658 **NAME**
 10659 search.h — search tables

10660 **SYNOPSIS**
 10661 XSI `#include <search.h>`

10662 **DESCRIPTION**
 10663 The <search.h> header shall define the **ENTRY** type for structure **entry** which shall include the
 10664 following members:

10665 char *key
 10666 void *data

10667 and shall define **ACTION** and **VISIT** as enumeration data types through type definitions as
 10668 follows:

10669 enum { FIND, ENTER } ACTION;
 10670 enum { preorder, postorder, endorder, leaf } VISIT;

10671 The **size_t** type shall be defined as described in <sys/types.h>.

10672 The following shall be declared as functions and may also be defined as macros. Function
 10673 prototypes shall be provided.

10674 int hcreate(size_t);
 10675 void hdestroy(void);
 10676 ENTRY *hsearch(ENTRY, ACTION);
 10677 void insque(void *, void *);
 10678 void *lfind(const void *, const void *, size_t *,
 10679 size_t, int (*)(const void *, const void *));
 10680 void *lsearch(const void *, void *, size_t *,
 10681 size_t, int (*)(const void *, const void *));
 10682 void remque(void *);
 10683 void *tdelete(const void *restrict, void **restrict,
 10684 int (*)(const void *, const void *));
 10685 void *tfind(const void *, void *const *,
 10686 int (*)(const void *, const void *));
 10687 void *tsearch(const void *, void **,
 10688 int (*)(const void *, const void *));
 10689 void twalk(const void *,
 10690 void (*)(const void *, VISIT, int));

10691 **APPLICATION USAGE**
 10692 None.

10693 **RATIONALE**
 10694 None.

10695 **FUTURE DIRECTIONS**
 10696 None.

10697 **SEE ALSO**
 10698 [<sys/types.h>](#)
 10699 XSH [hcreate\(\)](#), [insque\(\)](#), [lsearch\(\)](#), [tdelete\(\)](#)

10700
10701
10702
10703
10704
10705

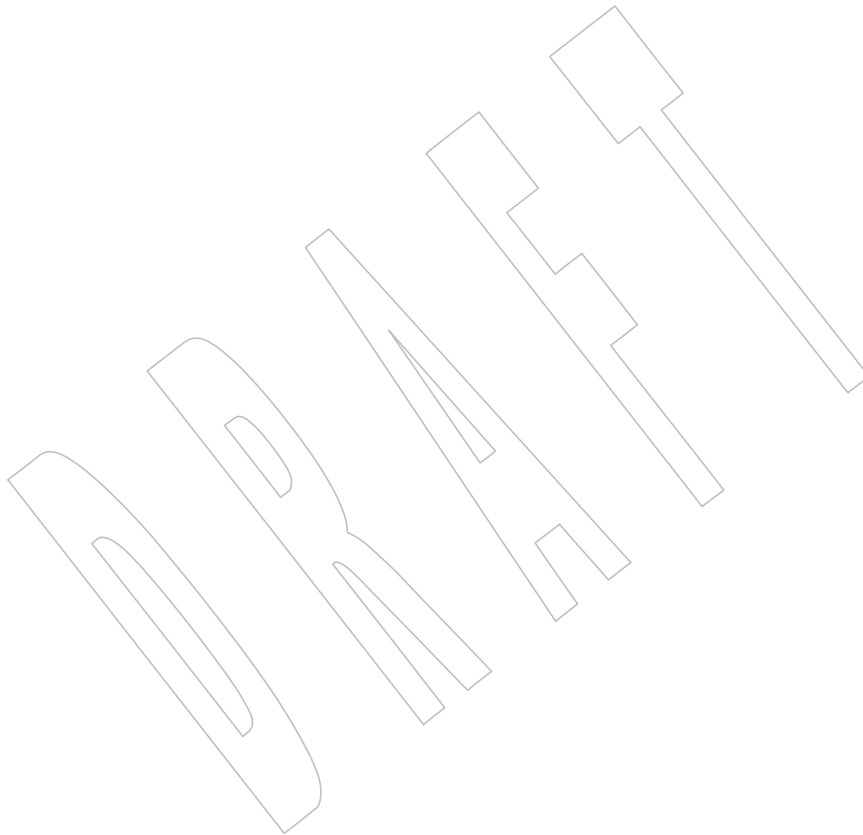
CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The Open Group Corrigendum U021/6 is applied, updating the prototypes for *tdelete()* and *tsearch()*.

The **restrict** keyword is added to the prototype for *tdelete()*.



10706 **NAME**

10707 semaphore.h — semaphores

10708 **SYNOPSIS**

10709 #include <semaphore.h>

10710 **DESCRIPTION**

10711 The <semaphore.h> header shall define the **sem_t** type, used in performing semaphore
 10712 operations. The semaphore may be implemented using a file descriptor, in which case
 10713 applications are able to open up at least a total of {OPEN_MAX} files and semaphores.

10714 The <semaphore.h> header shall define the symbolic constant SEM_FAILED which shall have
 10715 type **sem_t** *.

10716 The following shall be declared as functions and may also be defined as macros. Function
 10717 prototypes shall be provided.

```

10718 int     sem_close(sem_t *);
10719 int     sem_destroy(sem_t *);
10720 int     sem_getvalue(sem_t *restrict, int *restrict);
10721 int     sem_init(sem_t *, int, unsigned);
10722 sem_t *sem_open(const char *, int, ...);
10723 int     sem_post(sem_t *);
10724 int     sem_timedwait(sem_t *restrict, const struct timespec *restrict);
10725 int     sem_trywait(sem_t *);
10726 int     sem_unlink(const char *);
10727 int     sem_wait(sem_t *);

```

10728 Inclusion of the <semaphore.h> header may make visible symbols defined in the <fcntl.h>,
 10729 <sys/types.h>, and <time.h> headers.

10730 **APPLICATION USAGE**

10731 None.

10732 **RATIONALE**

10733 None.

10734 **FUTURE DIRECTIONS**

10735 None.

10736 **SEE ALSO**

10737 <fcntl.h>, <sys/types.h>, <time.h>

10738 XSH *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_timedwait()*,
 10739 *sem_trywait()*, *sem_unlink()*

10740 **CHANGE HISTORY**

10741 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

10742 **Issue 6**

10743 The <semaphore.h> header is marked as part of the Semaphores option.

10744 The Open Group Corrigendum U021/3 is applied, adding a description of SEM_FAILED.

10745 The *sem_timedwait()* function is added for alignment with IEEE Std 1003.1d-1999.10746 The **restrict** keyword is added to the prototypes for *sem_getvalue()* and *sem_timedwait()*.

<semaphore.h>*Headers*

10747

Issue 7

10748

SD5-XBD-ERN-57 is applied, allowing the header to make visible symbols from the **<time.h>** header.

10749

10750

The **<semaphore.h>** header is moved from the Semaphores option to the Base.

10751

This reference page is clarified with respect to macros and symbolic constants.

+



10752 **NAME**

10753 setjmp.h — stack environment declarations

10754 **SYNOPSIS**

10755 #include <setjmp.h>

10756 **DESCRIPTION**

10757 CX Some of the functionality described on this reference page extends the ISO C standard.
 10758 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 10759 enable the visibility of these symbols in this header.

10760 CX The <setjmp.h> header shall define the array types **jmp_buf** and **sigjmp_buf**.

10761 The following shall be declared as functions and may also be defined as macros. Function
 10762 prototypes shall be provided.

10763 void longjmp(jmp_buf, int);

10764 CX void siglongjmp(sigjmp_buf, int);

10765 OB XSI void _longjmp(jmp_buf, int);

10766 The following may be declared as functions, or defined as macros, or both. If functions are
 10767 declared, function prototypes shall be provided.

10768 int setjmp(jmp_buf);

10769 CX int sigsetjmp(sigjmp_buf, int);

10770 OB XSI int _setjmp(jmp_buf);

10771 **APPLICATION USAGE**

10772 None.

10773 **RATIONALE**

10774 None.

10775 **FUTURE DIRECTIONS**

10776 None.

10777 **SEE ALSO**10778 XSH Section 2.2 (on page 448), *longjmp()*, *_longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*10779 **CHANGE HISTORY**

10780 First released in Issue 1.

10781 **Issue 6**

10782 Extensions beyond the ISO C standard are marked.

10783 **Issue 7**

10784 SD5-XBD-ERN-6 is applied.

10785 **NAME**10786 `signal.h` — signals10787 **SYNOPSIS**10788 `#include <signal.h>`10789 **DESCRIPTION**

10790 CX Some of the functionality described on this reference page extends the ISO C standard.
 10791 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 10792 enable the visibility of these symbols in this header.

10793 The <signal.h> header shall define the following macros, which shall expand to constant
 10794 expressions with distinct values that have a type compatible with the second argument to, and
 10795 the return value of, the *signal()* function, and whose values shall compare unequal to the
 10796 address of any declarable function.

10797 `SIG_DFL` Request for default signal handling.

10798 `SIG_ERR` Return value from *signal()* in case of error.

10799 OB CX `SIG_HOLD` Request that signal be held.

10800 `SIG_IGN` Request that signal be ignored.

10801 The <signal.h> header shall define the following data types through **typedef**:

10802 `sig_atomic_t` Possibly volatile-qualified integer type of an object that can be accessed as
 10803 an atomic entity, even in the presence of asynchronous interrupts.

10804 CX `sigset_t` Integer or structure type of an object used to represent sets of signals.

10805 CX `pid_t` As described in <sys/types.h>.

10806 CX The <signal.h> header shall define the **sigevent** structure, which has at least the following
 10807 members:

10808 `int sigev_notify` Notification type.

10809 `int sigev_signo` Signal number.

10810 `union sigval sigev_value` Signal value.

10811 `void (*sigev_notify_function)(union sigval)`
 10812 Notification function.

10813 `pthread_attr_t *sigev_notify_attributes` Notification attributes.

10814 The <signal.h> header shall define the following symbolic constants for the values of
 10815 *sigev_notify*:

10816 `SIGEV_NONE` No asynchronous notification is delivered when the event of interest
 10817 occurs.

10818 `SIGEV_SIGNAL` A queued signal, with an application-defined value, is generated when
 10819 the event of interest occurs.

10820 `SIGEV_THREAD` A notification function is called to perform notification.

10821 The **sigval** union shall be defined as:

10822 `int sival_int` Integer signal value.

10823 `void *sival_ptr` Pointer signal value.

10824 The <signal.h> header shall declare the macros `SIGRTMIN` and `SIGRTMAX`, which shall expand
 10825 to positive integer expressions with type **int**, but which need not be constant expressions. These
 10826 macros specify a range of signal numbers that are reserved for application use and for which the

10827 realtime signal behavior specified in this volume of POSIX.1-200x is supported. The signal
10828 numbers in this range do not overlap any of the signals specified in the following table.

10829 The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal
10830 numbers.

10831 It is implementation-defined whether realtime signal behavior is supported for other signals.

10832 The <signal.h> header shall define the following macros that are used to refer to the signals that
10833 occur in the system. Signals defined here begin with the letters SIG followed by an uppercase
10834 letter. The macros shall expand to positive integer constant expressions with type **int** and
10835 distinct values. The value 0 is reserved for use as the null signal (see *kill()*). Additional
10836 implementation-defined signals may occur in the system.

10837 CX The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIGILL, SIGINT,
10838 SIGSEGV, and SIGTERM to be defined.

10839 The following signals shall be supported on all implementations (default actions are explained
10840 below the table):

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped, or continued.
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.
SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

10871		The default actions are as follows:	
10872		T Abnormal termination of the process.	-
10873	XSI	A Abnormal termination of the process with additional actions.	
10874		I Ignore the signal.	
10875		S Stop the process.	
10876		C Continue the process, if it is stopped; otherwise, ignore the signal.	
10877		The effects on the process in each case are described in XSH Section 2.4.3 (on page 465).	
10878	CX	The <signal.h> header shall provide a declaration of struct sigaction , including at least the following members:	
10879			
10880		void (*sa_handler)(int) Pointer to a signal-catching function	
10881		or one of the SIG_IGN or SIG_DFL.	
10882		sigset_t sa_mask Set of signals to be blocked during execution	
10883		of the signal handling function.	
10884		int sa_flags Special flags.	
10885		void (*sa_sigaction)(int, siginfo_t *, void *)	
10886		Pointer to a signal-catching function.	
10887	XSI	The storage occupied by <i>sa_handler</i> and <i>sa_sigaction</i> may overlap, and a conforming application shall not use both simultaneously.	
10888			
10889		The <signal.h> header shall define the following macros which shall expand to integer constant expressions that need not be usable in #if preprocessing directives:	
10890			
10891	CX	SIG_BLOCK The resulting set is the union of the current set and the signal set pointed to by the argument <i>set</i> .	-
10892			
10893	CX	SIG_UNBLOCK The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument <i>set</i> .	
10894			
10895	CX	SIG_SETMASK The resulting set is the signal set pointed to by the argument <i>set</i> .	
10896		The <signal.h> header shall also define the following symbolic constants:	+
10897	CX	SA_NOCLDSTOP Do not generate SIGCHLD when children stop	+
10898	XSI	or stopped children continue.	
10899	XSI	SA_ONSTACK Causes signal delivery to occur on an alternate stack.	
10900		SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry to signal handlers.	
10901			
10902		SA_RESTART Causes certain functions to become restartable.	
10903		SA_SIGINFO Causes extra information to be passed to signal handlers at the time of receipt of a signal.	
10904			
10905		SA_NOCLDWAIT Causes implementations not to create zombie processes on child death.	
10906		SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.	
10907	XSI	SS_ONSTACK Process is executing on an alternate signal stack.	
10908	XSI	SS_DISABLE Alternate signal stack is disabled.	
10909	XSI	MINSIGSTKSZ Minimum stack size for a signal handler.	
10910	XSI	SIGSTKSZ Default size in bytes for the alternate signal stack.	
10911	CX	The <signal.h> header shall define the mcontext_t type through typedef .	

10912	CX	The <signal.h> header shall define the ucontext_t type as a structure that shall include at least the following members:
10913		
10914		<code>ucontext_t *uc_link</code> Pointer to the context that is resumed
10915		when this context returns.
10916		<code>sigset_t uc_sigmask</code> The set of signals that are blocked when this
10917		context is active.
10918		<code>stack_t uc_stack</code> The stack used by this context.
10919		<code>mcontext_t uc_mcontext</code> A machine-specific representation of the saved
10920		context.
10921		The <signal.h> header shall define the stack_t type as a structure that includes at least the
10922		following members:
10923		<code>void *ss_sp</code> Stack base or pointer.
10924		<code>size_t ss_size</code> Stack size.
10925		<code>int ss_flags</code> Flags.
10926	XSI	The size_t type shall be defined as described in <sys/types.h>.
10927	CX	The <signal.h> header shall define the siginfo_t type as a structure that includes at least the
10928		following members:
10929	CX	<code>int si_signo</code> Signal number.
10930		<code>int si_code</code> Signal code.
10931	XSI	<code>int si_errno</code> If non-zero, an <i>errno</i> value associated with
10932		this signal, as defined in <errno.h>.
10933	CX	<code>pid_t si_pid</code> Sending process ID.
10934		<code>uid_t si_uid</code> Real user ID of sending process.
10935		<code>void *si_addr</code> Address of faulting instruction.
10936		<code>int si_status</code> Exit value or signal.
10937	OB XSR	<code>long si_band</code> Band event for SIGPOLL.
10938	CX	<code>union sigval si_value</code> Signal value.
10939	CX	The <signal.h> header shall define the symbolic constants in the Code column of the following
10940		table for use as values of <i>si_code</i> that are signal-specific or non-signal-specific reasons why the
10941		signal was generated.

	Signal	Code	Reason		
10942	CX	SIGILL	ILL_ILLOPC	Illegal opcode.	
10943			ILL_ILLOPN	Illegal operand.	
10944			ILL_ILLADR	Illegal addressing mode.	
10945			ILL_ILLTRP	Illegal trap.	
10946			ILL_PRVOPC	Privileged opcode.	
10947			ILL_PRVREG	Privileged register.	
10948			ILL_COPROC	Coprocessor error.	
10949			ILL_BADSTK	Internal stack error.	
10950					
10951			SIGFPE	FPE_INTDIV	Integer divide by zero.
10952	FPE_INTOVF	Integer overflow.			
10953	FPE_FLTDIV	Floating-point divide by zero.			
10954	FPE_FLTOVF	Floating-point overflow.			
10955	FPE_FLTUND	Floating-point underflow.			
10956	FPE_FLTRES	Floating-point inexact result.			
10957	FPE_FLTINV	Invalid floating-point operation.			
10958	FPE_FLTSUB	Subscript out of range.			
10959	SIGSEGV	SEGV_MAPERR	Address not mapped to object.		
10960			SEGV_ACCERR	Invalid permissions for mapped object.	
10961	SIGBUS	BUS_ADRALN	Invalid address alignment.		
10962			BUS_ADRERR	Nonexistent physical address.	
10963			BUS_OBJERR	Object-specific hardware error.	
10964	XSI	SIGTRAP	TRAP_BRKPT	Process breakpoint.	
10965			TRAP_TRACE	Process trace trap.	
10966	CX	SIGCHLD	CLD_EXITED	Child has exited.	
10967			CLD_KILLED	Child has terminated abnormally and did not create a core file.	
10968			CLD_DUMPED	Child has terminated abnormally and created a core file.	
10969			CLD_TRAPPED	Traced child has trapped.	
10970			CLD_STOPPED	Child has stopped.	
10971			CLD_CONTINUED	Stopped child has continued.	
10972	OB XSR	SIGPOLL	POLL_IN	Data input available.	
10973			POLL_OUT	Output buffers available.	
10974			POLL_MSG	Input message available.	
10975			POLL_ERR	I/O error.	
10976			POLL_PRI	High priority input available.	
10977			POLL_HUP	Device disconnected.	
10978	CX	Any	SI_USER	Signal sent by <i>kill()</i> .	
10979			SI_QUEUE	Signal sent by the <i>sigqueue()</i> .	
10980			SI_TIMER	Signal generated by expiration of a timer set by <i>timer_settime()</i> .	
10981			SI_ASYNCIO	Signal generated by completion of an asynchronous I/O request.	
10982			SI_MESGQ	Signal generated by arrival of a message on an empty message queue	
10983					
10984					

10985 CX Implementations may support additional *si_code* values not included in this list, may generate
 10986 values included in this list under circumstances other than those described in this list, and may
 10987 contain extensions or limitations that prevent some values from being generated.
 10988 Implementations do not generate a different value from the ones described in this list for
 10989 circumstances described in this list.

10990 CX In addition, the following signal-specific information shall be available:

Signal	Member	Value
SIGILL SIGFPE	void * <i>si_addr</i>	Address of faulting instruction.
SIGSEGV SIGBUS	void * <i>si_addr</i>	Address of faulting memory reference.
SIGCHLD	pid_t <i>si_pid</i> int <i>si_status</i> uid_t <i>si_uid</i>	Child process ID. Exit value or signal. Real user ID of the process that sent the signal.
SIGPOLL	long <i>si_band</i>	Band event for POLL_IN, POLL_OUT, or POLL_MSG

11000 For some implementations, the value of *si_addr* may be inaccurate.

11001 The following shall be declared as functions and may also be defined as macros. Function
11002 prototypes shall be provided.

```

11003 CX int kill(pid_t, int);
11004 XSI int killpg(pid_t, int);
11005 CX void psiginfo(siginfo_t *, const char *);
11006 void psignal(int, const char *);
11007 int pthread_kill(pthread_t, int);
11008 int pthread_sigmask(int, const sigset_t *restrict,
11009 sigset_t *restrict);
11010 int raise(int);
11011 CX int sigaction(int, const struct sigaction *restrict,
11012 struct sigaction *restrict);
11013 int sigaddset(sigset_t *, int);
11014 XSI int sigaltstack(const stack_t *restrict, stack_t *restrict);
11015 CX int sigdelset(sigset_t *, int);
11016 int sigemptyset(sigset_t *);
11017 int sigfillset(sigset_t *);
11018 OB XSI int sighold(int);
11019 int sigignore(int);
11020 int siginterrupt(int, int);
11021 CX int sigismember(const sigset_t *, int);
11022 void (*signal(int, void (*)(int)))(int);
11023 OB XSI int sigpause(int);
11024 CX int sigpending(sigset_t *);
11025 int sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
11026 int sigqueue(pid_t, int, const union sigval);
11027 OB XSI int sigrelse(int);
11028 void (*sigset(int, void (*)(int)))(int);
11029 CX int sigsuspend(const sigset_t *);
11030 int sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
11031 const struct timespec *restrict);
11032 int sigwait(const sigset_t *restrict, int *restrict);
11033 int sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);

```

11034 CX Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The SIGPOLL and SIGPROF signals may be removed in a future version.

SEE ALSO

<errno.h>, <stropts.h>, <sys/types.h>, <time.h>

XSH Section 2.2 (on page 448), *alarm()*, *ioctl()*, *kill*, *killpg()*, *pthread_sigmask()*, *raise()*, *sigaction()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *siginterrupt()*, *sigismember()*, *signal()*, *sigpending()*, *sigqueue()*, *sigsuspend()*, *sigtimedwait()*, *timer_create()*, *wait*, *waitid()*

CHANGE HISTORY

First released in Issue 1.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

The default action for SIGURG is changed from i to iii. The function prototype for *sigmask()* is removed.

Issue 6

The Open Group Corrigendum U035/2 is applied. In the DESCRIPTION, the wording for abnormal termination is clarified.

The Open Group Corrigendum U028/8 is applied, correcting the prototype for the *sigset()* function.

The Open Group Corrigendum U026/3 is applied, correcting the type of the *sigev_notify_function* function member of the **sigevent** structure.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals are now mandated. This is also a FIPS requirement.
- The **pid_t** definition is mandated.

The RT markings are changed to RTS to denote that the semantics are part of the Realtime Signals Extension option.

The **restrict** keyword is added to the prototypes for *sigaction()*, *sigaltstack()*, *sigprocmask()*, *sigtimedwait()*, *sigwait()*, and *sigwaitinfo()*.

IEEE PASC Interpretation 1003.1 #85 is applied, adding the statement that symbols from <time.h> may be made visible when <signal.h> is included.

Extensions beyond the ISO C standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/14 is applied, changing the descriptive text for members of **struct sigaction**.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/15 is applied, correcting the definition of the *sa_sigaction* member of **struct sigaction**.

IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/24 is applied, reworking the ordering of the **siginfo_t** type structure in the DESCRIPTION. This is an editorial change and no normative change is intended.

11079
11080
11081
11082
11083
11084
11085
11086
11087
11088
11089
11090
11091
11092
11093
11094
11095

Issue 7

SD5-XBD-ERN-5 is applied.

SD5-XBD-ERN-39 is applied, removing the **sigstack** structure which should have been removed at the same time as the LEGACY *sigstack()* function.

SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the **size_t** type.

Austin Group Interpretation 1003.1-2001 #034 is applied.

The **ucontext_t** and **mcontext_t** structures are added here from the obsolescent <ucontext.h> header.

The *psiginfo()* and *psignal()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The SIGPOLL and SIGPROF signals and text relating to the XSI STREAMS option are marked obsolescent.

The SA_RESETHAND, SA_RESTART, SA_SIGINFO, SA_NOCLDWAIT, and SA_NODEFER constants are moved from the XSI option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

This reference page is clarified with respect to macros and symbolic constants. +

SIGRTMIN and SIGRTMAX are required to be positive integer expressions.

DRAFT

11096 NAME

11097 spawn.h — spawn (ADVANCED REALTIME)

11098 SYNOPSIS

11099 SPN #include <spawn.h>

11100 DESCRIPTION

11101 The <spawn.h> header shall define the **posix_spawnattr_t** and **posix_spawn_file_actions_t**
11102 types used in performing spawn operations.11103 The <spawn.h> header shall define the following symbolic constants for use as the flags that
11104 may be set in a **posix_spawnattr_t** object using the *posix_spawnattr_setflags()* function:

11105 POSIX_SPAWN_RESETEIDS

11106 POSIX_SPAWN_SETPGROUP

11107 PS POSIX_SPAWN_SETSCHEDPARAM

11108 POSIX_SPAWN_SETSCHEDULER

11109 POSIX_SPAWN_SETSIGDEF

11110 POSIX_SPAWN_SETSIGMASK

11111 The following shall be declared as functions and may also be defined as macros. Function
11112 prototypes shall be provided.

```

11113 int  posix_spawn(pid_t *restrict, const char *restrict,
11114                const posix_spawn_file_actions_t *,
11115                const posix_spawnattr_t *restrict, char *const [restrict],
11116                char *const [restrict]);
11117 int  posix_spawn_file_actions_addclose(posix_spawn_file_actions_t *,
11118                int);
11119 int  posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t *,
11120                int, int);
11121 int  posix_spawn_file_actions_addopen(posix_spawn_file_actions_t *restrict,
11122                int, const char *restrict, int, mode_t);
11123 int  posix_spawn_file_actions_destroy(posix_spawn_file_actions_t *);
11124 int  posix_spawn_file_actions_init(posix_spawn_file_actions_t *);
11125 int  posix_spawnattr_destroy(posix_spawnattr_t *);
11126 int  posix_spawnattr_getflags(const posix_spawnattr_t *restrict,      -
11127                short *restrict);
11128 int  posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict,
11129                pid_t *restrict);
11130 PS  int  posix_spawnattr_getschedparam(const posix_spawnattr_t *restrict,
11131                struct sched_param *restrict);
11132 int  posix_spawnattr_getschedpolicy(const posix_spawnattr_t *restrict,
11133                int *restrict);
11134 int  posix_spawnattr_getsigdefault(const posix_spawnattr_t *restrict,  +
11135                sigset_t *restrict);
11136 int  posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict,    +
11137                sigset_t *restrict);
11138 int  posix_spawnattr_init(posix_spawnattr_t *);
11139 int  posix_spawnattr_setflags(posix_spawnattr_t *, short);          -
11140 int  posix_spawnattr_setpgroup(posix_spawnattr_t *, pid_t);

```



```

11141 PS      int  posix_spawnattr_setschedparam(posix_spawnattr_t *restrict,
11142          const struct sched_param *restrict);
11143          int  posix_spawnattr_setschedpolicy(posix_spawnattr_t *, int);
11144          int  posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict,      +
11145          const sigset_t *restrict);                                          +
11146          int  posix_spawnattr_setsigmask(posix_spawnattr_t *restrict,
11147          const sigset_t *restrict);
11148          int  posix_spawn(pid_t *restrict, const char *restrict,
11149          const posix_spawn_file_actions_t *,
11150          const posix_spawnattr_t *restrict,
11151          char *const [restrict], char *const [restrict]);

```

11152 Inclusion of the <spawn.h> header may make visible symbols defined in the <sched.h>,
 11153 <signal.h>, and <sys/types.h> headers.

11154 APPLICATION USAGE

11155 None.

11156 RATIONALE

11157 None.

11158 FUTURE DIRECTIONS

11159 None.

11160 SEE ALSO

11161 <sched.h>, <semaphore.h>, <signal.h>, <sys/types.h> -

11162 XSH *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*, |
 11163 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*, |
 11164 *posix_spawnattr_getsigmask()*, *posix_spawn()*, *posix_spawn_file_actions_addclose()*, |
 11165 *posix_spawn_file_actions_adddup2()*, *posix_spawn_file_actions_destroy()*

11166 CHANGE HISTORY

11167 First released in Issue 6. Included for alignment with IEEE Std 1003.1d-1999.

11168 The **restrict** keyword is added to the prototypes for *posix_spawn()*,
 11169 *posix_spawn_file_actions_addopen()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
 11170 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
 11171 *posix_spawnattr_getsigmask()*, *posix_spawnattr_setsigdefault()*, *posix_spawnattr_setschedparam()*,
 11172 *posix_spawnattr_setsigmask()*, and *posix_spawnnp()*.

11173 Issue 7

11174 This reference page is clarified with respect to macros and symbolic constants. +

11175 **NAME**

11176 stdarg.h — handle variable argument list

11177 **SYNOPSIS**

```
11178     #include <stdarg.h>
11179
11179     void va_start(va_list ap, argN);
11180     void va_copy(va_list dest, va_list src);
11181     type va_arg(va_list ap, type);
11182     void va_end(va_list ap);
```

11183 **DESCRIPTION**

11184 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11185 conflict between the requirements described here and the ISO C standard is unintentional. This
 11186 volume of POSIX.1-200x defers to the ISO C standard.

11187 The <stdarg.h> header shall contain a set of macros which allows portable functions that accept
 11188 variable argument lists to be written. Functions that have variable argument lists (such as
 11189 *printf()*) but do not use these macros are inherently non-portable, as different systems use
 11190 different argument-passing conventions.

11191 The type **va_list** shall be defined for variables used to traverse the list.

11192 The *va_start()* macro is invoked to initialize *ap* to the beginning of the list before any calls to
 11193 *va_arg()*.

11194 The *va_copy()* macro initializes *dest* as a copy of *src*, as if the *va_start()* macro had been applied
 11195 to *dest* followed by the same sequence of uses of the *va_arg()* macro as had previously been used
 11196 to reach the present state of *src*. Neither the *va_copy()* nor *va_start()* macro shall be invoked to
 11197 reinitialize *dest* without an intervening invocation of the *va_end()* macro for the same *dest*.

11198 The object *ap* may be passed as an argument to another function; if that function invokes the
 11199 *va_arg()* macro with parameter *ap*, the value of *ap* in the calling function is unspecified and shall
 11200 be passed to the *va_end()* macro prior to any further reference to *ap*. The parameter *argN* is the
 11201 identifier of the rightmost parameter in the variable parameter list in the function definition (the
 11202 one just before the *...*). If the parameter *argN* is declared with the **register** storage class, with a
 11203 function type or array type, or with a type that is not compatible with the type that results after
 11204 application of the default argument promotions, the behavior is undefined.

11205 The *va_arg()* macro shall return the next argument in the list pointed to by *ap*. Each invocation
 11206 of *va_arg()* modifies *ap* so that the values of successive arguments are returned in turn. The *type*
 11207 parameter shall be a type name specified such that the type of a pointer to an object that has the
 11208 specified type can be obtained simply by postfixing a *'*'* to type. If there is no actual next
 11209 argument, or if *type* is not compatible with the type of the actual next argument (as promoted
 11210 according to the default argument promotions), the behavior is undefined, except for the
 11211 following cases:

- 11212 • One type is a signed integer type, the other type is the corresponding unsigned integer
 11213 type, and the value is representable in both types.
- 11214 • One type is a pointer to **void** and the other is a pointer to a character type.
- 11215 XSI • Both types are pointers.

11216 Different types can be mixed, but it is up to the routine to know what type of argument is
 11217 expected.

11218 The *va_end()* macro is used to clean up; it invalidates *ap* for use (unless *va_start()* or *va_copy()* is
 11219 invoked again).

11220 Each invocation of the *va_start()* and *va_copy()* macros shall be matched by a corresponding
 11221 invocation of the *va_end()* macro in the same function.

11222 Multiple traversals, each bracketed by *va_start()* ... *va_end()*, are possible.

11223 EXAMPLES

11224 This example is a possible implementation of *execl()*:

```

11225 #include <stdarg.h>
11226 #define MAXARGS 31
11227 /*
11228  * execl is called by
11229  * execl(file, arg1, arg2, ..., (char *)0);
11230  */
11231 int execl(const char *file, const char *args, ...)
11232 {
11233     va_list ap;
11234     char *array[MAXARGS + 1];
11235     int argno = 0;
11236     va_start(ap, args);
11237     while (args != 0 && argno < MAXARGS)
11238     {
11239         array[argno++] = args;
11240         args = va_arg(ap, const char *);
11241     }
11242     array[argno] = (char *) 0;
11243     va_end(ap);
11244     return execv(file, array);
11245 }
```

11246 APPLICATION USAGE

11247 It is up to the calling routine to communicate to the called routine how many arguments there
 11248 are, since it is not always possible for the called routine to determine this in any other way. For
 11249 example, *execl()* is passed a null pointer to signal the end of the list. The *printf()* function can tell
 11250 how many arguments are there by the *format* argument.

11251 RATIONALE

11252 None.

11253 FUTURE DIRECTIONS

11254 None.

11255 SEE ALSO

11256 XSH *exec*, *printf*

11257 CHANGE HISTORY

11258 First released in Issue 4. Derived from the ANSI C standard.

11259 Issue 6

11260 This reference page is updated to align with the ISO/IEC 9899:1999 standard.

11261 **NAME**

11262 stdbool.h — boolean type and values

11263 **SYNOPSIS**

11264 #include <stdbool.h>

11265 **DESCRIPTION**

11266 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11267 conflict between the requirements described here and the ISO C standard is unintentional. This
 11268 volume of POSIX.1-200x defers to the ISO C standard.

11269 The <stdbool.h> header shall define the following macros:

11270 bool Expands to **_Bool**.

11271 true Expands to the integer constant 1.

11272 false Expands to the integer constant 0.

11273 __bool_true_false_are_defined

11274 Expands to the integer constant 1.

11275 An application may undefine and then possibly redefine the macros bool, true, and false.

11276 **APPLICATION USAGE**

11277 None.

11278 **RATIONALE**

11279 None.

11280 **FUTURE DIRECTIONS**

11281 The ability to undefine and redefine the macros bool, true, and false is an obsolescent feature
 11282 and may be removed in a future version.

11283 **SEE ALSO**

11284 None.

11285 **CHANGE HISTORY**

11286 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

11287 **NAME**

11288 stddef.h — standard type definitions

11289 **SYNOPSIS**

11290 #include <stddef.h>

11291 **DESCRIPTION**

11292 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 11293 conflict between the requirements described here and the ISO C standard is unintentional. This
 11294 volume of POSIX.1-200x defers to the ISO C standard.

11295 The <stddef.h> header shall define the following macros:

11296 CX **NULL** Null pointer constant. The macro shall expand to an integer constant expression +
 11297 with the value 0 cast to type **void ***.

11298 offsetof(*type*, *member-designator*)

11299 Integer constant expression of type **size_t**, the value of which is the offset in bytes
 11300 to the structure member (*member-designator*), from the beginning of its structure
 11301 (*type*).

11302 The <stddef.h> header shall define the following types:

11303 **ptrdiff_t** Signed integer type of the result of subtracting two pointers.

11304 **wchar_t** Integer type whose range of values can represent distinct wide-character codes for
 11305 all members of the largest character set specified among the locales supported by
 11306 the compilation environment: the null character has the code value 0 and each
 11307 member of the portable character set has a code value equal to its value when used
 11308 as the lone character in an integer character constant.

11309 **size_t** Unsigned integer type of the result of the *sizeof* operator.

11310 The implementation shall support one or more programming environments in which the widths
 11311 of **ptrdiff_t**, **size_t**, and **wchar_t** are no greater than the width of type **long**. The names of these
 11312 programming environments can be obtained using the *confstr()* function or the *getconf* utility.

11313 **APPLICATION USAGE**

11314 None.

11315 **RATIONALE**

11316 The ISO C standard does not require the **NULL** macro to include the cast to type **void *** and |
 11317 specifies that the **NULL** macro be implementation-defined. POSIX.1-200x requires the cast and |
 11318 therefore need not be implementation-defined.

11319 **FUTURE DIRECTIONS**

11320 None.

11321 **SEE ALSO**

11322 <sys/types.h>, <wchar.h> -

11323 XSH *confstr()* |11324 XCU *getconf*11325 **CHANGE HISTORY**

11326 First released in Issue 4. Derived from the ANSI C standard.

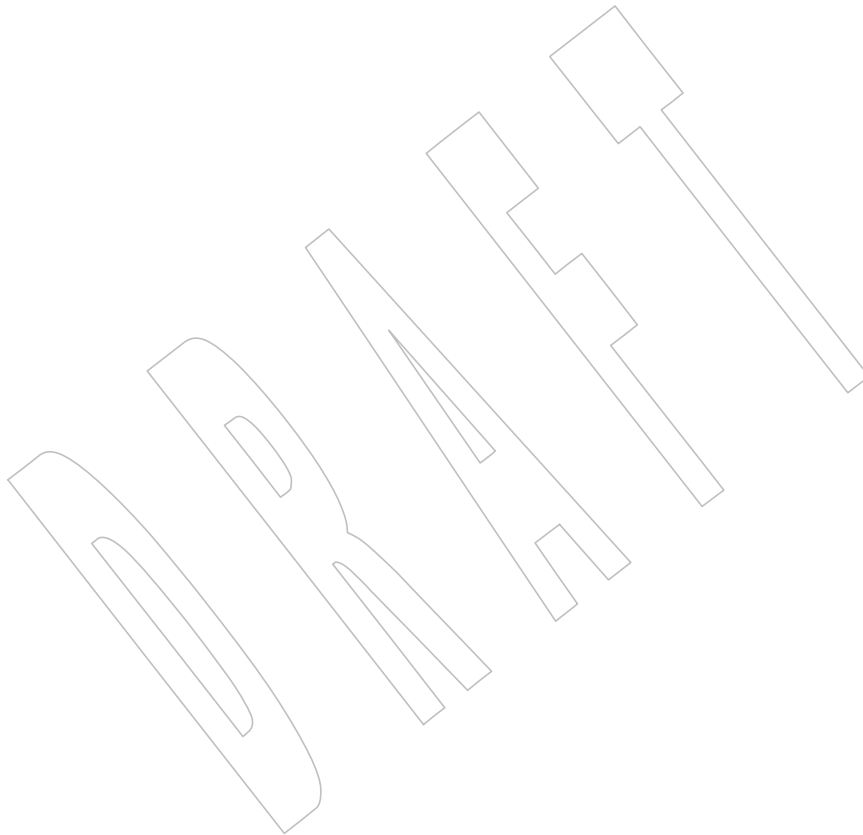
11327

Issue 7

11328

This reference page is clarified with respect to macros and symbolic constants.

+



11329 **NAME**

11330 stdint.h — integer types

11331 **SYNOPSIS**

11332 #include <stdint.h>

11333 **DESCRIPTION**

11334 **CX** Some of the functionality described on this reference page extends the ISO C standard.
 11335 Applications shall define the appropriate feature test macro (see XSH [Section 2.2](#), on page 448) to
 11336 enable the visibility of these symbols in this header.

11337 The <stdint.h> header shall declare sets of integer types having specified widths, and shall
 11338 define corresponding sets of macros. It shall also define macros that specify limits of integer
 11339 types corresponding to types defined in other standard headers.

11340 **Note:** The “width” of an integer type is the number of bits used to store its value in a pure binary
 11341 system; the actual type may use more bits than that (for example, a 28-bit type could be stored
 11342 in 32 bits of actual storage). An N -bit signed type has values in the range -2^{N-1} or $1-2^{N-1}$ to
 11343 $2^{N-1}-1$, while an N -bit unsigned type has values in the range 0 to 2^N-1 .

11344 Types are defined in the following categories:

- 11345 • Integer types having certain exact widths
- 11346 • Integer types having at least certain specified widths
- 11347 • Fastest integer types having at least certain specified widths
- 11348 • Integer types wide enough to hold pointers to objects
- 11349 • Integer types having greatest width

11350 (Some of these types may denote the same type.)

11351 Corresponding macros specify limits of the declared types and construct suitable constants.

11352 For each type described herein that the implementation provides, the <stdint.h> header shall
 11353 declare that **typedef** name and define the associated macros. Conversely, for each type described
 11354 herein that the implementation does not provide, the <stdint.h> header shall not declare that
 11355 **typedef** name, nor shall it define the associated macros. An implementation shall provide those
 11356 types described as required, but need not provide any of the others (described as optional).

11357 **Integer Types**

11358 When **typedef** names differing only in the absence or presence of the initial u are defined, they
 11359 shall denote corresponding signed and unsigned types as described in the ISO/IEC 9899:1999
 11360 standard, Section 6.2.5; an implementation providing one of these corresponding types shall also
 11361 provide the other.

11362 In the following descriptions, the symbol N represents an unsigned decimal integer with no
 11363 leading zeros (for example, 8 or 24, but not 04 or 048).

- 11364 • Exact-width integer types

11365 The **typedef** name **int N _t** designates a signed integer type with width N , no padding bits,
 11366 and a two’s-complement representation. Thus, **int8_t** denotes a signed integer type with a
 11367 width of exactly 8 bits.

11368 The **typedef** name **uint N _t** designates an unsigned integer type with width N . Thus,
 11369 **uint24_t** denotes an unsigned integer type with a width of exactly 24 bits.

11370	CX	The following types are required:
11371		int8_t
11372		int16_t
11373		int32_t
11374		uint8_t
11375		uint16_t
11376		uint32_t
11377		If an implementation provides integer types with width 64 that meet these requirements,
11378		then the following types are required:
11379		int64_t
11380		uint64_t
11381	CX	In particular, this will be the case if any of the following are true:
11382		— The implementation supports the <code>_POSIX_V7_ILP32_OFFBIG</code> programming
11383		environment and the application is being built in the <code>_POSIX_V7_ILP32_OFFBIG</code>
11384		programming environment (see the Shell and Utilities volume of POSIX.1-200x, c99,
11385		Programming Environments).
11386		— The implementation supports the <code>_POSIX_V7_LP64_OFF64</code> programming
11387		environment and the application is being built in the <code>_POSIX_V7_LP64_OFF64</code>
11388		programming environment.
11389		— The implementation supports the <code>_POSIX_V7_LPBIG_OFFBIG</code> programming
11390		environment and the application is being built in the <code>_POSIX_V7_LPBIG_OFFBIG</code>
11391		programming environment.
11392		All other types of this form are optional.
11393		• Minimum-width integer types
11394		◁ The typedef name int_leastN_t designates a signed integer type with a width of at least <i>N</i> ,
11395		such that no signed integer type with lesser size has at least the specified width. Thus,
11396		int_least32_t denotes a signed integer type with a width of at least 32 bits.
11397		The typedef name uint_leastN_t designates an unsigned integer type with a width of at
11398		least <i>N</i> , such that no unsigned integer type with lesser size has at least the specified width.
11399		Thus, uint_least16_t denotes an unsigned integer type with a width of at least 16 bits.
11400		The following types are required:
11401		int_least8_t
11402		int_least16_t
11403		int_least32_t
11404		int_least64_t
11405		uint_least8_t
11406		uint_least16_t
11407		uint_least32_t
11408		uint_least64_t
11409		All other types of this form are optional.
11410		• Fastest minimum-width integer types
11411		Each of the following types designates an integer type that is usually fastest to operate
11412		with among all integer types that have at least the specified width.

11413 The designated type is not guaranteed to be fastest for all purposes; if the implementation
 11414 has no clear grounds for choosing one type over another, it will simply pick some integer
 11415 type satisfying the signedness and width requirements.

11416 The **typedef** name **int_fastN_t** designates the fastest signed integer type with a width of at
 11417 least *N*. The **typedef** name **uint_fastN_t** designates the fastest unsigned integer type with
 11418 a width of at least *N*.

11419 The following types are required:

11420 **int_fast8_t**
 11421 **int_fast16_t**
 11422 **int_fast32_t**
 11423 **int_fast64_t**
 11424 **uint_fast8_t**
 11425 **uint_fast16_t**
 11426 **uint_fast32_t**
 11427 **uint_fast64_t**

11428 All other types of this form are optional.

- 11429 • Integer types capable of holding object pointers

11430 The following type designates a signed integer type with the property that any valid
 11431 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11432 the result will compare equal to the original pointer:

11433 **intptr_t**

11434 The following type designates an unsigned integer type with the property that any valid
 11435 pointer to **void** can be converted to this type, then converted back to a pointer to **void**, and
 11436 the result will compare equal to the original pointer:

11437 **uintptr_t**

11438 XSI On XSI-conformant systems, the **intptr_t** and **uintptr_t** types are required; otherwise, they
 11439 are optional.

- 11440 • Greatest-width integer types

11441 The following type designates a signed integer type capable of representing any value of
 11442 any signed integer type:

11443 **intmax_t**

11444 The following type designates an unsigned integer type capable of representing any value
 11445 of any unsigned integer type:

11446 **uintmax_t**

11447 These types are required.

11448 **Note:** Applications can test for optional types by using the corresponding limit macro from [Limits of](#)
 11449 [Specified-Width Integer Types](#) (on page 330).

11450 **Limits of Specified-Width Integer Types**

11451 The following macros specify the minimum and maximum limits of the types declared in the
 11452 <stdint.h> header. Each macro name corresponds to a similar type name in [Integer Types](#) (on
 11453 page 327).

11454 Each instance of any defined macro shall be replaced by a constant expression suitable for use in
 11455 #if preprocessing directives, and this expression shall have the same type as would an
 11456 expression that is an object of the corresponding type converted according to the integer
 11457 promotions. Its implementation-defined value shall be equal to or greater in magnitude
 11458 (absolute value) than the corresponding value given below, with the same sign, except where
 11459 stated to be exactly the given value.

11460 • Limits of exact-width integer types

11461 — Minimum values of exact-width signed integer types:

11462 {INTN_MIN} Exactly $-(2^{N-1})$

11463 — Maximum values of exact-width signed integer types:

11464 {INTN_MAX} Exactly $2^{N-1}-1$

11465 — Maximum values of exact-width unsigned integer types:

11466 {UINTN_MAX} Exactly 2^N-1

11467 • Limits of minimum-width integer types

11468 — Minimum values of minimum-width signed integer types:

11469 {INT_LEASTN_MIN} $-(2^{N-1}-1)$

11470 — Maximum values of minimum-width signed integer types:

11471 {INT_LEASTN_MAX} $2^{N-1}-1$

11472 — Maximum values of minimum-width unsigned integer types:

11473 {UINT_LEASTN_MAX} 2^N-1

11474 • Limits of fastest minimum-width integer types

11475 — Minimum values of fastest minimum-width signed integer types:

11476 {INT_FASTN_MIN} $-(2^{N-1}-1)$

11477 — Maximum values of fastest minimum-width signed integer types:

11478 {INT_FASTN_MAX} $2^{N-1}-1$

11479 — Maximum values of fastest minimum-width unsigned integer types:

11480 {UINT_FASTN_MAX} 2^N-1

11481 • Limits of integer types capable of holding object pointers

11482 — Minimum value of pointer-holding signed integer type:

11483 {INTPTR_MIN} $-(2^{15}-1)$

11484 — Maximum value of pointer-holding signed integer type:

11485 {INTPTR_MAX} $2^{15}-1$

11486 — Maximum value of pointer-holding unsigned integer type:

11487 {UINTPTR_MAX} $2^{16} - 1$

11488 • Limits of greatest-width integer types

11489 — Minimum value of greatest-width signed integer type:

11490 {INTMAX_MIN} $-(2^{63} - 1)$

11491 — Maximum value of greatest-width signed integer type:

11492 {INTMAX_MAX} $2^{63} - 1$

11493 — Maximum value of greatest-width unsigned integer type:

11494 {UINTMAX_MAX} $2^{64} - 1$

11495 Limits of Other Integer Types

11496 The following macros specify the minimum and maximum limits of integer types corresponding
11497 to types defined in other standard headers.

11498 Each instance of these macros shall be replaced by a constant expression suitable for use in **#if**
11499 preprocessing directives, and this expression shall have the same type as would an expression
11500 that is an object of the corresponding type converted according to the integer promotions. Its
11501 implementation-defined value shall be equal to or greater in magnitude (absolute value) than
11502 the corresponding value given below, with the same sign.

- 11503 • Limits of **ptrdiff_t**:

11504 {PTRDIFF_MIN} -65 535

11505 {PTRDIFF_MAX} +65 535

- 11506 • Limits of **sig_atomic_t**:

11507 {SIG_ATOMIC_MIN} See below.

11508 {SIG_ATOMIC_MAX} See below.

- 11509 • Limit of **size_t**:

11510 {SIZE_MAX} 65 535

- 11511 • Limits of **wchar_t**:

11512 {WCHAR_MIN} See below.

11513 {WCHAR_MAX} See below.

- 11514 • Limits of **wint_t**:

11515 {WINT_MIN} See below.

11516 {WINT_MAX} See below.

11517 If **sig_atomic_t** (see the <signal.h> header) is defined as a signed integer type, the value of
11518 {SIG_ATOMIC_MIN} shall be no greater than -127 and the value of {SIG_ATOMIC_MAX} shall
11519 be no less than 127; otherwise, **sig_atomic_t** shall be defined as an unsigned integer type, and
11520 the value of {SIG_ATOMIC_MIN} shall be 0 and the value of {SIG_ATOMIC_MAX} shall be no
11521 less than 255.

11522 If **wchar_t** (see the <stddef.h> header) is defined as a signed integer type, the value of
11523 {WCHAR_MIN} shall be no greater than -127 and the value of {WCHAR_MAX} shall be no less
11524 than 127; otherwise, **wchar_t** shall be defined as an unsigned integer type, and the value of
11525 {WCHAR_MIN} shall be 0 and the value of {WCHAR_MAX} shall be no less than 255.

11526 If **wint_t** (see the <wchar.h> header) is defined as a signed integer type, the value of

11527 {WINT_MIN} shall be no greater than -32767 and the value of {WINT_MAX} shall be no less
 11528 than 32767 ; otherwise, **wint_t** shall be defined as an unsigned integer type, and the value of
 11529 {WINT_MIN} shall be 0 and the value of {WINT_MAX} shall be no less than 65535 .

11530 **Macros for Integer Constant Expressions**

11531 The following macros expand to integer constant expressions suitable for initializing objects that
 11532 have integer types corresponding to types defined in the <stdint.h> header. Each macro name
 11533 corresponds to a similar type name listed under *Minimum-width integer types* and *Greatest-width*
 11534 *integer types*.

11535 Each invocation of one of these macros shall expand to an integer constant expression suitable
 11536 for use in **#if** preprocessing directives. The type of the expression shall have the same type as
 11537 would an expression that is an object of the corresponding type converted according to the
 11538 integer promotions. The value of the expression shall be that of the argument.

11539 The argument in any instance of these macros shall be a decimal, octal, or hexadecimal constant
 11540 with a value that does not exceed the limits for the corresponding type.

- 11541 • Macros for minimum-width integer constant expressions

11542 The macro *INTN_C(value)* shall expand to an integer constant expression corresponding to
 11543 the type **int_leastN_t**. The macro *UINTN_C(value)* shall expand to an integer constant
 11544 expression corresponding to the type **uint_leastN_t**. For example, if **uint_least64_t** is a
 11545 name for the type **unsigned long long**, then *UINT64_C(0x123)* might expand to the integer
 11546 constant `0x123ULL`.

- 11547 • Macros for greatest-width integer constant expressions

11548 The following macro expands to an integer constant expression having the value specified
 11549 by its argument and the type **intmax_t**:

11550 *INTMAX_C(value)*

11551 The following macro expands to an integer constant expression having the value specified
 11552 by its argument and the type **uintmax_t**:

11553 *UINTMAX_C(value)*

11554 **APPLICATION USAGE**

11555 None.

11556 **RATIONALE**

11557 The <stdint.h> header is a subset of the <inttypes.h> header more suitable for use in
 11558 freestanding environments, which might not support the formatted I/O functions. In some
 11559 environments, if the formatted conversion support is not wanted, using this header instead of
 11560 the <inttypes.h> header avoids defining such a large number of macros.

11561 As a consequence of adding **int8_t**, the following are true:

- 11562 • A byte is exactly 8 bits.
- 11563 • {CHAR_BIT} has the value 8, {SCHAR_MAX} has the value 127, {SCHAR_MIN} has the
 11564 value -128 , and {UCHAR_MAX} has the value 255.

11565 (The POSIX standard explicitly requires 8-bit char and two's-complement arithmetic.)

11566 **FUTURE DIRECTIONS**

11567 **typedef** names beginning with **int** or **uint** and ending with **_t** may be added to the types defined
 11568 in the <stdint.h> header. Macro names beginning with **INT** or **UINT** and ending with **_MAX**,
 11569 **_MIN**, or **_C** may be added to the macros defined in the <stdint.h> header.

11570	SEE ALSO		
11571		<inttypes.h> , <signal.h> , <stddef.h> , <wchar.h>	
11572		XSH Section 2.2 (on page 448)	+
11573	CHANGE HISTORY		
11574		First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.	
11575		ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.	
11576	Issue 7		
11577		SD5-XBD-ERN-67 is applied.	+



11578 **NAME**11579 `stdio.h` — standard buffered input/output11580 **SYNOPSIS**11581 `#include <stdio.h>`11582 **DESCRIPTION**

11583 CX Some of the functionality described on this reference page extends the ISO C standard.
 11584 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 11585 enable the visibility of these symbols in this header.

11586 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11587 expressions:

11588 CX **BUFSIZ** Size of <stdio.h> buffers. This shall expand to a positive value.

11589 CX **L_ctermid** Maximum size of character array to hold `ctermid()` output.

11590 OB **L_tmpnam** Maximum size of character array to hold `tmpnam()` output.

11591 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11592 expressions with distinct values:

11593 **_IOFBF** Input/output fully buffered.

11594 **_IOLBF** Input/output line buffered.

11595 **_IONBF** Input/output unbuffered.

11596 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11597 expressions with distinct values:

11598 **SEEK_CUR** Seek relative to current position.

11599 **SEEK_END** Seek relative to end-of-file.

11600 **SEEK_SET** Seek relative to start-of-file.

11601 The <stdio.h> header shall define the following macros which shall expand to integer constant
 11602 expressions denoting implementation limits:

11603 **{FILENAME_MAX}** Maximum size in bytes of the longest filename string that the
 11604 implementation guarantees can be opened.

11605 **{FOPEN_MAX}** Number of streams which the implementation guarantees can be open
 11606 simultaneously. The value is at least eight.

11607 OB **{TMP_MAX}** Minimum number of unique filenames generated by `tmpnam()`.
 11608 Maximum number of times an application can call `tmpnam()` reliably. The
 11609 value of **{TMP_MAX}** is at least 25.

11610 OB XSI On XSI-conformant systems, the value of **{TMP_MAX}** is at least 10 000.

11611 The <stdio.h> header shall define the following macro which shall expand to an integer constant
 11612 expression with type **int** and a negative value:

11613 **EOF** End-of-file return value.

11614 The <stdio.h> header shall define **NULL** as described in <stddef.h>.

11615 The <stdio.h> header shall define the following macro which shall expand to a string constant:

11616	OB XSI	P_tmpdir	Default directory prefix for <i>tempnam()</i> .
11617			The <stdio.h> header shall define the following macros which shall expand to expressions of
11618			type “pointer to FILE ” that point to the FILE objects associated, respectively, with the standard
11619			error, input, and output streams:
11620		<i>stderr</i>	Standard error output stream.
11621		<i>stdin</i>	Standard input stream.
11622		<i>stdout</i>	Standard output stream.
11623			The following data types shall be defined through typedef :
11624		FILE	A structure containing information about a file.
11625		fpos_t	A non-array type containing all information needed to specify uniquely
11626			every position within a file.
11627	CX	va_list	As described in <stdarg.h>.
11628		size_t	As described in <stddef.h>.
11629	CX	ssize_t	As described in <sys/types.h>.
11630			The following shall be declared as functions and may also be defined as macros. Function
11631			prototypes shall be provided.
11632		void	clearerr(FILE *);
11633	CX	char	*ctermid(char *);
11634		int	dprintf(int, const char *, ...)
11635		int	fclose(FILE *);
11636	CX	FILE	*fdopen(int, const char *);
11637		int	feof(FILE *);
11638		int	ferror(FILE *);
11639		int	fflush(FILE *);
11640		int	fgetc(FILE *);
11641		int	fgetpos(FILE *restrict, fpos_t *restrict);
11642		char	*fgets(char *restrict, int, FILE *restrict);
11643	CX	int	fileno(FILE *);
11644		void	flockfile(FILE *);
11645		FILE	*fmemopen(void *restrict, size_t, const char *restrict);
11646		FILE	*fopen(const char *restrict, const char *restrict);
11647		int	fprintf(FILE *restrict, const char *restrict, ...);
11648		int	fputc(int, FILE *);
11649		int	fputs(const char *restrict, FILE *restrict);
11650		size_t	fread(void *restrict, size_t, size_t, FILE *restrict);
11651		FILE	*freopen(const char *restrict, const char *restrict,
11652			FILE *restrict);
11653		int	fscanf(FILE *restrict, const char *restrict, ...);
11654		int	fseek(FILE *, long, int);
11655	CX	int	fseeko(FILE *, off_t, int);
11656		int	fsetpos(FILE *, const fpos_t *);
11657		long	ftell(FILE *);
11658	CX	off_t	ftello(FILE *);
11659		int	ftrylockfile(FILE *);
11660		void	funlockfile(FILE *);
11661		size_t	fwrite(const void *restrict, size_t, size_t, FILE *restrict);
11662		int	getc(FILE *);
11663		int	getchar(void);

<stdio.h>

Headers

```

11664 CX      int      getc_unlocked(FILE *);
11665         int      getchar_unlocked(void);
11666         ssize_t  getdelim(char **, size_t *, int, FILE *);
11667         ssize_t  getline(char **, size_t *, FILE *);
11668 OB      char      *gets(char *);
11669 CX      FILE      *open_memstream(char **, size_t *);
11670         int      pclose(FILE *);
11671         void     perror(const char *);
11672 CX      FILE      *popen(const char *, const char *);
11673         int      printf(const char *restrict, ...);
11674         int      putc(int, FILE *);
11675         int      putchar(int);
11676 CX      int      putc_unlocked(int, FILE *);
11677         int      putchar_unlocked(int);
11678         int      puts(const char *);
11679         int      remove(const char *);
11680         int      rename(const char *, const char *);
11681 CX      int      renameat(int, const char *, int, const char *);
11682         void     rewind(FILE *);
11683         int      scanf(const char *restrict, ...);
11684         void     setbuf(FILE *restrict, char *restrict);
11685         int      setvbuf(FILE *restrict, char *restrict, int, size_t);
11686         int      snprintf(char *restrict, size_t, const char *restrict, ...);
11687         int      sprintf(char *restrict, const char *restrict, ...);
11688         int      sscanf(const char *restrict, const char *restrict, ...);
11689 OB XSI   char      *tempnam(const char *, const char *);
11690         FILE      *tmpfile(void);
11691 OB      char      *tmpnam(char *);
11692         int      ungetc(int, FILE *);
11693         int      vfprintf(FILE *restrict, const char *restrict, va_list);
11694         int      vfscanf(FILE *restrict, const char *restrict, va_list);
11695         int      vprintf(const char *restrict, va_list);
11696         int      vscanf(const char *restrict, va_list);
11697         int      vsnprintf(char *restrict, size_t, const char *restrict,
11698             va_list);
11699         int      vsprintf(char *restrict, const char *restrict, va_list);
11700         int      vsscanf(const char *restrict, const char *restrict, va_list);
11701 CX      Inclusion of the <stdio.h> header may also make visible all symbols from <stddef.h>.

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO[<stdarg.h>](#), [<stddef.h>](#), [<sys/types.h>](#)

```

11710 XSH Section 2.2 (on page 448), clearerr(), ctermid(), fclose(), fdopen(), fgetc(), fgetpos(), ferror(),
11711 feof(), fflush(), fgets(), fileno(), flockfile(), fopen(), fputc(), fputs(), fread(), freopen(), fscanf(),
11712 fseek(), fsetpos(), ftell(), fwrite(), getc(), getc_unlocked(), getwchar(), getchar(), getopt(), gets(),
11713 pclose(), perror(), popen(), printf(), putc(), putchar(), puts(), putwchar(), remove(), rename(),
11714 rewind(), setbuf(), setvbuf(), stdin, system(), tempnam(), tmpfile(), tmpnam(), ungetc(), vfprintf(),

```


11715
11716
11717
11718
11719
11720
11721
11722
11723
11724
11725
11726
11727
11728
11729
11730
11731
11732
11733
11734
11735
11736
11737
11738
11739
11740

vfscanf()

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Large File System extensions are added.

The constant `L_cuserid` and the external variables *optarg*, *opterr*, *optind*, and *optopt* are marked as extensions and LEGACY.

The *cuserid()* and *getopt()* functions are marked LEGACY.

Issue 6

The constant `L_cuserid` and the external variables *optarg*, *opterr*, *optind*, and *optopt* are removed as they were previously marked LEGACY.

The *cuserid()*, *getopt()*, and *getw()* functions are removed as they were previously marked LEGACY.

Several functions are marked as part of the Thread-Safe Functions option.

This reference page is updated to align with the ISO/IEC 9899:1999 standard. Note that the description of the `fpos_t` type is now explicitly updated to exclude array types.

Extensions beyond the ISO C standard are marked.

Issue 7

The *dprintf()*, *fmemopen()*, *getdelim()*, *getline()*, and *open_memstream()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *renameat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

The *gets()*, *tmpnam()*, and *tempnam()* functions and the `L_tmpnam` macro are marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

+

11741	NAME			
11742		stdlib.h	— standard library definitions	
11743	SYNOPSIS			
11744		#include	<stdlib.h>	
11745	DESCRIPTION			
11746	CX	Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to enable the visibility of these symbols in this header.		
11747				
11748				
11749		The <stdlib.h> header shall define the following macros which shall expand to integer constant expressions:		
11750				
11751		EXIT_FAILURE	Unsuccessful termination for <i>exit()</i> ; evaluates to a non-zero value.	
11752		EXIT_SUCCESS	Successful termination for <i>exit()</i> ; evaluates to 0.	-
11753		{RAND_MAX}	Maximum value returned by <i>rand()</i> ; at least 32767.	
11754		The <stdlib.h> header shall define the following macro which shall expand to a positive integer expression with type size_t :		
11755				+
11756		{MB_CUR_MAX}	Maximum number of bytes in a character specified by the current locale (category <i>LC_CTYPE</i>).	
11757				
11758		The <stdlib.h> header shall define NULL as described in <stddef.h>.		
11759				+
11759		The following data types shall be defined through typedef :		
11760		div_t	Structure type returned by the <i>div()</i> function.	
11761		ldiv_t	Structure type returned by the <i>ldiv()</i> function.	
11762		lldiv_t	Structure type returned by the <i>lldiv()</i> function.	
11763		size_t	As described in <stddef.h>.	
11764		wchar_t	As described in <stddef.h>.	
11765	CX	In addition, the following symbolic constants and macros shall be defined as in <sys/wait.h>:		
11766		WEXITSTATUS		
11767		WIFEXITED		-
11768		WIFSIGNALED		
11769		WIFSTOPPED		
11770		WNOHANG		+
11771		WSTOPSIG		
11772		WTERMSIG		
11773		WUNTRACED		+
11774		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.		
11775				
11776		void	_Exit (int);	
11777	XSI	long	a64l (const char *);	
11778		void	abort (void);	
11779		int	abs (int);	
11780		int	atexit (void (*)(void));	
11781		double	atof (const char *);	

```

11782         int          atoi(const char *);
11783         long         atol(const char *);
11784         long long    atoll(const char *);
11785         void         *bsearch(const void *, const void *, size_t, size_t,
11786                               int (*)(const void *, const void *));
11787         void         *calloc(size_t, size_t);
11788         div_t        div(int, int);
11789     XSI         double   drand48(void);
11790         double       erand48(unsigned short [3]);
11791         void         exit(int);
11792         void         free(void *);
11793         char         *getenv(const char *);
11794         int          getsuopt(char **, char *const *, char **);
11795     XSI         int          grantpt(int);
11796         char         *initstate(unsigned, char *, size_t);
11797         long         jrand48(unsigned short [3]);
11798         char         *l64a(long);
11799         long         labs(long);
11800     XSI         void       lcong48(unsigned short [7]);
11801         ldiv_t       ldiv(long, long);
11802         long long    llabs(long long);
11803         lldiv_t      lldiv(long long, long long);
11804     XSI         long       lrand48(void);
11805         void         *malloc(size_t);
11806         int          mblen(const char *, size_t);
11807         size_t       mbstowcs(wchar_t *restrict, const char *restrict, size_t);
11808         int          mbtowc(wchar_t *restrict, const char *restrict, size_t);
11809     CX         char         *mkdtemp(char *);
11810         int          mkstemp(char *);
11811     XSI         long       mrand48(void);
11812         long         nrand48(unsigned short [3]);
11813     ADV         int          posix_memalign(void **, size_t, size_t);
11814     XSI         int          posix_openpt(int);
11815         char         *ptsname(int);
11816         int          putenv(char *);
11817         void         qsort(void *, size_t, size_t, int (*)(const void *,
11818                                                           const void *));
11819         int          rand(void);
11820     OB CX      int          rand_r(unsigned *);
11821     XSI         long       random(void);
11822         void         *realloc(void *, size_t);
11823     XSI         char         *realpath(const char *restrict, char *restrict);
11824         unsigned short *seed48(unsigned short [3]);
11825     CX         int          setenv(const char *, const char *, int);
11826     XSI         void         setkey(const char *);
11827         char         *setstate(const char *);
11828         void         srand(unsigned);
11829     XSI         void         srand48(long);
11830         void         srandom(unsigned);
11831         double       strtod(const char *restrict, char **restrict);
11832         float        strtodf(const char *restrict, char **restrict);
11833         long         strtol(const char *restrict, char **restrict, int);
11834         long double  strtold(const char *restrict, char **restrict);
11835         long long    strtoll(const char *restrict, char **restrict, int);

```

```

11836         unsigned long strtoul(const char *restrict, char **restrict, int);
11837         unsigned long long
11838             strtoull(const char *restrict, char **restrict, int);
11839         int
11840     XSI     int      system(const char *);
11841     CX     int      unsetenv(const char *);
11842         size_t      wcstombs(char *restrict, const wchar_t *restrict, size_t);
11843         int          wctomb(char *, wchar_t);

```

11844 CX Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>, <limits.h>, <math.h>, and <sys/wait.h>.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

<limits.h>, <math.h>, <stddef.h>, <sys/types.h>, <sys/wait.h>

XSH Section 2.2 (on page 448), *_Exit()*, *a64l()*, *abort()*, *abs()*, *atexit()*, *atof()*, *atoi()*, *atol()*, *bsearch()*, *calloc()*, *div()*, *drand48()*, *exit()*, *free()*, *getenv()*, *getsubopt()*, *grantpt()*, *initstate()*, *labs()*, *ldiv()*, *malloc()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *mkdtemp()*, *posix_memalign()*, *ptsname()*, *putenv()*, *qsort()*, *rand()*, *realloc()*, *realpath()*, *strtod()*, *strtol()*, *strtoul()*, *unlockpt()*, *wcstombs()*, *wctomb()*

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *ttyslot()* and *valloc()* functions are marked LEGACY.

The type of the third argument to *initstate()* is changed from **int** to **size_t**. The type of the return value from *setstate()* is changed from **char** to **char ***, and the type of the first argument is changed from **char *** to **const char ***.

Issue 6

The Open Group Corrigendum U021/1 is applied, correcting the prototype for *realpath()* to be consistent with the reference page.

The Open Group Corrigendum U028/13 is applied, correcting the prototype for *putenv()* to be consistent with the reference page.

The *rand_r()* function is marked as part of the Thread-Safe Functions option.

Function prototypes for *setenv()* and *unsetenv()* are added.

The *posix_memalign()* function is added for alignment with IEEE Std 1003.1d-1999.

This reference page is updated to align with the ISO/IEC 9899:1999 standard.

The *ecvt()*, *fcvt()*, *gcvt()*, and *mktemp()* functions are marked LEGACY.

The *ttyslot()* and *valloc()* functions are removed as they were previously marked LEGACY.

Extensions beyond the ISO C standard are marked.

11878	Issue 7	
11879	SD5-XBD-ERN-79 is applied.	+
11880	The LEGACY functions are removed.	
11881	The <i>mkdtemp()</i> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.	
11882		
11883	The <i>rand_r()</i> function is marked obsolescent.	+
11884	This reference page is clarified with respect to macros and symbolic constants.	



11885 NAME

11886 string.h — string operations

11887 SYNOPSIS

11888 #include <string.h>

11889 DESCRIPTION

11890 CX Some of the functionality described on this reference page extends the ISO C standard.
 11891 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 11892 enable the visibility of these symbols in this header.

11893 The <string.h> header shall define NULL and size_t as described in <stddef.h>.

11894 The following shall be declared as functions and may also be defined as macros. Function
 11895 prototypes shall be provided for use with ISO C standard compilers.

```

11896 XSI void *memcpy(void *restrict, const void *restrict, int, size_t);
11897 void *memchr(const void *, int, size_t);
11898 int memcmp(const void *, const void *, size_t);
11899 void *memcpy(void *restrict, const void *restrict, size_t);
11900 void *memmove(void *, const void *, size_t);
11901 void *memset(void *, int, size_t);
11902 CX char *strcpy(char *restrict, const char *restrict);
11903 char *strncpy(char *restrict, const char *restrict, size_t);
11904 char *strcat(char *restrict, const char *restrict);
11905 char *strchr(const char *, int);
11906 int strcmp(const char *, const char *);
11907 int strcoll(const char *, const char *);
11908 CX int strcoll_l(const char *, const char *, locale_t);
11909 char *strcpy(char *restrict, const char *restrict);
11910 size_t strcspn(const char *, const char *);
11911 CX char *strdup(const char *);
11912 char *strerror(int);
11913 CX char *strerror_l(int, locale_t);
11914 int strerror_r(int, char *, size_t);
11915 size_t strlen(const char *);
11916 char *strncat(char *restrict, const char *restrict, size_t);
11917 int strncmp(const char *, const char *, size_t);
11918 char *strncpy(char *restrict, const char *restrict, size_t);
11919 CX char *strndup(const char *, size_t);
11920 size_t strnlen(const char *, size_t);
11921 char *strpbrk(const char *, const char *);
11922 char *strrchr(const char *, int);
11923 CX char *strsignal(int);
11924 size_t strspn(const char *, const char *);
11925 char *strstr(const char *, const char *);
11926 char *strtok(char *restrict, const char *restrict);
11927 CX char *strtok_r(char *restrict, const char *restrict, char **restrict);
11928 size_t strxfrm(char *restrict, const char *restrict, size_t);
11929 CX size_t strxfrm_l(char *restrict, const char *restrict,
11930 size_t, locale_t);

```

11931 CX Inclusion of the <string.h> header may also make visible all symbols from <stddef.h>.

11932	APPLICATION USAGE	
11933	None.	
11934	RATIONALE	
11935	None.	
11936	FUTURE DIRECTIONS	
11937	None.	
11938	SEE ALSO	
11939	<stddef.h> , <sys/types.h>	-
11940	XSH Section 2.2 (on page 448), memccpy() , memchr() , memcmp() , memcpy() , memmove() ,	
11941	memset() , strcat() , strchr() , strcmp() , strcoll() , strcpy() , strcspn() , strdup() , strerror() , strlen() ,	
11942	strncat() , strncmp() , strncpy() , strpbrk() , strrchr() , strspn() , strstr() , strtok() , strxfrm()	
11943	CHANGE HISTORY	
11944	First released in Issue 1. Derived from Issue 1 of the SVID.	
11945	Issue 5	
11946	The DESCRIPTION is updated for alignment with the POSIX Threads Extension.	
11947	Issue 6	
11948	The strtok_r() function is marked as part of the Thread-Safe Functions option.	
11949	This reference page is updated to align with the ISO/IEC 9899:1999 standard.	
11950	The strerror_r() function is added in response to IEEE PASC Interpretation 1003.1c #39.	
11951	Issue 7	
11952	SD5-XBD-ERN-15 is applied, correcting the prototype for the strerror_r() function.	
11953	The stpcpy() , stpncpy() , strndup() , strlen() , and strsignal() functions are added from The Open	
11954	Group Technical Standard, 2006, Extended API Set Part 1.	
11955	The strcoll_l() , strerror_l() , and strxfrm_l() functions are added from The Open Group Technical	
11956	Standard, 2006, Extended API Set Part 4.	
11957	This reference page is clarified with respect to macros and symbolic constants.	+

11958 **NAME**
 11959 strings.h — string operations

11960 **SYNOPSIS**
 11961 #include <strings.h>

11962 **DESCRIPTION**
 11963 The following shall be declared as functions and may also be defined as macros. Function
 11964 prototypes shall be provided for use with ISO C standard compilers.

11965 XSI `int ffs(int);`
 11966 `int strcasecmp(const char *, const char *);`
 11967 `int strcasecmp_l(const char *, const char *, locale_t);`
 11968 `int strncasecmp(const char *, const char *, size_t);`
 11969 `int strncasecmp_l(const char *, const char *, size_t, locale_t);`

11970 The `size_t` type shall be defined as described in <sys/types.h>.

11971 **APPLICATION USAGE**
 11972 None.

11973 **RATIONALE**
 11974 None.

11975 **FUTURE DIRECTIONS**
 11976 None.

11977 **SEE ALSO**
 11978 <sys/types.h>

11979 XSH `ffs()`, `strcasecmp()`

11980 **CHANGE HISTORY**
 11981 First released in Issue 4, Version 2.

11982 **Issue 6**
 11983 The Open Group Corrigendum U021/2 is applied, correcting the prototype for `index()` to be
 11984 consistent with the reference page.

11985 The `bcmp()`, `bcopy()`, `bzero()`, `index()`, and `rindex()` functions are marked LEGACY.

11986 **Issue 7**
 11987 SD5-XBD-ERN-56 is applied, adding a reference to <sys/types.h> for the `size_t` type.

11988 The LEGACY functions are removed.

11989 The <strings.h> header is moved from the XSI option to the Base.

11990 The `strcasecmp_l()` and `strncasecmp_l()` functions are added from The Open Group Technical
 11991 Standard, 2006, Extended API Set Part 4.

11992 **NAME**11993 stropts.h — STREAMS interface (**STREAMS**)11994 **SYNOPSIS**

11995 OB XSR #include <stropts.h>

11996 **DESCRIPTION**11997 The <stropts.h> header shall define the **bandinfo** structure that includes at least the following
11998 members:11999 int bi_flag Flushing type.
12000 unsigned char bi_pri Priority band.12001 The <stropts.h> header shall define the **strpeek** structure that includes at least the following
12002 members:12003 struct strbuf ctlbuf The control portion of the message.
12004 struct strbuf databuf The data portion of the message.
12005 t_uscalar_t flags RS_HIPRI or 0.12006 The <stropts.h> header shall define the **strbuf** structure that includes at least the following
12007 members:12008 char *buf Pointer to buffer.
12009 int len Length of data.
12010 int maxlen Maximum buffer length.12011 The <stropts.h> header shall define the **strfdinsert** structure that includes at least the following
12012 members:12013 struct strbuf ctlbuf The control portion of the message.
12014 struct strbuf databuf The data portion of the message.
12015 int fildes File descriptor of the other STREAM.
12016 t_uscalar_t flags RS_HIPRI or 0.
12017 int offset Relative location of the stored value.12018 The <stropts.h> header shall define the **striocfl** structure that includes at least the following
12019 members:12020 int ic_cmd *ioctl()* command.
12021 char *ic_dp Pointer to buffer.
12022 int ic_len Length of data.
12023 int ic_timeout Timeout for response.12024 The <stropts.h> header shall define the **strrecvfd** structure that includes at least the following
12025 members:12026 int fda Received file descriptor.
12027 gid_t gid GID of sender.
12028 uid_t uid UID of sender.12029 The **uid_t** and **gid_t** types shall be defined through **typedef** as described in <sys/types.h>.12030 The <stropts.h> header shall define the **t_scalar_t** and **t_uscalar_t** types, respectively, as signed
12031 and unsigned opaque types of equal length of at least 32 bits.12032 The <stropts.h> header shall define the **str_list** structure that includes at least the following
12033 members:

12034 struct str_mlist *sl_modlist STREAMS module names.
 12035 int sl_nmods Number of STREAMS module names.

12036 The <stropts.h> header shall define the **str_mlist** structure that includes at least the following
 12037 member:

12038 char l_name[FMNAMESZ+1] A STREAMS module name.

12039 The <stropts.h> header shall define at least the following symbolic constants for use as the
 12040 *request* argument to *ioctl()*:

12041 L_ATMARK Is the top message “marked”?

12042 L_CANPUT Is a band writable?

12043 L_CKBAND See if any messages exist in a band.

12044 L_FDINSERT Send implementation-defined information about another STREAM.

12045 L_FIND Look for a STREAMS module.

12046 L_FLUSH Flush a STREAM.

12047 L_FLUSHBAND Flush one band of a STREAM.

12048 L_GETBAND Get the band of the top message on a STREAM.

12049 L_GETCLTIME Get close time delay.

12050 L_GETSIG Retrieve current notification signals.

12051 L_GRDOPT Get the read mode.

12052 L_GWROPT Get the write mode.

12053 L_LINK Connect two STREAMS.

12054 L_LIST Get all the module names on a STREAM.

12055 L_LOOK Get the top module name.

12056 L_NREAD Size the top message.

12057 L_PEEK Peek at the top message on a STREAM.

12058 L_PLINK Persistently connect two STREAMS.

12059 L_POP Pop a STREAMS module.

12060 L_PUNLINK Dismantle a persistent STREAMS link.

12061 L_PUSH Push a STREAMS module.

12062 L_RECVFD Get a file descriptor sent via L_SENDFD.

12063 L_SENDFD Pass a file descriptor through a STREAMS pipe.

12064 L_SETCLTIME Set close time delay.

12065 L_SETSIG Ask for notification signals.

12066 L_SRDOPT Set the read mode.

12067 L_STR Send a STREAMS *ioctl()*.

12068 L_SWROPT Set the write mode.

12069 L_UNLINK Disconnect two STREAMS.

12070 The <stropts.h> header shall define at least the following symbolic constant for use with
 12071 L_LOOK:

12072	FMNAMESZ	The minimum size in bytes of the buffer referred to by the <i>arg</i> argument.
12073	The <stropts.h> header shall define at least the following symbolic constants for use with	
12074	L_FLUSH:	
12075	FLUSHR	Flush read queues.
12076	FLUSHRW	Flush read and write queues.
12077	FLUSHW	Flush write queues.
12078	The <stropts.h> header shall define at least the following symbolic constants for use with	
12079	L_SETSIG:	
12080	S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of
12081		SIGPOLL when a priority message reaches the front of the STREAM head read
12082		queue.
12083	S_ERROR	Notification of an error condition reaches the STREAM head.
12084	S_HANGUP	Notification of a hangup reaches the STREAM head.
12085	S_HIPRI	A high-priority message is present on a STREAM head read queue.
12086	S_INPUT	A message, other than a high-priority message, has arrived at the head of a
12087		STREAM head read queue.
12088	S_MSG	A STREAMS signal message that contains the SIGPOLL signal reaches the
12089		front of the STREAM head read queue.
12090	S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM
12091		head is no longer full. This notifies the process that there is room on the queue
12092		for sending (or writing) normal data downstream.
12093	S_RDBAND	A message with a non-zero priority band has arrived at the head of a
12094		STREAM head read queue.
12095	S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a
12096		STREAM head read queue.
12097	S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is
12098		no longer full.
12099	S_WRNORM	Equivalent to S_OUTPUT.
12100	The <stropts.h> header shall define at least the following symbolic constant for use with	
12101	L_PEEK:	
12102	RS_HIPRI	Only look for high-priority messages.
12103	The <stropts.h> header shall define at least the following symbolic constants for use with	
12104	L_SRDOPT:	
12105	RMSGD	Message-discard mode.
12106	RMSGN	Message-non-discard mode.
12107	RNORM	Byte-STREAM mode, the default.
12108	RPROTDAT	Deliver the control part of a message as data when a process issues a <i>read()</i> .
12109	RPROTDIS	Discard the control part of a message, delivering any data part, when a
12110		process issues a <i>read()</i> .
12111	RPROTNORM	Fail <i>read()</i> with [EBADMSG] if a message containing a control part is at the
12112		front of the STREAM head read queue.

<stropts.h>

Headers

12113 The **<stropts.h>** header shall define at least the following symbolic constant for use with |
 12114 `L_SWOPT`:

12115 `SNDZERO` Send a zero-length message downstream when a `write()` of 0 bytes occurs.

12116 The **<stropts.h>** header shall define at least the following symbolic constants for use with |
 12117 `L_ATMARK`:

12118 `ANYMARK` Check if the message is marked.

12119 `LASTMARK` Check if the message is the last one marked on the queue.

12120 The **<stropts.h>** header shall define at least the following symbolic constant for use with |
 12121 `L_UNLINK`:

12122 `MUXID_ALL` Unlink all STREAMs linked to the STREAM associated with *fildev*.

12123 The **<stropts.h>** header shall define the following symbolic constants for `getmsg()`, `getpmsg()`, |
 12124 `putmsg()`, and `putpmsg()`:

12125 `MORECTL` More control information is left in message.

12126 `MOREDATA` More data is left in message.

12127 `MSG_ANY` Receive any message.

12128 `MSG_BAND` Receive message from specified band.

12129 `MSG_HIPRI` Send/receive high-priority message.

12130 The **<stropts.h>** header may make visible all of the symbols from **<unistd.h>**.

12131 The following shall be declared as functions and may also be defined as macros. Function
 12132 prototypes shall be provided.

```

12133 int  fattach(int, const char *);
12134 int  fdetach(const char *);
12135 int  getmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12136         int *restrict);
12137 int  getpmsg(int, struct strbuf *restrict, struct strbuf *restrict,
12138            int *restrict, int *restrict);
12139 int  isastream(int);
12140 int  ioctl(int, int, ...);
12141 int  putmsg(int, const struct strbuf *, const struct strbuf *, int);
12142 int  putpmsg(int, const struct strbuf *, const struct strbuf *, int,
12143            int);
  
```

12144 **APPLICATION USAGE**

12145 None.

12146 **RATIONALE**

12147 None.

12148 **FUTURE DIRECTIONS**

12149 None.

12150 **SEE ALSO**

12151 [<sys/types.h>](#), [<unistd.h>](#)

12152 XSH [close\(\)](#), [fcntl\(\)](#), [getmsg\(\)](#), [ioctl\(\)](#), [open\(\)](#), [pipe\(\)](#), [read\(\)](#), [poll\(\)](#), [putmsg\(\)](#), [signal\(\)](#), [write\(\)](#) |

12153
12154
12155
12156
12157
12158
12159
12160
12161
12162
12163

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

The *flags* members of the **strpeek** and **strfdinsert** structures are changed from **type long** to **t_uscalar_t**.

Issue 6

This header is marked as part of the XSI STREAMS Option Group.

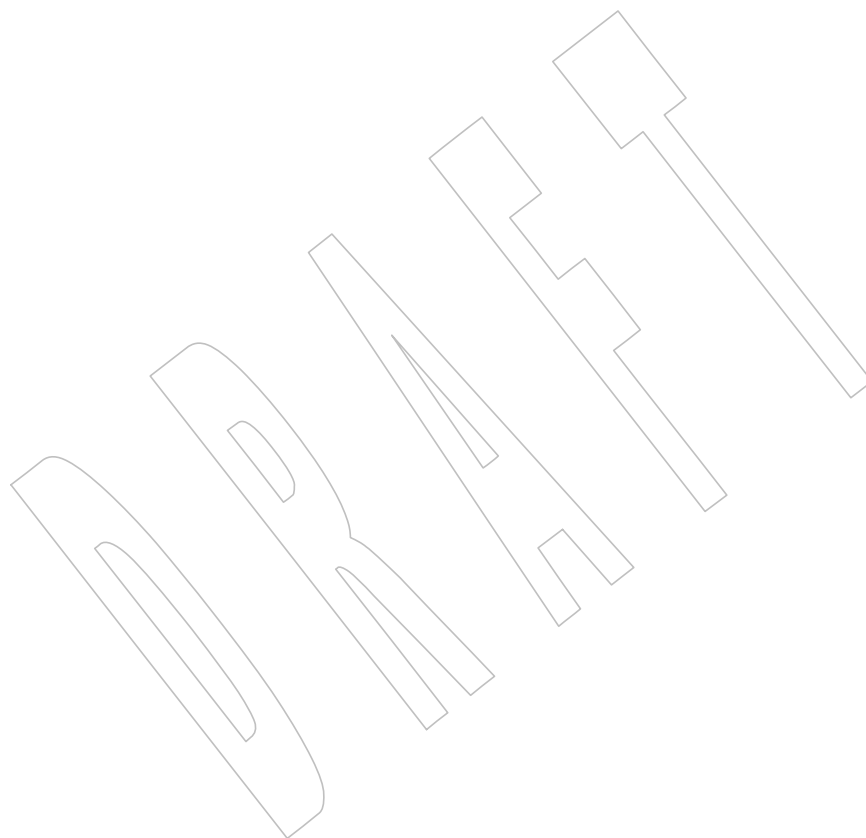
The **restrict** keyword is added to the prototypes for *getmsg()* and *getpmsg()*.

Issue 7

The <**stropts.h**> header is marked obsolescent.

This reference page is clarified with respect to macros and symbolic constants.

+



12164 **NAME**

12165 sys/ipc.h — XSI interprocess communication access structure

12166 **SYNOPSIS**12167 XSI `#include <sys/ipc.h>`12168 **DESCRIPTION**12169 The <sys/ipc.h> header is used by three mechanisms for XSI interprocess communication (IPC):
12170 messages, semaphores, and shared memory. All use a common structure type, **ipc_perm**, to pass
12171 information used in determining permission to perform an IPC operation.12172 The **ipc_perm** structure shall contain the following members:

12173	<code>uid_t</code>	<code>uid</code>	Owner's user ID.
12174	<code>gid_t</code>	<code>gid</code>	Owner's group ID.
12175	<code>uid_t</code>	<code>cuid</code>	Creator's user ID.
12176	<code>gid_t</code>	<code>cgid</code>	Creator's group ID.
12177	<code>mode_t</code>	<code>mode</code>	Read/write permission.

12178 The **uid_t**, **gid_t**, **mode_t**, and **key_t** types shall be defined as described in <sys/types.h>.

12179 The <sys/ipc.h> header shall define the following symbolic constants. |

12180 Mode bits:

12181 `IPC_CREAT` Create entry if key does not exist.12182 `IPC_EXCL` Fail if key exists.12183 `IPC_NOWAIT` Error if request must wait.

12184 Keys:

12185 `IPC_PRIVATE` Private key.

12186 Control commands:

12187 `IPC_RMID` Remove identifier.12188 `IPC_SET` Set options.12189 `IPC_STAT` Get options.12190 The following shall be declared as a function and may also be defined as a macro. A function
12191 prototype shall be provided.12192 `key_t ftok(const char *, int);`12193 **APPLICATION USAGE**

12194 None.

12195 **RATIONALE**

12196 None.

12197 **FUTURE DIRECTIONS**

12198 None.

12199 **SEE ALSO**12200 [<sys/types.h>](#)12201 XSH [ftok\(\)](#) |

12202

CHANGE HISTORY

12203

First released in Issue 2. Derived from System V Release 2.0.

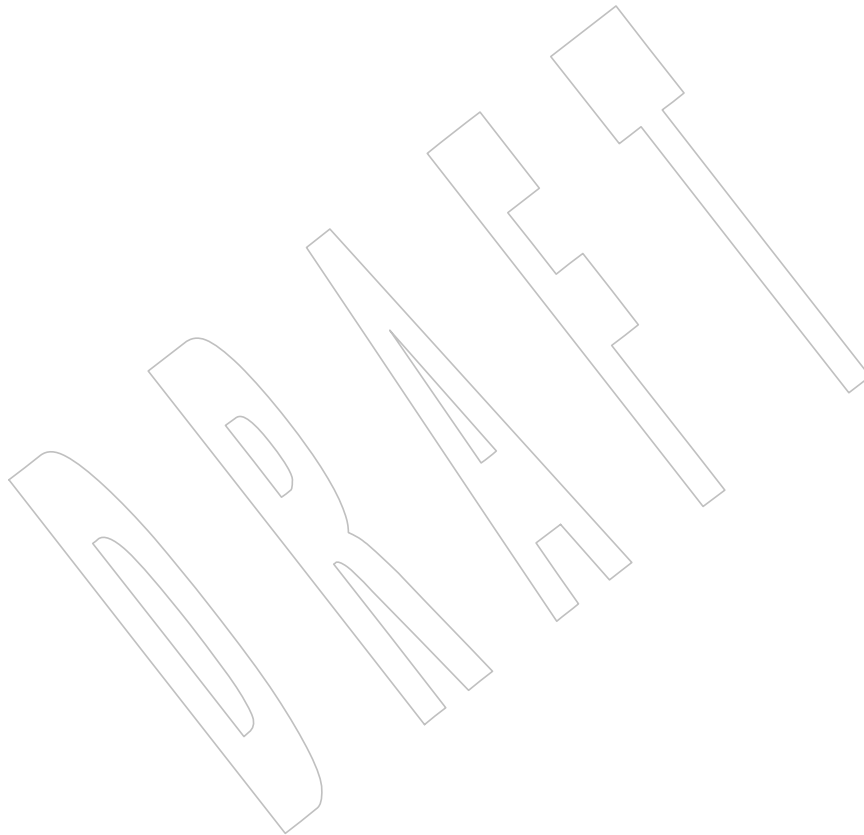
12204

Issue 7

12205

This reference page is clarified with respect to macros and symbolic constants.

+



12206 **NAME**
 12207 sys/mman.h — memory management declarations

12208 **SYNOPSIS**
 12209 #include <sys/mman.h>

12210 **DESCRIPTION**
 12211 The <sys/mman.h> header shall define the following symbolic constants for use as protection
 12212 options:

12213 PROT_EXEC Page can be executed.
 12214 PROT_NONE Page cannot be accessed.
 12215 PROT_READ Page can be read.
 12216 PROT_WRITE Page can be written.

12217 The <sys/mman.h> header shall define the following symbolic constants for use as flag options:

12218 MAP_FIXED Interpret *addr* exactly.
 12219 MAP_PRIVATE Changes are private.
 12220 MAP_SHARED Share changes.

12221 XSI/SIO The <sys/mman.h> header shall define the following symbolic constants for the *msync()*
 12222 function:

12223 MS_ASYNC Perform asynchronous writes.
 12224 MS_INVALIDATE Invalidate mappings.
 12225 MS_SYNC Perform synchronous writes.

12226 ML The <sys/mman.h> header shall define the following symbolic constants for the *mlockall()*
 12227 function:

12228 MCL_CURRENT Lock currently mapped pages.
 12229 MCL_FUTURE Lock pages that become mapped.

12230 The <sys/mman.h> header shall define the symbolic constant MAP_FAILED which shall have
 12231 type **void *** and shall be used to indicate a failure from the *mmap()* function .

12232 ADV If the Advisory Information option is supported, the <sys/mman.h> header shall define
 12233 symbolic constants for the *advice* argument to the *posix_madvise()* function as follows:

12234 POSIX_MADV_DONTNEED
 12235 The application expects that it will not access the specified range in the near future.
 12236 POSIX_MADV_NORMAL
 12237 The application has no advice to give on its behavior with respect to the specified range. It
 12238 is the default characteristic if no advice is given for a range of memory.
 12239 POSIX_MADV_RANDOM
 12240 The application expects to access the specified range in a random order.
 12241 POSIX_MADV_SEQUENTIAL
 12242 The application expects to access the specified range sequentially from lower addresses to
 12243 higher addresses.

12244		POSIX_MADV_WILLNEED
12245		The application expects to access the specified range in the near future.
12246	TYM	The <sys/mman.h> header shall define the following symbolic constants for use as flags for the <i>posix_typed_mem_open()</i> function:
12247		
12248		POSIX_TYPED_MEM_ALLOCATE
12249		Allocate on <i>mmap()</i> .
12250		POSIX_TYPED_MEM_ALLOCATE_CONTIG
12251		Allocate contiguously on <i>mmap()</i> .
12252		POSIX_TYPED_MEM_MAP_ALLOCATABLE
12253		Map on <i>mmap()</i> , without affecting allocatability.
12254		The mode_t , off_t , and size_t types shall be defined as described in <sys/types.h>.
12255	TYM	The <sys/mman.h> header shall define the structure posix_typed_mem_info , which includes at least the following member:
12256		
12257		<code>size_t posix_tmi_length</code> Maximum length which may be allocated
12258		from a typed memory object.
12259		The following shall be declared as functions and may also be defined as macros. Function
12260		prototypes shall be provided.
12261	MLR	<code>int mlock(const void *, size_t);</code>
12262	ML	<code>int mlockall(int);</code>
12263		<code>void *mmap(void *, size_t, int, int, int, off_t);</code>
12264		<code>int mprotect(void *, size_t, int);</code>
12265	XSI SIO	<code>int msync(void *, size_t, int);</code>
12266	MLR	<code>int munlock(const void *, size_t);</code>
12267	ML	<code>int munlockall(void);</code>
12268		<code>int munmap(void *, size_t);</code>
12269	ADV	<code>int posix_madvise(void *, size_t, int);</code>
12270	TYM	<code>int posix_mem_offset(const void *restrict, size_t, off_t *restrict,</code>
12271		<code>size_t *restrict, int *restrict);</code>
12272		<code>int posix_typed_mem_get_info(int, struct posix_typed_mem_info *);</code>
12273		<code>int posix_typed_mem_open(const char *, int, int);</code>
12274	SHM	<code>int shm_open(const char *, int, mode_t);</code>
12275		<code>int shm_unlink(const char *);</code>
12276		APPLICATION USAGE
12277		None.
12278		RATIONALE
12279		None.
12280		FUTURE DIRECTIONS
12281		None.
12282		SEE ALSO
12283		<sys/types.h>
12284		XSH <i>mlock()</i> , <i>mlockall()</i> , <i>mmap()</i> , <i>mprotect()</i> , <i>msync()</i> , <i>munmap()</i> , <i>posix_mem_offset()</i> ,
12285		<i>posix_typed_mem_get_info()</i> , <i>posix_typed_mem_open()</i> , <i>shm_open()</i> , <i>shm_unlink()</i>

CHANGE HISTORY

12286 First released in Issue 4, Version 2.
12287

Issue 5

12288 Updated for alignment with the POSIX Realtime Extension.
12289

Issue 6

12290 The <sys/mman.h> header is marked as dependent on support for either the Memory Mapped
12291 Files, Process Memory Locking, or Shared Memory Objects options.
12292

12293 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 12294 • The TYM margin code is added to the list of margin codes for the <sys/mman.h> header
12295 line, as well as for other lines.
- 12296 • The POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG,
12297 and POSIX_TYPED_MEM_MAP_ALLOCATABLE flags are added.
- 12298 • The `posix_tmi_length` structure is added.
- 12299 • The `posix_mem_offset()`, `posix_typed_mem_get_info()`, and `posix_typed_mem_open()` functions
12300 are added.

12301 The `restrict` keyword is added to the prototype for `posix_mem_offset()`.

12302 IEEE PASC Interpretation 1003.1 #102 is applied, adding the prototype for `posix_madvise()`.

12303 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/16 is applied, correcting margin code and
12304 shading errors for the `mlock()` and `munlock()` functions.

12305 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code
12306 for the `mmap()` function from MF|SHM to MC3 (notation for MF|SHM|TYM).

12307 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code
12308 for the `munmap()` function from MF|SHM to MC3 (notation for MF|SHM|TYM).

Issue 7

12309 SD5-XBD-ERN-5 is applied, rewriting the DESCRIPTION.
12310

12311 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
12312 the Base.

12313 This reference page is clarified with respect to macros and symbolic constants. +

12314 **NAME**
 12315 `sys/msg.h` — XSI message queue structures

12316 **SYNOPSIS**
 12317 XSI `#include <sys/msg.h>`

12318 DESCRIPTION

12319 The `<sys/msg.h>` header shall define the following data types through **typedef**:

12320 **msgqnum_t** Used for the number of messages in the message queue.

12321 **msglen_t** Used for the number of bytes allowed in a message queue.

12322 These types shall be unsigned integer types that are able to store values at least as large as a type
 12323 **unsigned short**.

12324 The `<sys/msg.h>` header shall define the following symbolic constant as a message operation
 12325 flag:

12326 **MSG_NOERROR** No error if big message.

12327 The **msqid_ds** structure shall contain the following members:

12328	<code>struct ipc_perm</code>	<code>msg_perm</code>	Operation permission structure.
12329	<code>msgqnum_t</code>	<code>msg_qnum</code>	Number of messages currently on queue.
12330	<code>msglen_t</code>	<code>msg_qbytes</code>	Maximum number of bytes allowed on queue.
12331	<code>pid_t</code>	<code>msg_lspid</code>	Process ID of last <code>msgsnd()</code> .
12332	<code>pid_t</code>	<code>msg_lrpid</code>	Process ID of last <code>msgrcv()</code> .
12333	<code>time_t</code>	<code>msg_stime</code>	Time of last <code>msgsnd()</code> .
12334	<code>time_t</code>	<code>msg_rtime</code>	Time of last <code>msgrcv()</code> .
12335	<code>time_t</code>	<code>msg_ctime</code>	Time of last change.

12336 The **pid_t**, **time_t**, **key_t**, **size_t**, and **ssize_t** types shall be defined as described in
 12337 `<sys/types.h>`.

12338 The following shall be declared as functions and may also be defined as macros. Function
 12339 prototypes shall be provided.

```
12340 int      msgctl(int, int, struct msqid_ds *);
12341 int      msgget(key_t, int);
12342 ssize_t  msgrcv(int, void *, size_t, long, int);
12343 int      msgsnd(int, const void *, size_t, int);
```

12344 In addition, all of the symbols from `<sys/ipc.h>` shall be defined when this header is included.

12345 APPLICATION USAGE

12346 None.

12347 RATIONALE

12348 None.

12349 FUTURE DIRECTIONS

12350 None.

12351 SEE ALSO

12352 `<sys/ipc.h>`, `<sys/types.h>`

12353 XSH `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()`

12354

CHANGE HISTORY

12355

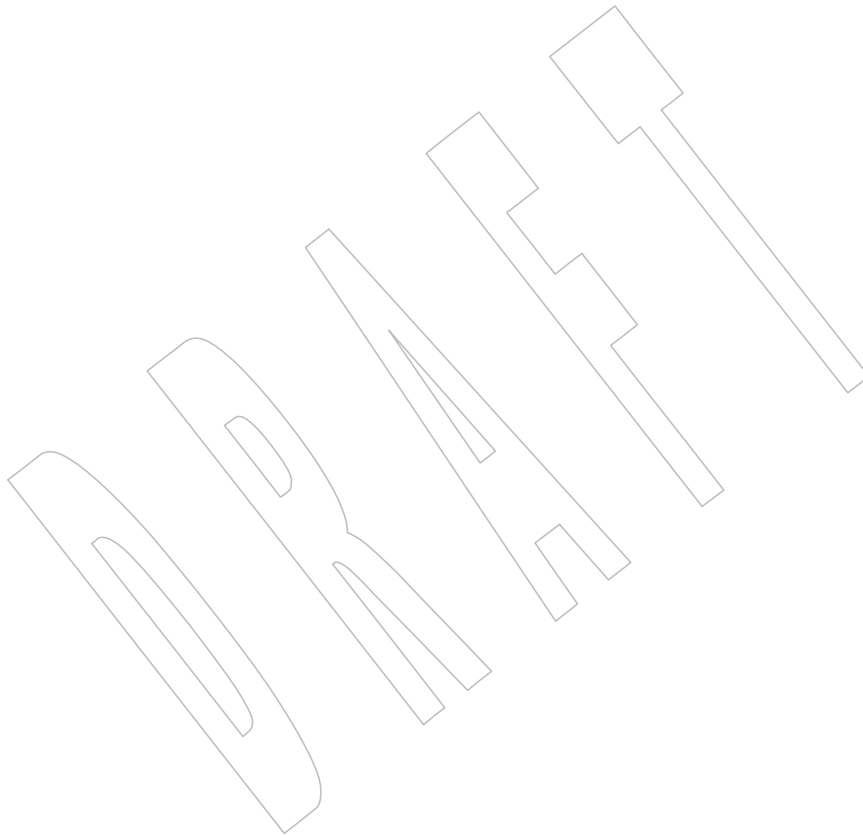
First released in Issue 2. Derived from System V Release 2.0.

12356

Issue 7

12357

This reference page is clarified with respect to macros and symbolic constants. +



12358 **NAME**
 12359 `sys/resource.h` — definitions for XSI resource operations

12360 **SYNOPSIS**
 12361 XSI `#include <sys/resource.h>`

12362 **DESCRIPTION**
 12363 The <sys/resource.h> header shall define the following symbolic constants as possible values of
 12364 the *which* argument of `getpriority()` and `setpriority()`:

12365 `PRIO_PROCESS` Identifies the *who* argument as a process ID.
 12366 `PRIO_PGRP` Identifies the *who* argument as a process group ID.
 12367 `PRIO_USER` Identifies the *who* argument as a user ID.

12368 The following type shall be defined through **typedef**:

12369 `rlim_t` Unsigned integer type used for limit values.

12370 The <sys/resource.h> header shall define the following symbolic constants, which shall have
 12371 values suitable for use in `#if` preprocessing directives:

12372 `RLIM_INFINITY` A value of `rlim_t` indicating no limit.
 12373 `RLIM_SAVED_MAX` A value of type `rlim_t` indicating an unrepresentable saved hard
 12374 limit.
 12375 `RLIM_SAVED_CUR` A value of type `rlim_t` indicating an unrepresentable saved soft limit.

12376 On implementations where all resource limits are representable in an object of type `rlim_t`,
 12377 `RLIM_SAVED_MAX` and `RLIM_SAVED_CUR` need not be distinct from `RLIM_INFINITY`.

12378 The following symbolic constants shall be defined as possible values of the *who* parameter of
 12379 `getrusage()`:

12380 `RUSAGE_SELF` Returns information about the current process.
 12381 `RUSAGE_CHILDREN` Returns information about children of the current process.

12382 The <sys/resource.h> header shall define the `rlimit` structure that includes at least the following
 12383 members:

12384 `rlim_t rlim_cur` The current (soft) limit.
 12385 `rlim_t rlim_max` The hard limit.

12386 The <sys/resource.h> header shall define the `rusage` structure that includes at least the
 12387 following members:

12388 `struct timeval ru_utime` User time used.
 12389 `struct timeval ru_stime` System time used.

12390 The `timeval` structure shall be defined as described in <sys/time.h>.

12391 The following symbolic constants shall be defined as possible values for the *resource* argument of
 12392 `getrlimit()` and `setrlimit()`:

12393 `RLIMIT_CORE` Limit on size of **core** file.
 12394 `RLIMIT_CPU` Limit on CPU time per process.

12395	RLIMIT_DATA	Limit on data segment size.
12396	RLIMIT_FSIZE	Limit on file size.
12397	RLIMIT_NOFILE	Limit on number of open files.
12398	RLIMIT_STACK	Limit on stack size.
12399	RLIMIT_AS	Limit on address space size.

12400 The following shall be declared as functions and may also be defined as macros. Function
12401 prototypes shall be provided.

```
12402 int getpriority(int, id_t);
12403 int getrlimit(int, struct rlimit *);
12404 int getrusage(int, struct rusage *);
12405 int setpriority(int, id_t, int);
12406 int setrlimit(int, const struct rlimit *);
```

12407 The `id_t` type shall be defined through **typedef** as described in <sys/types.h>.

12408 Inclusion of the <sys/resource.h> header may also make visible all symbols from <sys/time.h>.

12409 APPLICATION USAGE

12410 None.

12411 RATIONALE

12412 None.

12413 FUTURE DIRECTIONS

12414 None.

12415 SEE ALSO

12416 [<sys/time.h>](#), [<sys/types.h>](#)

12417 XSH [getpriority\(\)](#), [getrlimit\(\)](#), [getrusage\(\)](#)

12418 CHANGE HISTORY

12419 First released in Issue 4, Version 2.

12420 Issue 5

12421 Large File System extensions are added.

12422 Issue 7

12423 This reference page is clarified with respect to macros and symbolic constants. +

12424 **NAME**

12425 sys/select.h — select types

12426 **SYNOPSIS**

12427 #include <sys/select.h>

12428 **DESCRIPTION**12429 The <sys/select.h> header shall define the **timeval** structure that includes at least the following
12430 members:

12431	time_t	tv_sec	Seconds.
12432	suseconds_t	tv_usec	Microseconds.

12433 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.12434 The **sigset_t** type shall be defined as described in <signal.h>.12435 The **timespec** structure shall be defined as described in <time.h>.12436 The <sys/select.h> header shall define the **fd_set** type as a structure.12437 The <sys/select.h> header shall define the following symbolic constant, which shall have a value
12438 suitable for use in #if preprocessing directives:12439 **FD_SETSIZE** Maximum number of file descriptors in an **fd_set** structure.12440 The following shall be declared as functions, defined as macros, or both. If functions are
12441 declared, function prototypes shall be provided.

```

12442 void FD_CLR(int, fd_set *);
12443 int  FD_ISSET(int, fd_set *);
12444 void FD_SET(int, fd_set *);
12445 void FD_ZERO(fd_set *);

```

12446 If implemented as macros, these may evaluate their arguments more than once, so applications
12447 should ensure that the arguments they supply are never expressions with side effects.12448 The following shall be declared as functions and may also be defined as macros. Function
12449 prototypes shall be provided.

```

12450 int  pselect(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12451             const struct timespec *restrict, const sigset_t *restrict);
12452 int  select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
12453            struct timeval *restrict);

```

12454 Inclusion of the <sys/select.h> header may make visible all symbols from the headers
12455 <signal.h>, <sys/time.h>, and <time.h>.12456 **APPLICATION USAGE**

12457 None.

12458 **RATIONALE**

12459 None.

12460 **FUTURE DIRECTIONS**

12461 None.

12462

SEE ALSO

12463

[<signal.h>](#), [<sys/time.h>](#), [<sys/types.h>](#), [<time.h>](#)

12464

XSH *pselect()* |

12465

CHANGE HISTORY

12466

First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

12467

The requirement for the **fd_set** structure to have a member *fds_bits* has been removed as per The Open Group Base Resolution bwg2001-005.

12468

12469

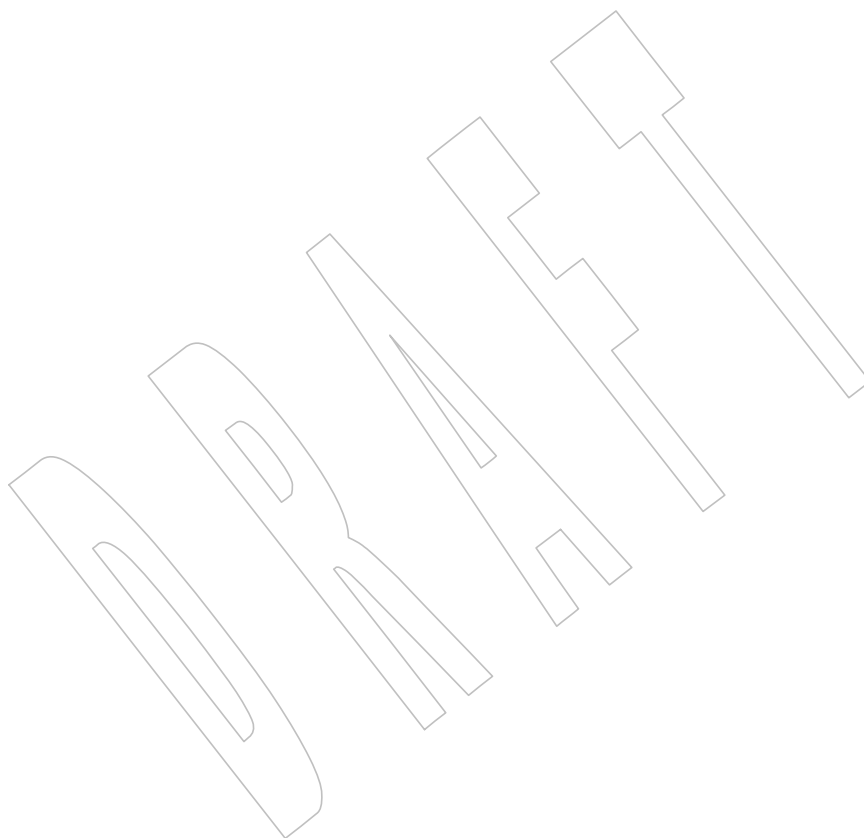
Issue 7

12470

SD5-XBD-ERN-6 is applied, reordering the DESCRIPTION.

12471

This reference page is clarified with respect to macros and symbolic constants. +



12472 **NAME**
 12473 sys/sem.h — XSI semaphore facility

12474 **SYNOPSIS**
 12475 XSI `#include <sys/sem.h>`

12476 **DESCRIPTION**

12477 The <sys/sem.h> header shall define the following symbolic constant for use as a semaphore
 12478 operation flag:

12479 SEM_UNDO Set up adjust on exit entry.

12480 The <sys/sem.h> header shall define the following symbolic constants for use as commands for
 12481 the *semctl()* function:

12482 GETNCNT Get *semncnt*.

12483 GETPID Get *sempid*.

12484 GETVAL Get *semval*.

12485 GETALL Get all cases of *semval*.

12486 GETZCNT Get *semzcnt*.

12487 SETVAL Set *semval*.

12488 SETALL Set all cases of *semval*.

12489 The <sys/sem.h> header shall define the **semid_ds** structure which shall contain the following
 12490 members:

12491 struct ipc_perm sem_perm Operation permission structure.

12492 unsigned short sem_nsems Number of semaphores in set.

12493 time_t sem_otime Last *semop()* time.

12494 time_t sem_ctime Last time changed by *semctl()*.

12495 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12496 A semaphore shall be represented by an anonymous structure containing the following
 12497 members:

12498 unsigned short semval Semaphore value.

12499 pid_t sempid Process ID of last operation.

12500 unsigned short semncnt Number of processes waiting for *semval*
 12501 to become greater than current value.

12502 unsigned short semzcnt Number of processes waiting for *semval*
 12503 to become 0.

12504 The <sys/sem.h> header shall define the **sembuf** structure which shall contain the following
 12505 members:

12506 unsigned short sem_num Semaphore number.

12507 short sem_op Semaphore operation.

12508 short sem_flg Operation flags.

12509 The following shall be declared as functions and may also be defined as macros. Function
 12510 prototypes shall be provided.

12511 int semctl(int, int, int, ...);

12512 int semget(key_t, int, int);

12513 int semop(int, struct sembuf *, size_t);

12514 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12515 **APPLICATION USAGE**

12516 None.

12517 **RATIONALE**

12518 None.

12519 **FUTURE DIRECTIONS**

12520 None.

12521 **SEE ALSO**

12522 <sys/ipc.h>, <sys/types.h>

12523 XSH *semctl()*, *semget()*, *semop()* |

12524 **CHANGE HISTORY**

12525 First released in Issue 2. Derived from System V Release 2.0.

12526 **Issue 7**

12527 This reference page is clarified with respect to macros and symbolic constants. +

DRAFT

12528 **NAME**
 12529 sys/shm.h — XSI shared memory facility

12530 **SYNOPSIS**
 12531 XSI `#include <sys/shm.h>`

12532 DESCRIPTION

12533 The <sys/shm.h> header shall define the following symbolic constants:

12534 SHM_RDONLY Attach read-only (else read-write).

12535 SHM_RND Round attach address to SHMLBA.

12536 SHMLBA Segment low boundary address multiple.

12537 The following data types shall be defined through **typedef**:

12538 **shmatt_t** Unsigned integer used for the number of current attaches that must be able to
 12539 store values at least as large as a type **unsigned short**.

12540 The **shmid_ds** structure shall contain the following members:

12541	<code>struct ipc_perm</code>	<code>shm_perm</code>	Operation permission structure.
12542	<code>size_t</code>	<code>shm_segsz</code>	Size of segment in bytes.
12543	<code>pid_t</code>	<code>shm_lpid</code>	Process ID of last shared memory operation.
12544	<code>pid_t</code>	<code>shm_cpid</code>	Process ID of creator.
12545	<code>shmatt_t</code>	<code>shm_nattch</code>	Number of current attaches.
12546	<code>time_t</code>	<code>shm_atime</code>	Time of last <i>shmat</i> ().
12547	<code>time_t</code>	<code>shm_dtime</code>	Time of last <i>shmdt</i> ().
12548	<code>time_t</code>	<code>shm_ctime</code>	Time of last change by <i>shmctl</i> ().

12549 The **pid_t**, **time_t**, **key_t**, and **size_t** types shall be defined as described in <sys/types.h>.

12550 The following shall be declared as functions and may also be defined as macros. Function
 12551 prototypes shall be provided.

```
12552 void *shmat(int, const void *, int);
12553 int shmctl(int, int, struct shmid_ds *);
12554 int shmdt(const void *);
12555 int shmget(key_t, size_t, int);
```

12556 In addition, all of the symbols from <sys/ipc.h> shall be defined when this header is included.

12557 APPLICATION USAGE

12558 None.

12559 RATIONALE

12560 None.

12561 FUTURE DIRECTIONS

12562 None.

12563 SEE ALSO

12564 <sys/ipc.h>, <sys/types.h>

12565 XSH *shmat*(), *shmctl*(), *shmdt*(), *shmget*()

12566
12567
12568
12569
12570
12571**CHANGE HISTORY**

First released in Issue 2. Derived from System V Release 2.0.

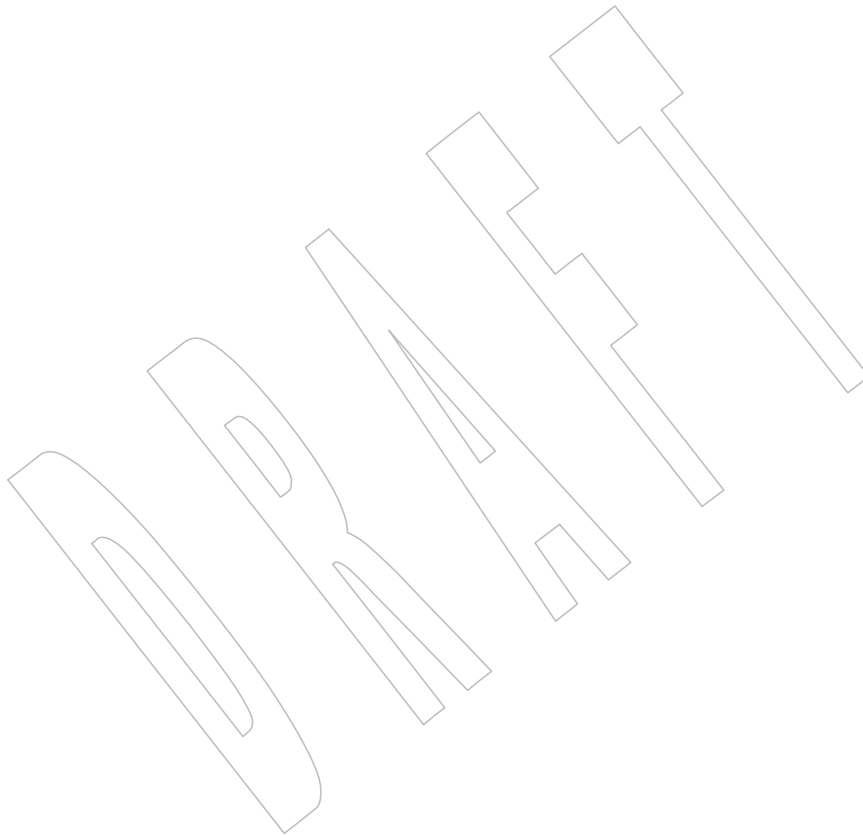
Issue 5

The type of *shm_segsz* is changed from **int** to **size_t**.

Issue 7

This reference page is clarified with respect to macros and symbolic constants.

+



12572 **NAME**

12573 sys/socket.h — main sockets header

12574 **SYNOPSIS**

12575 #include <sys/socket.h>

12576 **DESCRIPTION**12577 The <sys/socket.h> header shall define the type **socklen_t**, which is an integer type of width of
12578 at least 32 bits; see APPLICATION USAGE.12579 The <sys/socket.h> header shall define the unsigned integer type **sa_family_t**.12580 The <sys/socket.h> header shall define the **sockaddr** structure that includes at least the
12581 following members:12582 sa_family_t sa_family Address family.
12583 char sa_data[] Socket address (variable-length data).12584 The **sockaddr** structure is used to define a socket address which is used in the *bind()*, *connect()*,
12585 *getpeername()*, *getsockname()*, *recvfrom()*, and *sendto()* functions.12586 The <sys/socket.h> header shall define the **sockaddr_storage** structure. This structure shall be:

- 12587
- Large enough to accommodate all supported protocol-specific address structures
 - Aligned at an appropriate boundary so that pointers to it can be cast as pointers to
12588 protocol-specific address structures and used to access the fields of those structures
12589 without alignment problems
- 12590

12591 The **sockaddr_storage** structure shall contain at least the following members:

12592 sa_family_t ss_family

12593 When a **sockaddr_storage** structure is cast as a **sockaddr** structure, the *ss_family* field of the
12594 **sockaddr_storage** structure shall map onto the *sa_family* field of the **sockaddr** structure. When a
12595 **sockaddr_storage** structure is cast as a protocol-specific address structure, the *ss_family* field
12596 shall map onto a field of that structure that is of type **sa_family_t** and that identifies the
12597 protocol's address family.12598 The <sys/socket.h> header shall define the **msghdr** structure that includes at least the following
12599 members:12600 void *msg_name Optional address.
12601 socklen_t msg_namelen Size of address.
12602 struct iovec *msg_iov Scatter/gather array.
12603 int msg_iovlen Members in *msg_iov*.
12604 void *msg_control Ancillary data; see below.
12605 socklen_t msg_controllen Ancillary data buffer *len*.
12606 int msg_flags Flags on received message.12607 The **msghdr** structure is used to minimize the number of directly supplied parameters to the
12608 *recvmsg()* and *sendmsg()* functions. This structure is used as a *value-result* parameter in the
12609 *recvmsg()* function and *value* only for the *sendmsg()* function.12610 The **iovec** structure shall be defined as described in <sys/uio.h>.12611 The <sys/socket.h> header shall define the **cmsghdr** structure that includes at least the
12612 following members:12613 socklen_t cmsg_len Data byte count, including the **cmsghdr**.
12614 int cmsg_level Originating protocol.

12615 int msg_type Protocol-specific type.

12616 The **msg_hdr** structure is used for storage of ancillary data object information.

12617 Ancillary data consists of a sequence of pairs, each consisting of a **msg_hdr** structure followed
12618 by a data array. The data array contains the ancillary data message, and the **msg_hdr** structure
12619 contains descriptive information that allows an application to correctly parse the data.

12620 The values for *msg_level* shall be legal values for the *level* argument to the *getsockopt()* and
12621 *setsockopt()* functions. The system documentation shall specify the *msg_type* definitions for the
12622 supported protocols.

12623 Ancillary data is also possible at the socket level. The <sys/socket.h> header shall define the
12624 following symbolic constant for use as the *msg_type* value when *msg_level* is SOL_SOCKET:

12625 SCM_RIGHTS Indicates that the data array contains the access rights to be sent or
12626 received.

12627 The <sys/socket.h> header shall define the following macros to gain access to the data arrays in
12628 the ancillary data associated with a message header:

12629 MSG_DATA(*msg*)

12630 If the argument is a pointer to a **msg_hdr** structure, this macro shall return an unsigned
12631 character pointer to the data array associated with the **msg_hdr** structure.

12632 MSG_NXTHDR(*mhdr, msg*)

12633 If the first argument is a pointer to a **msg_hdr** structure and the second argument is a pointer
12634 to a **msg_hdr** structure in the ancillary data pointed to by the *msg_control* field of that
12635 **msg_hdr** structure, this macro shall return a pointer to the next **msg_hdr** structure, or a null
12636 pointer if this structure is the last **msg_hdr** in the ancillary data.

12637 MSG_FIRSTHDR(*mhdr*)

12638 If the argument is a pointer to a **msg_hdr** structure, this macro shall return a pointer to the
12639 first **msg_hdr** structure in the ancillary data associated with this **msg_hdr** structure, or a null
12640 pointer if there is no ancillary data associated with the **msg_hdr** structure.

12641 The <sys/socket.h> header shall define the **linger** structure that includes at least the following
12642 members:

12643 int l_onoff Indicates whether linger option is enabled.

12644 int l_linger Linger time, in seconds.

12645 The <sys/socket.h> header shall define the following symbolic constants with distinct values:

12646 SOCK_DGRAM Datagram socket.

12647 SOCK_RAW Raw Protocol Interface.

12648 SOCK_SEQPACKET Sequenced-packet socket.

12649 SOCK_STREAM Byte-stream socket.

12650 The <sys/socket.h> header shall define the following symbolic constant for use as the *level*
12651 argument of *setsockopt()* and *getsockopt()*.

12652 SOL_SOCKET Options to be accessed at socket level, not protocol level.

12653 The <sys/socket.h> header shall define the following symbolic constants with distinct values for
12654 use as the *option_name* argument in *getsockopt()* or *setsockopt()* calls:

12655 SO_ACCEPTCONN Socket is accepting connections.

12656 SO_BROADCAST Transmission of broadcast messages is supported.

12657	SO_DEBUG	Debugging information is being recorded.
12658	SO_DONTROUTE	Bypass normal routing.
12659	SO_ERROR	Socket error status.
12660	SO_KEEPALIVE	Connections are kept alive with periodic messages.
12661	SO_LINGER	Socket lingers on close.
12662	SO_OOBINLINE	Out-of-band data is transmitted in line.
12663	SO_RCVBUF	Receive buffer size.
12664	SO_RCVLOWAT	Receive “low water mark”.
12665	SO_RCVTIMEO	Receive timeout.
12666	SO_REUSEADDR	Reuse of local addresses is supported.
12667	SO_SNDBUF	Send buffer size.
12668	SO_SNDLOWAT	Send “low water mark”.
12669	SO_SNDTIMEO	Send timeout.
12670	SO_TYPE	Socket type.
12671	The <sys/socket.h> header shall define the following symbolic constant for use as the maximum	
12672	<i>backlog</i> queue length which may be specified by the <i>backlog</i> field of the <i>listen()</i> function:	
12673	SOMAXCONN	The maximum <i>backlog</i> queue length.
12674	The <sys/socket.h> header shall define the following symbolic constants with distinct values for	
12675	use as the valid values for the <i>msg_flags</i> field in the msghdr structure, or the <i>flags</i> parameter in	
12676	<i>recv()</i> , <i>recvfrom()</i> , <i>recvmsg()</i> , <i>send()</i> , <i>sendmsg()</i> , or <i>sendto()</i> calls:	
12677	MSG_CTRUNC	Control data truncated.
12678	MSG_DONTROUTE	Send without using routing tables.
12679	MSG_EOR	Terminates a record (if supported by the protocol).
12680	MSG_OOB	Out-of-band data.
12681	MSG_NOSIGNAL	No SIGPIPE generated when an attempt to send is made on a stream-
12682		oriented socket that is no longer connected.
12683	MSG_PEEK	Leave received data in queue.
12684	MSG_TRUNC	Normal data truncated.
12685	MSG_WAITALL	Attempt to fill the read buffer.
12686	The <sys/socket.h> header shall define the following symbolic constants with distinct values:	
12687	AF_INET	Internet domain sockets for use with IPv4 addresses.
12688	IP6 AF_INET6	Internet domain sockets for use with IPv6 addresses.
12689	AF_UNIX	UNIX domain sockets.
12690	AF_UNSPEC	Unspecified.
12691	The <sys/socket.h> header shall define the following symbolic constants with distinct values:	
12692	SHUT_RD	Disables further receive operations.

12693 SHUT_RDWR Disables further send and receive operations.

12694 SHUT_WR Disables further send operations.

12695 The `ssize_t` type shall be defined as described in <sys/types.h>.

12696 The following shall be declared as functions and may also be defined as macros. Function
12697 prototypes shall be provided.

```
12698 int accept(int, struct sockaddr *restrict, socklen_t *restrict);
12699 int bind(int, const struct sockaddr *, socklen_t);
12700 int connect(int, const struct sockaddr *, socklen_t);
12701 int getpeername(int, struct sockaddr *restrict, socklen_t *restrict);
12702 int getsockname(int, struct sockaddr *restrict, socklen_t *restrict);
12703 int getsockopt(int, int, int, void *restrict, socklen_t *restrict);
12704 int listen(int, int);
12705 ssize_t recv(int, void *, size_t, int);
12706 ssize_t recvfrom(int, void *restrict, size_t, int,
12707 struct sockaddr *restrict, socklen_t *restrict);
12708 ssize_t recvmsg(int, struct msghdr *, int);
12709 ssize_t send(int, const void *, size_t, int);
12710 ssize_t sendmsg(int, const struct msghdr *, int);
12711 ssize_t sendto(int, const void *, size_t, int, const struct sockaddr *,
12712 socklen_t);
12713 int setsockopt(int, int, int, const void *, socklen_t);
12714 int shutdown(int, int);
12715 int socketatmark(int);
12716 int socket(int, int, int);
12717 int socketpair(int, int, int, int [2]);
```

12718 Inclusion of <sys/socket.h> may also make visible all symbols from <sys/unistd.h>.

12719 APPLICATION USAGE

12720 To forestall portability problems, it is recommended that applications not use values larger than
12721 $2^{31} - 1$ for the `socklen_t` type.

12722 The `sockaddr_storage` structure solves the problem of declaring storage for automatic variables
12723 which is both large enough and aligned enough for storing the socket address data structure of
12724 any family. For example, code with a file descriptor and without the context of the address
12725 family can pass a pointer to a variable of this type, where a pointer to a socket address structure
12726 is expected in calls such as `getpeername()`, and determine the address family by accessing the
12727 received content after the call.

12728 The example below illustrates a data structure which aligns on a 64-bit boundary. An
12729 implementation-defined field `_ss_align` following `_ss_pad1` is used to force a 64-bit alignment
12730 which covers proper alignment good enough for needs of at least `sockaddr_in6` (IPv6) and
12731 `sockaddr_in` (IPv4) address data structures. The size of padding field `_ss_pad1` depends on the
12732 chosen alignment boundary. The size of padding field `_ss_pad2` depends on the value of overall
12733 size chosen for the total size of the structure. This size and alignment are represented in the
12734 above example by implementation-defined (not required) constants `_SS_MAXSIZE` (chosen
12735 value 128) and `_SS_ALIGNMENT` (with chosen value 8). Constants `_SS_PAD1SIZE` (derived
12736 value 6) and `_SS_PAD2SIZE` (derived value 112) are also for illustration and not required. The
12737 implementation-defined definitions and structure field names above start with an underscore to
12738 denote implementation private name space. Portable code is not expected to access or reference
12739 those fields or constants.

```
12740 /*
12741 * Desired design of maximum size and alignment.
12742 */
```



```

12743     #define _SS_MAXSIZE 128
12744         /* Implementation-defined maximum size. */
12745     #define _SS_ALIGNSIZE (sizeof(int64_t))
12746         /* Implementation-defined desired alignment. */
12747
12748     /*
12749     * Definitions used for sockaddr_storage structure paddings design.
12750     */
12751     #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
12752     #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t)+ \
12753         _SS_PAD1SIZE + _SS_ALIGNSIZE))
12754     struct sockaddr_storage {
12755         sa_family_t  ss_family; /* Address family. */
12756     /*
12757     * Following fields are implementation-defined.
12758     */
12759     char  _ss_pad1[_SS_PAD1SIZE];
12760         /* 6-byte pad; this is to make implementation-defined
12761         pad up to alignment field that follows explicit in
12762         the data structure. */
12763     int64_t  _ss_align; /* Field to force desired structure
12764         storage alignment. */
12765     char  _ss_pad2[_SS_PAD2SIZE];
12766         /* 112-byte pad to achieve desired size,
12767         _SS_MAXSIZE value minus size of ss_family
12768         __ss_pad1, __ss_align fields is 112. */
12769     };

```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO[<sys/types.h>](#), [<sys/uio.h>](#)

XSH [accept\(\)](#), [bind\(\)](#), [connect\(\)](#), [getpeername\(\)](#), [getsockname\(\)](#), [getsockopt\(\)](#), [listen\(\)](#), [recv\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#), [send\(\)](#), [sendmsg\(\)](#), [sendto\(\)](#), [setsockopt\(\)](#), [shutdown\(\)](#), [socket\(\)](#), [socketpair\(\)](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the prototypes for [accept\(\)](#), [getpeername\(\)](#), [getsockname\(\)](#), [getsockopt\(\)](#), and [recvfrom\(\)](#).

Issue 7SD5-XBD-ERN-56 is applied, adding a reference to [<sys/types.h>](#) for the **ssize_t** type.

SD5-XBD-ERN-62 is applied.

The [MSG_NOSIGNAL\(\)](#) macro is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

This reference page is clarified with respect to macros and symbolic constants. +

12787 **NAME**12788 sys/stat.h — data returned by the `stat()` function12789 **SYNOPSIS**

12790 #include <sys/stat.h>

12791 **DESCRIPTION**12792 The <sys/stat.h> header shall define the structure of the data returned by the functions `fstat()`,
12793 `lstat()`, and `stat()`.12794 The **stat** structure shall contain at least the following members:

12795		<code>dev_t st_dev</code>	Device ID of device containing file.
12796		<code>ino_t st_ino</code>	File serial number.
12797		<code>mode_t st_mode</code>	Mode of file (see below).
12798		<code>nlink_t st_nlink</code>	Number of hard links to the file.
12799		<code>uid_t st_uid</code>	User ID of file.
12800		<code>gid_t st_gid</code>	Group ID of file.
12801	XSI	<code>dev_t st_rdev</code>	Device ID (if file is character or block special).
12802		<code>off_t st_size</code>	For regular files, the file size in bytes.
12803			For symbolic links, the length in bytes of the
12804			pathname contained in the symbolic link.
12805	SHM		For a shared memory object, the length in bytes.
12806	TYM		For a typed memory object, the length in bytes.
12807			For other file types, the use of this field is
12808			unspecified.
12809		<code>struct timespec st_atim</code>	Last data access timestamp.
12810		<code>struct timespec at_mtim</code>	Last data modification timestamp.
12811		<code>struct timespec st_ctim</code>	Last file status change timestamp.
12812	XSI	<code>blksize_t st_blksize</code>	A file system-specific preferred I/O block size
12813			for this object. In some file system types, this
12814			may vary from file to file.
12815		<code>blkcnt_t st_blocks</code>	Number of blocks allocated for this object.

12816 The `st_ino` and `st_dev` fields taken together uniquely identify the file within the system. The
 12817 **blkcnt_t**, **blksize_t**, **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types shall be
 12818 defined as described in <sys/types.h>. The **timespec** structure shall be defined as described in
 12819 <time.h>. Times shall be given in seconds since the Epoch.

12820 Which structure members have meaningful values depends on the type of file. For further
 12821 information, see the descriptions of `fstat()`, `lstat()`, and `stat()` in the System Interfaces volume of
 12822 POSIX.1-200x.

12823 For compatibility with earlier versions of this standard, the `st_atime` macro shall be defined with
 12824 the value `st_atim.tv_sec`. Similarly, `st_ctime` and `st_mtime` shall be defined as macros with the
 12825 values `st_ctim.tv_sec` and `st_mtim.tv_sec`, respectively.

12826 The <sys/stat.h> header shall define the following symbolic constants for the file types encoded
 12827 in type **mode_t**. The values shall be suitable for use in **#if** preprocessing directives:

12828	XSI	S_IFMT	Type of file.
12829		S_IFBLK	Block special.

12830	S_IFCHR	Character special.
12831	S_IFIFO	FIFO special.
12832	S_IFREG	Regular.
12833	S_IFDIR	Directory.
12834	S_IFLNK	Symbolic link.
12835	S_IFSOCK	Socket.

12836 The <sys/stat.h> header shall define the following symbolic constants for the file mode bits
 12837 encoded in type **mode_t**, with the indicated numeric values. These macros shall expand to an
 12838 expression which has a type that allows them to be used, either singly or OR'ed together, as the
 12839 third argument to *open()* without the need for a **mode_t** cast. The values shall be suitable for use
 12840 in **#if** preprocessing directives.

12841

12842

12843

12844

12845

12846

12847

12848

12849

12850

12851

12852

12853

12854

12855

12856

12857

12858

12859

12860

12861

12862

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

XSI

12863 The following macros shall be provided to test whether a file is of the specified type. The value
 12864 *m* supplied to the macros is the value of *st_mode* from a **stat** structure. The macro shall evaluate
 12865 to a non-zero value if the test is true; 0 if the test is false.

12866	S_ISBLK(<i>m</i>)	Test for a block special file.
12867	S_ISCHR(<i>m</i>)	Test for a character special file.
12868	S_ISDIR(<i>m</i>)	Test for a directory.
12869	S_ISFIFO(<i>m</i>)	Test for a pipe or FIFO special file.
12870	S_ISREG(<i>m</i>)	Test for a regular file.
12871	S_ISLNK(<i>m</i>)	Test for a symbolic link.
12872	S_ISSOCK(<i>m</i>)	Test for a socket.

12873 The implementation may implement message queues, semaphores, or shared memory objects as
 12874 distinct file types. The following macros shall be provided to test whether a file is of the
 12875 specified type. The value of the *buf* argument supplied to the macros is a pointer to a **stat**
 12876 structure. The macro shall evaluate to a non-zero value if the specified object is implemented as
 12877 a distinct file type and the specified file type is contained in the **stat** structure referenced by *buf*.
 12878 Otherwise, the macro shall evaluate to zero.

12879 S_TYPEISMQ(*buf*) Test for a message queue.

12880 S_TYPEISSEM(*buf*) Test for a semaphore.

12881 S_TYPEISSHM(*buf*) Test for a shared memory object.

12882 TYM The implementation may implement typed memory objects as distinct file types, and the
 12883 following macro shall test whether a file is of the specified type. The value of the *buf* argument
 12884 supplied to the macros is a pointer to a **stat** structure. The macro shall evaluate to a non-zero
 12885 value if the specified object is implemented as a distinct file type and the specified file type is
 12886 contained in the **stat** structure referenced by *buf*. Otherwise, the macro shall evaluate to zero.

12887 S_TYPEISTMO(*buf*) Test macro for a typed memory object.

12888 The following symbolic constants shall be defined as distinct integer values outside of the range +
 12889 [0,999 999 999], for use with the *futimens()* and *utimensat()* functions: +

12890 UTIME_NOW

12891 UTIME_OMIT

12892 The following shall be declared as functions and may also be defined as macros. Function
 12893 prototypes shall be provided.

12894 int chmod(const char *, mode_t);

12895 int fchmod(int, mode_t);

12896 int fchmodat(int, const char *, mode_t, int);

12897 int fstat(int, struct stat *);

12898 int fstatat(int, const char *restrict, struct stat *restrict, int); |

12899 int futimens(int, const struct timespec [2]); |

12900 int lstat(const char *restrict, struct stat *restrict);

12901 int mkdir(const char *, mode_t);

12902 int mkdirat(int, const char *, mode_t);

12903 int mkfifo(const char *, mode_t);

12904 int mkfifoat(int, const char *, mode_t);

12905 XSI int mknod(const char *, mode_t, dev_t);

12906 int mknodat(int, const char *, mode_t, dev_t);

12907 int stat(const char *restrict, struct stat *restrict);

12908 mode_t umask(mode_t);

12909 int utimensat(int, const char *, const struct timespec [2], int); +

APPLICATION USAGE

12910 Use of the macros is recommended for determining the type of a file.

RATIONALE

12912 A conforming C-language application must include <sys/stat.h> for functions that have
 12913 arguments or return values of type **mode_t**, so that symbolic values for that type can be used.
 12914 An alternative would be to require that these constants are also defined by including
 12915 <sys/types.h>.

12917 The S_ISUID and S_ISGID bits may be cleared on any write, not just on *open()*, as some
 12918 historical implementations do.

12919 System calls that update the time entry fields in the **stat** structure must be documented by the
 12920 implementors. POSIX-conforming systems should not update the time entry fields for functions
 12921 listed in the System Interfaces volume of POSIX.1-200x unless the standard requires that they do,
 12922 except in the case of documented extensions to the standard.

12923 Upon assignment, file timestamps are immediately converted to the resolution of the file system +
 12924 by truncation (i.e., the recorded time can be older than the actual time). For example, if the file +

12925 system resolution is 1 microsecond, then a conforming *stat()* must always return an +
 12926 *st_mtim.tv_nsec* that is a multiple of 1000. Some older implementations returned higher- +
 12927 resolution timestamps while the inode information was cached, and then spontaneously +
 12928 truncated the *tv_nsec* fields when they were stored to and retrieved from disk, but this behavior +
 12929 does not conform. +

12930 Note that *st_dev* must be unique within a Local Area Network (LAN) in a “system” made up of +
 12931 multiple computers’ file systems connected by a LAN.

12932 Networked implementations of a POSIX-conforming system must guarantee that all files visible +
 12933 within the file tree (including parts of the tree that may be remotely mounted from other +
 12934 machines on the network) on each individual processor are uniquely identified by the +
 12935 combination of the *st_ino* and *st_dev* fields.

12936 The unit for the *st_blocks* member of the **stat** structure is not defined within POSIX.1-200x. In +
 12937 some implementations it is 512 bytes. It may differ on a file system basis. There is no correlation +
 12938 between values of the *st_blocks* and *st_blksize*, and the *f_bsize* (from <sys/statvfs.h>) structure +
 12939 members.

12940 Traditionally, some implementations defined the multiplier for *st_blocks* in <sys/param.h> as the +
 12941 symbol DEV_BSIZE.

12942 Some earlier versions of this standard did not specify values for the file mode bit macros. The +
 12943 expectation was that some implementors might choose to use a different encoding for these bits +
 12944 than the traditional one, and that new applications would use symbolic file modes instead of +
 12945 numeric. This version of the standard specifies the traditional encoding, in recognition that +
 12946 nearly 20 years after the first publication of this standard numeric file modes are still in +
 12947 widespread use by application writers, and that all conforming implementations still use the +
 12948 traditional encoding. +

12949 FUTURE DIRECTIONS

12950 No new S_IFMT symbolic names for the file type values of **mode_t** will be defined by +
 12951 POSIX.1-200x; if new file types are required, they will only be testable through *S_ISxx()* or +
 12952 *S_TYPEISxxx()* macros instead.

12953 SEE ALSO

12954 <time.h>, <sys/statvfs.h>, <sys/types.h>

12955 XSH *chmod*, *fchmod()*, *fstat()*, *fsstatat()*, *mkdir*, *mkfifo*, *mknod()*, *umask* |

12956 CHANGE HISTORY

12957 First released in Issue 1. Derived from Issue 1 of the SVID.

12958 Issue 5

12959 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

12960 The type of *st_blksize* is changed from **long** to **blksize_t**; the type of *st_blocks* is changed from +
 12961 **long** to **blkcnt_t**.

12962 Issue 6

12963 The *S_TYPEISMQ()*, *S_TYPEISSEM()*, and *S_TYPEISSHM()* macros are unconditionally +
 12964 mandated.

12965 The Open Group Corrigendum U035/4 is applied. In the DESCRIPTION, the types **blksize_t** +
 12966 and **blkcnt_t** have been described.

12967 The following new requirements on POSIX implementations derive from alignment with the +
 12968 Single UNIX Specification:

- 12969 • The **dev_t**, **ino_t**, **mode_t**, **nlink_t**, **uid_t**, **gid_t**, **off_t**, and **time_t** types are mandated.

12970 **S_IFSOCK** and **S_ISSOCK** are added for sockets.

12971	The description of stat structure members is changed to reflect contents when file type is a	
12972	symbolic link.	
12973	The test macro <code>S_TYPEISTMO</code> is added for alignment with IEEE Std 1003.1j-2000.	
12974	The restrict keyword is added to the prototypes for <code>lstat()</code> and <code>stat()</code> .	
12975	The <code>lstat()</code> function is made mandatory.	
12976	IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/17 is applied, adding text regarding the	
12977	<code>st_blocks</code> member of the stat structure to the RATIONALE.	
12978	IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/25 is applied, adding to the	
12979	DESCRIPTION that the timespec structure may be defined as described in the < time.h > header.	
12980	Issue 7	
12981	SD5-XSH-ERN-161 is applied, updating the DESCRIPTION to clarify that the descriptions of the	+
12982	interfaces should be consulted in order to determine which structure members have meaningful	+
12983	values.	+
12984	The <code>fchmodat()</code> , <code>fstatat()</code> , <code>mkdirat()</code> , <code>mkfifoat()</code> , <code>mknodat()</code> , and <code>utimensat()</code> functions are added	
12985	from The Open Group Technical Standard, 2006, Extended API Set Part 2.	
12986	The <code>futimens()</code> function is added.	
12987	This reference page is clarified with respect to macros and symbolic constants.	
12988	Changes are made related to support for finegrained timestamps and the <code>UTIME_NOW</code> and	
12989	<code>UTIME_OMIT</code> symbolic constants are added.	

12990 **NAME**
 12991 `sys/statvfs.h` — VFS File System information structure

12992 **SYNOPSIS**
 12993 `#include <sys/statvfs.h>`

12994 **DESCRIPTION**
 12995 The `<sys/statvfs.h>` header shall define the `statvfs` structure that includes at least the following
 12996 members:

12997	<code>unsigned long</code>	<code>f_bsize</code>	File system block size.
12998	<code>unsigned long</code>	<code>f_frsize</code>	Fundamental file system block size.
12999	<code>fsblkcnt_t</code>	<code>f_blocks</code>	Total number of blocks on file system in units of <i>f_frsize</i> .
13000	<code>fsblkcnt_t</code>	<code>f_bfree</code>	Total number of free blocks.
13001	<code>fsblkcnt_t</code>	<code>f_bavail</code>	Number of free blocks available to non-privileged process.
13002			
13003	<code>fsfilcnt_t</code>	<code>f_files</code>	Total number of file serial numbers.
13004	<code>fsfilcnt_t</code>	<code>f_ffree</code>	Total number of free file serial numbers.
13005	<code>fsfilcnt_t</code>	<code>f_favail</code>	Number of file serial numbers available to non-privileged process.
13006			
13007	<code>unsigned long</code>	<code>f_fsid</code>	File system ID.
13008	<code>unsigned long</code>	<code>f_flag</code>	Bit mask of <i>f_flag</i> values.
13009	<code>unsigned long</code>	<code>f_namemax</code>	Maximum filename length.

13010 The `fsblkcnt_t` and `fsfilcnt_t` types shall be defined as described in `<sys/types.h>`.

13011 The `<sys/statvfs.h>` header shall define the following symbolic constants for the *f_flag* member:

13012	<code>ST_RDONLY</code>	Read-only file system.
13013	<code>ST_NOSUID</code>	Does not support the semantics of the <code>ST_ISUID</code> and <code>ST_ISGID</code> file mode bits.

13014 The following shall be declared as functions and may also be defined as macros. Function
 13015 prototypes shall be provided.

```
13016 int fstatvfs(int, struct statvfs *); -
13017 int statvfs(const char *restrict, struct statvfs *restrict); +
```

13018 **APPLICATION USAGE**
 13019 None.

13020 **RATIONALE**
 13021 None.

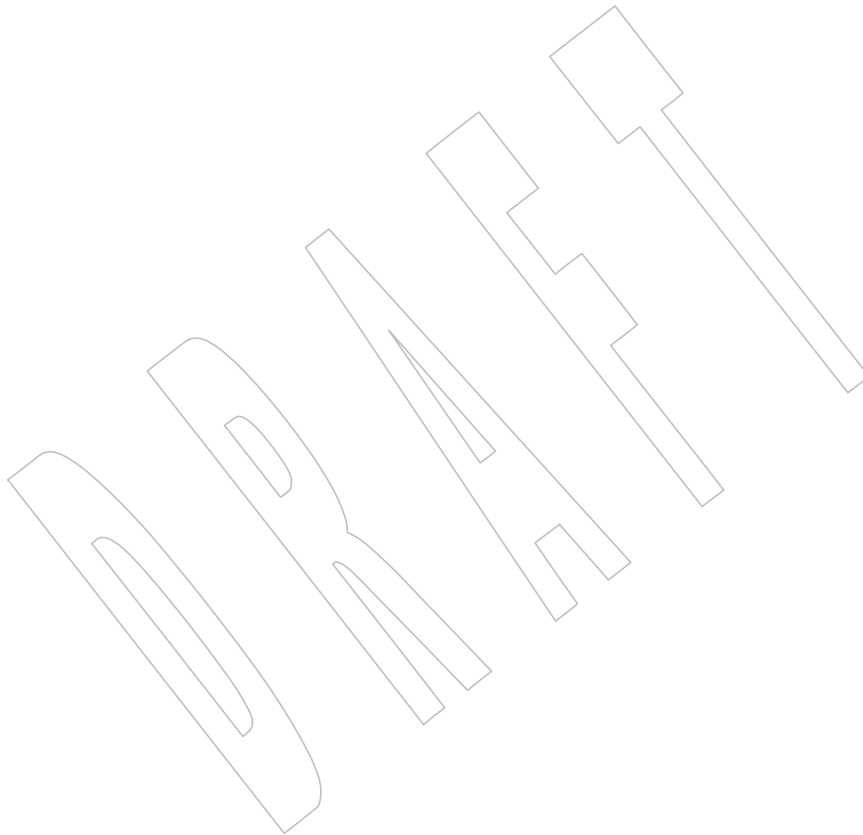
13022 **FUTURE DIRECTIONS**
 13023 None.

13024 **SEE ALSO**
 13025 [<sys/types.h>](#)
 13026 XSH [fstatvfs\(\)](#)

13027 **CHANGE HISTORY**
 13028 First released in Issue 4, Version 2.

13029 **Issue 5**
 13030 The type of *f_blocks*, *f_bfree*, and *f_bavail* is changed from **unsigned long** to **fsblkcnt_t**; the type of
 13031 *f_files*, *f_ffree*, and *f_favail* is changed from **unsigned long** to **fsfilcnt_t**.

- 13032 **Issue 6**
- 13033 The Open Group Corrigendum U035/5 is applied. In the DESCRIPTION, the types **fsblkcnt_t**
- 13034 and **fsfilcnt_t** have been described.
- 13035 The **restrict** keyword is added to the prototype for *statvfs()*.
- 13036 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/18 is applied, changing the description of
- 13037 ST_NOSUID from “Does not support *setuid()/setgid()* semantics” to “Does not support the
- 13038 semantics of the ST_ISUID and ST_ISGID file mode bits”.
- 13039 **Issue 7**
- 13040 The **<sys/statvfs.h>** header is moved from the XSI option to the Base.
- 13041 This reference page is clarified with respect to macros and symbolic constants. +



13042 **NAME**

13043 sys/time.h — time types

13044 **SYNOPSIS**13045 XSI `#include <sys/time.h>`13046 **DESCRIPTION**13047 The <sys/time.h> header shall define the **timeval** structure that includes at least the following
13048 members:

13049	time_t	tv_sec	Seconds.
13050	suseconds_t	tv_usec	Microseconds.

13051 OB The <sys/time.h> header shall define the **itimerval** structure that includes at least the following
13052 members:

13053	struct timeval	it_interval	Timer interval.
13054	struct timeval	it_value	Current value.

13055 The **time_t** and **suseconds_t** types shall be defined as described in <sys/types.h>.13056 The **fd_set** type shall be defined as described in <sys/select.h>.13057 OB The <sys/time.h> header shall define the following symbolic constants for the *which* argument of
13058 *getitimer()* and *setitimer()*:

13059	ITIMER_REAL	Decrements in real time.
13060	ITIMER_VIRTUAL	Decrements in process virtual time.
13061	ITIMER_PROF	Decrements both in process virtual time and when the system is running 13062 on behalf of the process.

13063 The following shall be defined as described in <sys/select.h>:

```
13064 FD_CLR()
13065 FD_ISSET()
13066 FD_SET()
13067 FD_ZERO()
13068 FD_SETSIZE
```

13069 The following shall be declared as functions and may also be defined as macros. Function
13070 prototypes shall be provided.

```
13071 OB int getitimer(int, struct itimerval *);
13072 int gettimeofday(struct timeval *restrict, void *restrict);
13073 int setitimer(int, const struct itimerval *restrict,
13074             struct itimerval *restrict);
13075 int select(int, fd_set *restrict, fd_set *restrict, fd_set *restrict,
13076           struct timeval *restrict);
13077 int utimes(const char *, const struct timeval [2]);
```

13078 Inclusion of the <sys/time.h> header may make visible all symbols from the <sys/select.h>
13079 header.

13080	APPLICATION USAGE	
13081	None.	
13082	RATIONALE	
13083	None.	
13084	FUTURE DIRECTIONS	
13085	None.	
13086	SEE ALSO	
13087	<sys/select.h> , <sys/types.h>	
13088	XSH getitimer() , gettimeofday() , pselect()	
13089	CHANGE HISTORY	
13090	First released in Issue 4, Version 2.	
13091	Issue 5	
13092	The type of <i>tv_usec</i> is changed from long to suseconds_t .	
13093	Issue 6	
13094	The restrict keyword is added to the prototypes for gettimeofday() , select() , and setitimer() .	
13095	The note is added that inclusion of this header may also make symbols visible from	
13096	<sys/select.h> .	
13097	The utimes() function is marked LEGACY.	
13098	Issue 7	
13099	This reference page is clarified with respect to macros and symbolic constants.	

13100 **NAME**
 13101 `sys/times.h` — file access and modification times structure

13102 **SYNOPSIS**
 13103 `#include <sys/times.h>`

13104 **DESCRIPTION**
 13105 The **<sys/times.h>** header shall define the structure **tms**, which is returned by `times()` and
 13106 includes at least the following members:

13107 `clock_t tms_utime` User CPU time.
 13108 `clock_t tms_stime` System CPU time.
 13109 `clock_t tms_cutime` User CPU time of terminated child processes.
 13110 `clock_t tms_cstime` System CPU time of terminated child processes.

13111 The **clock_t** type shall be defined as described in **<sys/types.h>**.

13112 The following shall be declared as a function and may also be defined as a macro. A function
 13113 prototype shall be provided.

13114 `clock_t times(struct tms *);`

13115 **APPLICATION USAGE**
 13116 None.

13117 **RATIONALE**
 13118 None.

13119 **FUTURE DIRECTIONS**
 13120 None.

13121 **SEE ALSO**
 13122 [<sys/types.h>](#)

13123 XSH `times()`

13124 **CHANGE HISTORY**
 13125 First released in Issue 1. Derived from Issue 1 of the SVID.

13126		NAME	
13127			sys/types.h — data types
13128		SYNOPSIS	
13129			#include <sys/types.h>
13130		DESCRIPTION	
13131			The <sys/types.h> header shall include definitions for at least the following types:
13132		blkcnt_t	Used for file block counts.
13133		blksize_t	Used for block sizes.
13134		clock_t	Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.
13135			
13136		clockid_t	Used for clock ID type in the clock and timer functions.
13137		dev_t	Used for device IDs.
13138	XSI	fsblkcnt_t	Used for file system block counts.
13139	XSI	fsfilcnt_t	Used for file system file counts.
13140		gid_t	Used for group IDs.
13141		id_t	Used as a general identifier; can be used to contain at least a pid_t , uid_t , or gid_t .
13142			
13143		ino_t	Used for file serial numbers.
13144	XSI	key_t	Used for XSI interprocess communication.
13145		mode_t	Used for some file attributes.
13146		nlink_t	Used for link counts.
13147		off_t	Used for file sizes.
13148		pid_t	Used for process IDs and process group IDs.
13149		pthread_attr_t	Used to identify a thread attribute object.
13150		pthread_barrier_t	Used to identify a barrier.
13151		pthread_barrierattr_t	Used to define a barrier attributes object.
13152		pthread_cond_t	Used for condition variables.
13153		pthread_condattr_t	Used to identify a condition attribute object.
13154		pthread_key_t	Used for thread-specific data keys.
13155		pthread_mutex_t	Used for mutexes.
13156		pthread_mutexattr_t	Used to identify a mutex attribute object.
13157		pthread_once_t	Used for dynamic package initialization.
13158		pthread_rwlock_t	Used for read-write locks.
13159		pthread_rwlockattr_t	Used for read-write lock attributes.
13160		pthread_spinlock_t	Used to identify a spin lock.

13161		pthread_t	Used to identify a thread.
13162		size_t	Used for sizes of objects.
13163		ssize_t	Used for a count of bytes or an error indication.
13164	XSI	suseconds_t	Used for time in microseconds.
13165		time_t	Used for time in seconds.
13166		timer_t	Used for timer ID returned by <i>timer_create()</i> .
13167	OB TRC		Also used to identify a trace stream attributes object.
13168	OB TRC	trace_event_id_t	Used to identify a trace event type.
13169	OB TEF	trace_event_set_t	Used to identify a trace event type set.
13170	OB TRC	trace_id_t	Used to identify a trace stream.
13171		uid_t	Used for user IDs.
13172		All of the types shall be defined as arithmetic types of an appropriate length, with the following exceptions:	
13173			
13174		pthread_attr_t	
13175		pthread_barrier_t	
13176		pthread_barrierattr_t	
13177		pthread_cond_t	
13178		pthread_condattr_t	
13179		pthread_key_t	
13180		pthread_mutex_t	
13181		pthread_mutexattr_t	
13182		pthread_once_t	
13183		pthread_rwlock_t	
13184		pthread_rwlockattr_t	
13185		pthread_spinlock_t	
13186		pthread_t	
13187	OB TRC	trace_attr_t	
13188		trace_event_id_t	
13189	OB TEF	trace_event_set_t	
13190	OB TRC	trace_id_t	
13191		Additionally:	
13192		• mode_t shall be an integer type.	
13193		• nlink_t , uid_t , gid_t , and id_t shall be integer types.	
13194		• blkcnt_t and off_t shall be signed integer types.	
13195	XSI	• fsblkcnt_t , fsfilcnt_t , and ino_t shall be defined as unsigned integer types.	
13196		• size_t shall be an unsigned integer type.	
13197		• blksize_t , pid_t , and ssize_t shall be signed integer types.	
13198		• time_t and clock_t shall be integer or real-floating types.	
13199	XSI	The type ssize_t shall be capable of storing values at least in the range $[-1, \{SSIZE_MAX\}]$. The	
13200		type suseconds_t shall be a signed integer type capable of storing values at least in the range	
13201		$[-1, 1\,000\,000]$.	
13202		The implementation shall support one or more programming environments in which the widths	

13203 of **blksize_t**, **pid_t**, **size_t**, **ssize_t**, and **suseconds_t** are no greater than the width of type **long**.
 13204 The names of these programming environments can be obtained using the *confstr()* function or
 13205 the *getconf* utility.

13206 There are no defined comparison or assignment operators for the following types:

13207 **pthread_attr_t**
 13208 **pthread_barrier_t**
 13209 **pthread_barrierattr_t**
 13210 **pthread_cond_t**
 13211 **pthread_condattr_t**
 13212 **pthread_mutex_t**
 13213 **pthread_mutexattr_t**
 13214 **pthread_rwlock_t**
 13215 **pthread_rwlockattr_t**
 13216 **pthread_spinlock_t**
 13217 OB TRC **trace_attr_t**

13218 APPLICATION USAGE

13219 None.

13220 RATIONALE

13221 None.

13222 FUTURE DIRECTIONS

13223 None.

13224 SEE ALSO

13225 [<time.h>](#)

13226 XSH *confstr()*

13227 XCU *getconf*

13228 CHANGE HISTORY

13229 First released in Issue 1. Derived from Issue 1 of the SVID.

13230 Issue 5

13231 The **clockid_t** and **timer_t** types are defined for alignment with the POSIX Realtime Extension.

13232 The types **blkcnt_t**, **blksize_t**, **fsblkcnt_t**, **fsfilcnt_t**, and **suseconds_t** are added.

13233 Large File System extensions are added.

13234 Updated for alignment with the POSIX Threads Extension.

13235 Issue 6

13236 The **pthread_barrier_t**, **pthread_barrierattr_t**, and **pthread_spinlock_t** types are added for
 13237 alignment with IEEE Std 1003.1j-2000.

13238 The margin code is changed from XSI to THR for the **pthread_rwlock_t** and
 13239 **pthread_rwlockattr_t** types as Read-Write Locks have been absorbed into the POSIX Threads
 13240 option. The threads types are marked THR.

13241 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/26 is applied, adding **pthread_t** to the list
 13242 of types that are not required to be arithmetic types, thus allowing **pthread_t** to be defined as a
 13243 structure.

13244
13245
13246
13247
13248
13249
13250
13251

Issue 7

Austin Group Interpretation 1003.1-2001 #033 is applied.

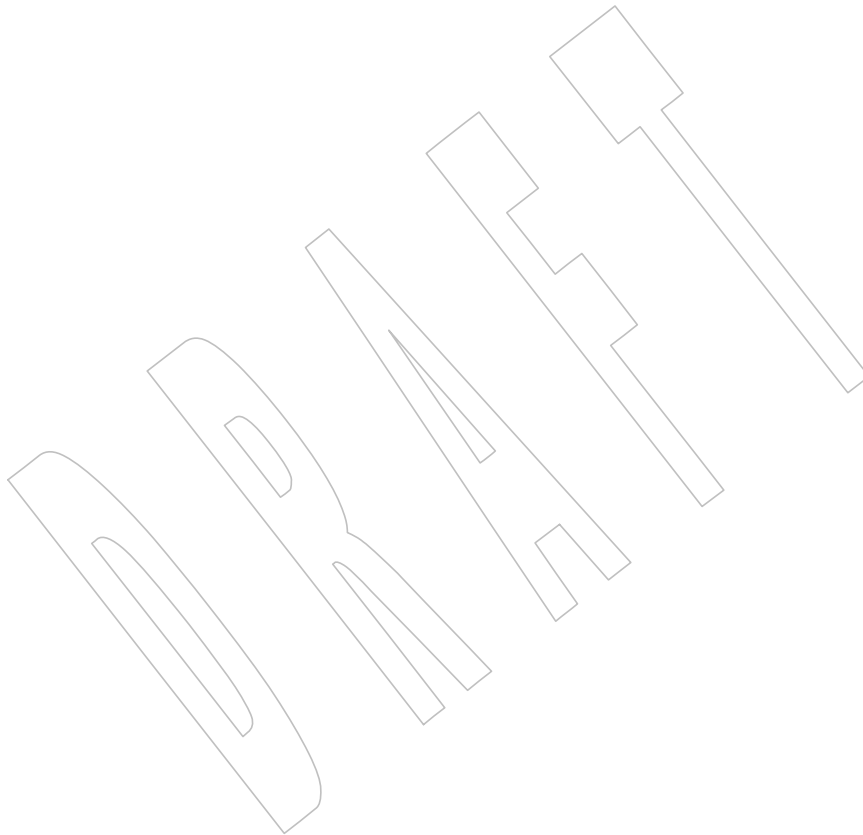
The Trace option types are marked obsolescent.

The **clock_t** and **id_t** types are moved from the XSI option to the Base.

The **pthread_barrier_t** and **pthread_barrierattr_t** types are moved from the Barriers option to the Base.

The **pthread_spinlock_t** type is moved from the Spin Locks option to the Base.

Functionality relating to the Timers and Threads options is moved to the Base.



13252 **NAME**
 13253 sys/uio.h — definitions for vector I/O operations

13254 **SYNOPSIS**
 13255 XSI `#include <sys/uio.h>`

13256 **DESCRIPTION**
 13257 The <sys/uio.h> header shall define the **iovec** structure that includes at least the following
 13258 members:

13259 `void *iov_base` Base address of a memory region for input or output.
 13260 `size_t iov_len` The size of the memory pointed to by `iov_base`.

13261 The <sys/uio.h> header uses the **iovec** structure for scatter/gather I/O.

13262 The **ssize_t** and **size_t** types shall be defined as described in <sys/types.h>.

13263 The following shall be declared as functions and may also be defined as macros. Function
 13264 prototypes shall be provided.

13265 `ssize_t readv(int, const struct iovec *, int);`
 13266 `ssize_t writev(int, const struct iovec *, int);`

13267 **APPLICATION USAGE**
 13268 The implementation can put a limit on the number of scatter/gather elements which can be
 13269 processed in one call. The symbol {IOV_MAX} defined in <limits.h> should always be used to
 13270 learn about the limits instead of assuming a fixed value.

13271 **RATIONALE**
 13272 Traditionally, the maximum number of scatter/gather elements the system can process in one
 13273 call were described by the symbolic value {UIO_MAXIOV}. In IEEE Std 1003.1-2001 this value is
 13274 replaced by the constant {IOV_MAX} which can be found in <limits.h>.

13275 **FUTURE DIRECTIONS**
 13276 None.

13277 **SEE ALSO**
 13278 [<limits.h>](#), [<sys/types.h>](#)

13279 XSH *read*, *write*

13280 **CHANGE HISTORY**
 13281 First released in Issue 4, Version 2.

13282 **Issue 6**
 13283 Text referring to scatter/gather I/O is added to the DESCRIPTION.

13284 **NAME**
 13285 sys/un.h — definitions for UNIX domain sockets

13286 **SYNOPSIS**
 13287 #include <sys/un.h>

13288 **DESCRIPTION**
 13289 The <sys/un.h> header shall define the **sockaddr_un** structure that includes at least the
 13290 following members:

13291 sa_family_t sun_family Address family.
 13292 char sun_path[] Socket pathname.

13293 The **sockaddr_un** structure is used to store addresses for UNIX domain sockets. Values of this
 13294 type shall be cast by applications to **struct sockaddr** for use with socket functions.

13295 The **sa_family_t** type shall be defined as described in <sys/socket.h>.

13296 **APPLICATION USAGE**
 13297 The size of *sun_path* has intentionally been left undefined. This is because different
 13298 implementations use different sizes. For example, 4.3 BSD uses a size of 108, and 4.4 BSD uses a
 13299 size of 104. Since most implementations originate from BSD versions, the size is typically in the
 13300 range 92 to 108.

13301 Applications should not assume a particular length for *sun_path* or assume that it can hold
 13302 {_POSIX_PATH_MAX} characters (255).

13303 **RATIONALE**
 13304 None.

13305 **FUTURE DIRECTIONS**
 13306 None.

13307 **SEE ALSO**
 13308 <sys/socket.h>
 13309 XSH *bind()*, *socket()*, *socketpair()*

13310 **CHANGE HISTORY**
 13311 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

13312 **NAME**
 13313 sys/utsname.h — system name structure

13314 **SYNOPSIS**
 13315 #include <sys/utsname.h>

13316 **DESCRIPTION**
 13317 The <sys/utsname.h> header shall define the structure **utsname** which shall include at least the
 13318 following members:

13319 char sysname[] Name of this implementation of the operating system.
 13320 char nodename[] Name of this node within the communications
 13321 network to which this node is attached, if any.
 13322 char release[] Current release level of this implementation.
 13323 char version[] Current version level of this release.
 13324 char machine[] Name of the hardware type on which the system is running.

13325 The character arrays are of unspecified size, but the data stored in them shall be terminated by a
 13326 null byte.

13327 The following shall be declared as a function and may also be defined as a macro:

13328 int uname(struct utsname *);

13329 **APPLICATION USAGE**
 13330 None.

13331 **RATIONALE**
 13332 None.

13333 **FUTURE DIRECTIONS**
 13334 None.

13335 **SEE ALSO**
 13336 XSH *uname*

13337 **CHANGE HISTORY**
 13338 First released in Issue 1. Derived from Issue 1 of the SVID.

13339 **Issue 6**
 13340 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/27 is applied, changing the description of
 13341 *nodename* within the **utsname** structure from “an implementation-defined communications
 13342 network” to “the communications network to which this node is attached, if any”.

13343 **NAME**
 13344 `sys/wait.h` — declarations for waiting

13345 **SYNOPSIS**
 13346 `#include <sys/wait.h>`

13347 **DESCRIPTION**
 13348 The **<sys/wait.h>** header shall define the following symbolic constants for use with `waitpid()`:

13349 `WNOHANG` Do not hang if no status is available; return immediately.
 13350 `WUNTRACED` Report status of stopped child process.

13351 The **<sys/wait.h>** header shall define the following macros for analysis of process status values:

13352 `WEXITSTATUS` Return exit status.

13353 XSI `WIFCONTINUED` True if child has been continued.
 13354 `WIFEXITED` True if child exited normally.
 13355 `WIFSIGNALED` True if child exited due to uncaught signal.
 13356 `WIFSTOPPED` True if child is currently stopped.
 13357 `WSTOPSIG` Return signal number that caused process to stop.
 13358 `WTERMSIG` Return signal number that caused process to terminate.

13359 The following symbolic constants shall be defined as possible values for the *options* argument to
 13360 `waitid()`:

13361 `WEXITED` Wait for processes that have exited.
 13362 `WSTOPPED` Status is returned for any child that has stopped upon receipt of a signal.
 13363 XSI `WCONTINUED` Status is returned for any child that was stopped and has been continued.

13364 `WNOHANG` Return immediately if there are no children to wait for.
 13365 `WNOWAIT` Keep the process whose status is returned in *infop* in a waitable state.

13366 The type **idtype_t** shall be defined as an enumeration type whose possible values shall include
 13367 at least the following:
 13368 `P_ALL`
 13369 `P_PID`
 13370 `P_PGID`

13371 The **id_t** and **pid_t** types shall be defined as described in **<sys/types.h>**.
 13372 The **siginfo_t** type shall be defined as described in **<signal.h>**.
 13373 Inclusion of the **<sys/wait.h>** header may also make visible all symbols from **<signal.h>**.
 13374 The following shall be declared as functions and may also be defined as macros. Function
 13375 prototypes shall be provided.
 13376 `pid_t wait(int *);`
 13377 `int waitid(idtype_t, id_t, siginfo_t *, int);`
 13378 `pid_t waitpid(pid_t, int *, int);`

13379 **APPLICATION USAGE**

13380 None.

13381 **RATIONALE**

13382 None.

13383 **FUTURE DIRECTIONS**

13384 None.

13385 **SEE ALSO**13386 [<signal.h>](#), [<sys/types.h>](#)13387 XSH *wait*, *waitid()*13388 **CHANGE HISTORY**

13389 First released in Issue 3.

13390 Included for alignment with the POSIX.1-1988 standard.

13391 **Issue 6**13392 The *wait3()* function is removed.13393 **Issue 7**13394 The *waitid()* function and symbolic constants for its options argument are moved to the Base.

DRAFT

13395 **NAME**
 13396 syslog.h — definitions for system error logging

13397 **SYNOPSIS**

13398 XSI `#include <syslog.h>`

13399 **DESCRIPTION**

13400 The <syslog.h> header shall define the following symbolic constants, zero or more of which
 13401 may be OR'ed together to form the *logopt* option of *openlog()*:

13402 LOG_PID Log the process ID with each message.

13403 LOG_CONS Log to the system console on error.

13404 LOG_NDELAY Connect to syslog daemon immediately.

13405 LOG_ODELAY Delay open until *syslog()* is called.

13406 LOG_NOWAIT Do not wait for child processes.

13407 The <syslog.h> header shall define the following symbolic constants for use as the *facility* |
 13408 argument to *openlog()*:

13409 LOG_KERN Reserved for message generated by the system.

13410 LOG_USER Message generated by a process.

13411 LOG_MAIL Reserved for message generated by mail system.

13412 LOG_NEWS Reserved for message generated by news system.

13413 LOG_UUCP Reserved for message generated by UUCP system.

13414 LOG_DAEMON Reserved for message generated by system daemon.

13415 LOG_AUTH Reserved for message generated by authorization daemon.

13416 LOG_CRON Reserved for message generated by clock daemon.

13417 LOG_LPR Reserved for message generated by printer system.

13418 LOG_LOCAL0 Reserved for local use.

13419 LOG_LOCAL1 Reserved for local use.

13420 LOG_LOCAL2 Reserved for local use.

13421 LOG_LOCAL3 Reserved for local use.

13422 LOG_LOCAL4 Reserved for local use.

13423 LOG_LOCAL5 Reserved for local use.

13424 LOG_LOCAL6 Reserved for local use.

13425 LOG_LOCAL7 Reserved for local use.

13426 The <syslog.h> header shall define the following macros for constructing the *maskpri* argument |
 13427 to *setlogmask()*. The following macros expand to an expression of type **int** when the argument
 13428 *pri* is an expression of type **int**:

13429 LOG_MASK(*pri*) A mask for priority *pri*.

13430 The <syslog.h> header shall define the following symbolic constants for use as the *priority* |
 13431 argument of *syslog()*:

13432	LOG_EMERG	A panic condition was reported to all processes.	
13433	LOG_ALERT	A condition that should be corrected immediately.	
13434	LOG_CRIT	A critical condition.	
13435	LOG_ERR	An error message.	
13436	LOG_WARNING	A warning message.	
13437	LOG_NOTICE	A condition requiring special handling.	
13438	LOG_INFO	A general information message.	
13439	LOG_DEBUG	A message useful for debugging programs.	
13440		The following shall be declared as functions and may also be defined as macros. Function	
13441		prototypes shall be provided.	
13442		<code>void closelog(void);</code>	
13443		<code>void openlog(const char *, int, int);</code>	
13444		<code>int setlogmask(int);</code>	
13445		<code>void syslog(int, const char *, ...);</code>	
13446	APPLICATION USAGE		
13447		None.	
13448	RATIONALE		
13449		None.	
13450	FUTURE DIRECTIONS		
13451		None.	
13452	SEE ALSO		
13453		XSH <i>closelog()</i>	
13454	CHANGE HISTORY		
13455		First released in Issue 4, Version 2.	
13456	Issue 5		
13457		Moved from X/Open UNIX to BASE.	
13458	Issue 7		
13459		This reference page is clarified with respect to macros and symbolic constants.	+

13460 **NAME**
 13461 tar.h — extended tar definitions

13462 **SYNOPSIS**
 13463 #include <tar.h>

13464 **DESCRIPTION**
 13465 The <tar.h> header shall define the following symbolic constants with the indicated values.

13466 General definitions:

Name	Value	Description
TMAGIC	"ustar"	ustar plus null byte.
TMAGLEN	6	Length of the above.
TVERSION	"00"	00 without a null byte.
TVERSLN	2	Length of the above.

13472 *Typeflag* field definitions:

Name	Value	Description
REGTYPE	'0'	Regular file.
AREGTYPE	'\0'	Regular file.
LNKTYPE	'1'	Link.
SYMTYPE	'2'	Symbolic link.
CHRTYPE	'3'	Character special.
BLKTYPE	'4'	Block special.
DIRTYPE	'5'	Directory.
FIFOTYPE	'6'	FIFO special.
CONTTYPE	'7'	Reserved.

13483 *Mode* field bit definitions (octal):

Name	Value	Description
TSUID	04000	Set UID on execution.
TSGID	02000	Set GID on execution.
TSVTX	01000	On directories, restricted deletion flag.
TUREAD	00400	Read by owner.
TUWRITE	00200	Write by owner special.
TUEXEC	00100	Execute/search by owner.
TGREAD	00040	Read by group.
TGWRITE	00020	Write by group.
TGEXEC	00010	Execute/search by group.
TOREAD	00004	Read by other.
TOWRITE	00002	Write by other.
TOEXEC	00001	Execute/search by other.

13497	APPLICATION USAGE	
13498	None.	
13499	RATIONALE	
13500	None.	
13501	FUTURE DIRECTIONS	
13502	None.	
13503	SEE ALSO	
13504	XCU <i>pax</i>	
13505	CHANGE HISTORY	
13506	First released in Issue 3. Derived from the POSIX.1-1988 standard.	
13507	Issue 6	
13508	The SEE ALSO section is updated to refer to <i>pax</i> .	
13509	Issue 7	
13510	This reference page is clarified with respect to macros and symbolic constants.	+

DRAFT

13511 **NAME**
 13512 `termios.h` — define values for `termios`

13513 **SYNOPSIS**
 13514 `#include <termios.h>`

13515 **DESCRIPTION**
 13516 The **<termios.h>** header shall contain the definitions used by the terminal I/O interfaces (see
 13517 [Chapter 11](#) (on page 185) for the structures and names defined).

13518 **The `termios` Structure**

13519 The following data types shall be defined through **typedef**:

13520 **cc_t** Used for terminal special characters.

13521 **speed_t** Used for terminal baud rates.

13522 **tcflag_t** Used for terminal modes.

13523 The above types shall be all unsigned integer types.

13524 The implementation shall support one or more programming environments in which the widths
 13525 of **cc_t**, **speed_t**, and **tcflag_t** are no greater than the width of type **long**. The names of these
 13526 programming environments can be obtained using the `confstr()` function or the `getconf` utility.

13527 The **termios** structure shall be defined, and shall include at least the following members:

13528 `tcflag_t c_iflag` Input modes.
 13529 `tcflag_t c_oflag` Output modes.
 13530 `tcflag_t c_cflag` Control modes.
 13531 `tcflag_t c_lflag` Local modes.
 13532 `cc_t c_cc[NCCS]` Control characters.

13533 The **<termios.h>** header shall define the following symbolic constant:

13534 **NCCS** Size of the array `c_cc` for control characters.

13535 The **<termios.h>** header shall define the following symbolic constants for use as subscripts for
 13536 the array `c_cc`:

Subscript Usage		Description
Canonical Mode	Non-Canonical Mode	
VEOF		EOF character.
VEOL		EOL character.
VERASE		ERASE character.
VINTR	VINTR	INTR character.
VKILL		KILL character.
	VMIN	MIN value.
VQUIT	VQUIT	QUIT character.
VSTART	VSTART	START character.
VSTOP	VSTOP	STOP character.
VSUSP	VSUSP	SUSP character.
	VTIME	TIME value.

13550 The subscript values shall be suitable for use in **#if** preprocessing directives and shall be distinct,
 13551 except that the **VMIN** and **VTIME** subscripts may have the same values as the **VEOF** and **VEOL**
 13552 subscripts, respectively.

13553 **Input Modes**

13554 The <termios.h> header shall define the following symbolic constants for use as flags in the +
 13555 *c_iflag* field. The *c_iflag* field describes the basic terminal input control. |

13556		BRKINT	Signal interrupt on break.
13557		ICRNL	Map CR to NL on input.
13558		IGNBRK	Ignore break condition.
13559		IGNCR	Ignore CR.
13560		IGNPAR	Ignore characters with parity errors.
13561		INLCR	Map NL to CR on input.
13562		INPCK	Enable input parity check.
13563		ISTRIP	Strip character.
13564	XSI	IXANY	Enable any character to restart output.
13565		IXOFF	Enable start/stop input control.
13566		IXON	Enable start/stop output control.
13567		PARMRK	Mark parity errors.

13568 **Output Modes**

13569 The <termios.h> header shall define the following symbolic constants for use as flags in the +
 13570 *c_oflag* field. The *c_oflag* field specifies the system treatment of output. |

13571		OPOST	Post-process output.
13572	XSI	ONLCR	Map NL to CR-NL on output.
13573	XSI	OCRNL	Map CR to NL on output.
13574	XSI	ONOCR	No CR output at column 0.
13575	XSI	ONLRET	NL performs CR function.
13576	XSI	OFDEL	Fill is DEL.
13577	XSI	OFILL	Use fill characters for delay.
13578	XSI	NLDLY	Select newline delays:
13579		NL0	Newline type 0.
13580		NL1	Newline type 1.
13581	XSI	CRDLY	Select carriage-return delays:
13582		CR0	Carriage-return delay type 0.
13583		CR1	Carriage-return delay type 1.
13584		CR2	Carriage-return delay type 2.
13585		CR3	Carriage-return delay type 3.
13586	XSI	TABDLY	Select horizontal-tab delays:

13587		TAB0	Horizontal-tab delay type 0.
13588		TAB1	Horizontal-tab delay type 1.
13589		TAB2	Horizontal-tab delay type 2.
13590		TAB3	Expand tabs to spaces.
13591	XSI	BSDLY	Select backspace delays:
13592		BS0	Backspace-delay type 0.
13593		BS1	Backspace-delay type 1.
13594	XSI	VTDLY	Select vertical-tab delays:
13595		VT0	Vertical-tab delay type 0.
13596		VT1	Vertical-tab delay type 1.
13597	XSI	FFDLY	Select form-feed delays:
13598		FF0	Form-feed delay type 0.
13599		FF1	Form-feed delay type 1.

13600 Baud Rate Selection

13601 The <termios.h> header shall define the following symbolic constants for use as values of
 13602 objects of type **speed_t**.

13603 The input and output baud rates are stored in the **termios** structure. These are the valid values
 13604 for objects of type **speed_t**. Not all baud rates need be supported by the underlying hardware.

13605	B0	Hang up
13606	B50	50 baud
13607	B75	75 baud
13608	B110	110 baud
13609	B134	134.5 baud
13610	B150	150 baud
13611	B200	200 baud
13612	B300	300 baud
13613	B600	600 baud
13614	B1200	1 200 baud
13615	B1800	1 800 baud
13616	B2400	2 400 baud
13617	B4800	4 800 baud
13618	B9600	9 600 baud
13619	B19200	19 200 baud

13620	B38400	38 400 baud	
13621	Control Modes		-
13622	The <termios.h> header shall define the following symbolic constants for use as flags in the		
13623	<i>c_cflag</i> field. The <i>c_cflag</i> field describes the hardware control of the terminal; not all values		
13624	specified are required to be supported by the underlying hardware.		
13625	CSIZE	Character size:	
13626		CS5 5 bits	
13627		CS6 6 bits	
13628		CS7 7 bits	
13629		CS8 8 bits	
13630	CSTOPB	Send two stop bits, else one.	
13631	CREAD	Enable receiver.	
13632	PARENB	Parity enable.	
13633	PARODD	Odd parity, else even.	
13634	HUPCL	Hang up on last close.	
13635	CLOCAL	Ignore modem status lines.	
13636	The implementation shall support the functionality associated with the symbols CS7, CS8,		
13637	CSTOPB, PARODD, and PARENB.		
13638	Local Modes		-
13639	The <termios.h> header shall define the following symbolic constants for use as flags in the		
13640	<i>c_lflag</i> field. The <i>c_lflag</i> field of the argument structure is used to control various terminal		
13641	functions.		
13642	ECHO	Enable echo.	
13643	ECHOE	Echo erase character as error-correcting backspace.	
13644	ECHOK	Echo KILL.	
13645	ECHONL	Echo NL.	
13646	ICANON	Canonical input (erase and kill processing).	
13647	IEXTEN	Enable extended input character processing.	
13648	ISIG	Enable signals.	
13649	NOFLSH	Disable flush after interrupt or quit.	
13650	TOSTOP	Send SIGTTOU for background output.	
13651	Attribute Selection		-
13652	The <termios.h> header shall define the following symbolic constants for use with <i>tcsetattr()</i> :		
13653	TCSANOW	Change attributes immediately.	
13654	TCSADRAIN	Change attributes when output has drained.	
13655	TCSAFLUSH	Change attributes when output has drained; also flush pending input.	

13656 **Line Control** -13657 The <termios.h> header shall define the following symbolic constants for use with *tcflush()*: |

13658 TCIFLUSH Flush pending input.

13659 TCIOFLUSH Flush both pending input and untransmitted output.

13660 TCOFLUSH Flush untransmitted output.

13661 The <termios.h> header shall define the following symbolic constants for use with *tcflow()*: |

13662 TCIOFF Transmit a STOP character, intended to suspend input data.

13663 TCION Transmit a START character, intended to restart input data.

13664 TCOOFF Suspend output.

13665 TCOON Restart output.

13666 The following shall be declared as functions and may also be defined as macros. Function
13667 prototypes shall be provided.13668 `speed_t cfgetispeed(const struct termios *);`13669 `speed_t cfgetospeed(const struct termios *);`13670 `int cfsetispeed(struct termios *, speed_t);`13671 `int cfsetospeed(struct termios *, speed_t);`13672 `int tcdrain(int);`13673 `int tcflow(int, int);`13674 `int tcflush(int, int);`13675 `int tcgetattr(int, struct termios *);`13676 `pid_t tcgetsid(int);`13677 `int tcsendbreak(int, int);`13678 `int tcsetattr(int, int, const struct termios *);`13679 **APPLICATION USAGE**13680 The following names are reserved for XSI-conformant systems to use as an extension to the
13681 above; therefore strictly conforming applications shall not use them:

13682 CBAUD	EXTB	VDSUSP
13683 DEFCHO	FLUSHO	VLNEXT
13684 ECHOCTL	LOBLK	VREPRINT
13685 ECHOK	PENDIN	VSTATUS
13686 ECHOPRT	SWTCH	VWERASE
13687 EXTA	VDISCARD	

13688 **RATIONALE**

13689 None.

13690 **FUTURE DIRECTIONS**

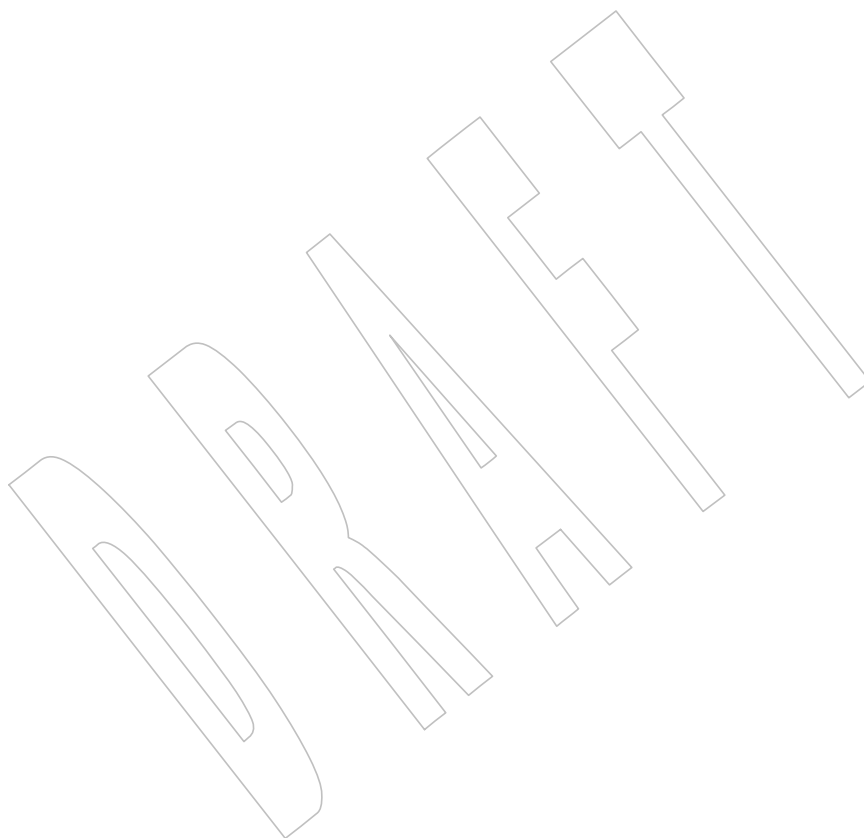
13691 None.

13692 **SEE ALSO**13693 XSH *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *confstr()*, *tcdrain()*, *tcflow()*, *tcflush()*, |
13694 *tcgetattr()*, *tcgetsid()*, *tcsendbreak()*, *tcsetattr()* |13695 XCU Chapter 11 (on page 185), *getconf*13696 **CHANGE HISTORY**

13697 First released in Issue 3.

13698 Included for alignment with the ISO POSIX-1 standard.

- 13699 **Issue 6**
- 13700 The LEGACY symbols IUCLC, OLCUC, and XCASE are removed.
- 13701 FIPS 151-2 requirements for the symbols CS7, CS8, CSTOPB, PARODD, and PARENB are
- 13702 reaffirmed.
- 13703 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/19 is applied, changing ECHOK to
- 13704 ECHOK in the APPLICATION USAGE section.
- 13705 **Issue 7**
- 13706 SD5-XBD-ERN-35 is applied, adding the OFDEL output mode.
- 13707 This reference page is clarified with respect to macros and symbolic constants. +



13708 **NAME**
 13709 `tgmath.h` — type-generic macros

13710 **SYNOPSIS**
 13711 `#include <tgmath.h>`

13712 **DESCRIPTION**

13713 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 13714 conflict between the requirements described here and the ISO C standard is unintentional. This
 13715 volume of POSIX.1-200x defers to the ISO C standard.

13716 The <tgmath.h> header shall include the headers <math.h> and <complex.h> and shall define
 13717 several type-generic macros.

13718 Of the functions contained within the <math.h> and <complex.h> headers without an *f* (**float**)
 13719 or *l* (**long double**) suffix, several have one or more parameters whose corresponding real type is
 13720 **double**. For each such function, except *modf()*, there shall be a corresponding type-generic
 13721 macro. The parameters whose corresponding real type is **double** in the function synopsis are
 13722 generic parameters. Use of the macro invokes a function whose corresponding real type and
 13723 type domain are determined by the arguments for the generic parameters.

13724 Use of the macro invokes a function whose generic parameters have the corresponding real type
 13725 determined as follows:

- 13726 • First, if any argument for generic parameters has type **long double**, the type determined is
 13727 **long double**.
- 13728 • Otherwise, if any argument for generic parameters has type **double** or is of integer type,
 13729 the type determined is **double**.
- 13730 • Otherwise, the type determined is **float**.

13731 For each unsuffixed function in the <math.h> header for which there is a function in the
 13732 <complex.h> header with the same name except for a *c* prefix, the corresponding type-generic
 13733 macro (for both functions) has the same name as the function in the <math.h> header. The
 13734 corresponding type-generic macro for *fabs()* and *cabs()* is *fabs()*.

13735	<math.h> Function	<complex.h> Function	Type-Generic Macro
13736	<i>acos()</i>	<i>cacos()</i>	<i>acos()</i>
13737	<i>asin()</i>	<i>casin()</i>	<i>asin()</i>
13738	<i>atan()</i>	<i>catan()</i>	<i>atan()</i>
13739	<i>acosh()</i>	<i>cacosh()</i>	<i>acosh()</i>
13740	<i>asinh()</i>	<i>casinh()</i>	<i>asinh()</i>
13741	<i>atanh()</i>	<i>catanh()</i>	<i>atanh()</i>
13742	<i>cos()</i>	<i>ccos()</i>	<i>cos()</i>
13743	<i>sin()</i>	<i>csin()</i>	<i>sin()</i>
13744	<i>tan()</i>	<i>ctan()</i>	<i>tan()</i>
13745	<i>cosh()</i>	<i>ccosh()</i>	<i>cosh()</i>
13746	<i>sinh()</i>	<i>csinh()</i>	<i>sinh()</i>
13747	<i>tanh()</i>	<i>ctanh()</i>	<i>tanh()</i>
13748	<i>exp()</i>	<i>cexp()</i>	<i>exp()</i>
13749	<i>log()</i>	<i>clog()</i>	<i>log()</i>
13750	<i>pow()</i>	<i>cpow()</i>	<i>pow()</i>
13751	<i>sqrt()</i>	<i>csqrt()</i>	<i>sqrt()</i>
13752	<i>fabs()</i>	<i>cabs()</i>	<i>fabs()</i>
13753			

13754 If at least one argument for a generic parameter is complex, then use of the macro invokes a
13755 complex function; otherwise, use of the macro invokes a real function.

13756 For each unsuffixed function in the <math.h> header without a *c*-prefixed counterpart in the
13757 <complex.h> header, the corresponding type-generic macro has the same name as the function.
13758 These type-generic macros are:

13759	<i>atan2()</i>	<i>fma()</i>	<i>llround()</i>	<i>remainder()</i>
13760	<i>cbrt()</i>	<i>fmax()</i>	<i>log10()</i>	<i>remquo()</i>
13761	<i>ceil()</i>	<i>fmin()</i>	<i>log1p()</i>	<i>rint()</i>
13762	<i>copysign()</i>	<i>fmod()</i>	<i>log2()</i>	<i>round()</i>
13763	<i>erf()</i>	<i>frexp()</i>	<i>logb()</i>	<i>scalbn()</i>
13764	<i>erfc()</i>	<i>hypot()</i>	<i>lrint()</i>	<i>scalbln()</i>
13765	<i>exp2()</i>	<i>ilogb()</i>	<i>lround()</i>	<i>tgamma()</i>
13766	<i>expm1()</i>	<i>ldexp()</i>	<i>nearbyint()</i>	<i>trunc()</i>
13767	<i>fdim()</i>	<i>lgamma()</i>	<i>nextafter()</i>	
13768	<i>floor()</i>	<i>llrint()</i>	<i>nexttoward()</i>	

13769 If all arguments for generic parameters are real, then use of the macro invokes a real function;
13770 otherwise, use of the macro results in undefined behavior.

13771 For each unsuffixed function in the <complex.h> header that is not a *c*-prefixed counterpart to a
13772 function in the <math.h> header, the corresponding type-generic macro has the same name as
13773 the function. These type-generic macros are:

13774	<i>carg()</i>
13775	<i>cimag()</i>
13776	<i>conj()</i>
13777	<i>cproj()</i>
13778	<i>creal()</i>

13779 Use of the macro with any real or complex argument invokes a complex function.

13780 APPLICATION USAGE

13781 With the declarations:

```
13782 #include <tgmath.h>
13783 int n;
13784 float f;
13785 double d;
13786 long double ld;
13787 float complex fc;
13788 double complex dc;
13789 long double complex ldc;
```

13790 functions invoked by use of type-generic macros are shown in the following table:

13791 Macro	Use Invokes
13792 <i>exp(n)</i>	<i>exp(n)</i> , the function
13793 <i>acosh(f)</i>	<i>acoshf(f)</i>
13794 <i>sin(d)</i>	<i>sin(d)</i> , the function
13795 <i>atan(ld)</i>	<i>atanl(ld)</i>
13796 <i>log(fc)</i>	<i>clogf(fc)</i>
13797 <i>sqrt(dc)</i>	<i>csqrt(dc)</i>
13798 <i>pow(ldc,f)</i>	<i>cpowl(ldc, f)</i>
13799 <i>remainder(n,n)</i>	<i>remainder(n, n)</i> , the function
13800 <i>nextafter(d,f)</i>	<i>nextafter(d, f)</i> , the function

13801
13802
13803
13804
13805
13806
13807
13808
13809
13810
13811
13812
13813

Macro	Use Invokes
<i>nexttoward(f,ld)</i>	<i>nexttowardf(f, ld)</i>
<i>copysign(n,ld)</i>	<i>copysignl(n, ld)</i>
<i>ceil(fc)</i>	Undefined behavior
<i>rint(dc)</i>	Undefined behavior
<i>fmax(ldc,ld)</i>	Undefined behavior
<i>carg(n)</i>	<i>carg(n)</i> , the function
<i>cproj(f)</i>	<i>cprojf(f)</i>
<i>creal(d)</i>	<i>creal(d)</i> , the function
<i>cimag(ld)</i>	<i>cimagl(ld)</i>
<i>cabs(fc)</i>	<i>cabsf(fc)</i>
<i>carg(dc)</i>	<i>carg(dc)</i> , the function
<i>cproj ldc)</i>	<i>cprojl(ldc)</i>

13814

RATIONALE13815
13816
13817
13818
13819

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as '+' and '*'. For example, with a type-generic *cos()* macro, the expression *cos((float)x)* will have type **float**. This feature enables writing more portably efficient code and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. Generic math functions are a widely appreciated feature of Fortran.

13820
13821
13822
13823

The only arguments that affect the type resolution are the arguments corresponding to the parameters that have type **double** in the synopsis. Hence the type of a type-generic call to *nexttoward()*, whose second parameter is **long double** in the synopsis, is determined solely by the type of the first argument.

13824
13825
13826

The term "type-generic" was chosen over the proposed alternatives of intrinsic and overloading. The term is more specific than intrinsic, which already is widely used with a more general meaning, and reflects a closer match to Fortran's generic functions than to C++ overloading.

13827
13828

The macros are placed in their own header in order not to silently break old programs that include the <math.h> header; for example, with:

13829

```
printf ("%e", sin(x))
```

13830
13831

*modf(double, double *)* is excluded because no way was seen to make it safe without complicating the type resolution.

13832
13833

The implementation might, as an extension, endow appropriate ones of the macros that POSIX.1-200x specifies only for real arguments with the ability to invoke the complex functions.

13834
13835
13836

POSIX.1-200x does not prescribe any particular implementation mechanism for generic macros. It could be implemented simply with built-in macros. The generic macro for *sqrt()*, for example, could be implemented with:

13837

```
#undef sqrt
```

13838

```
#define sqrt(x) __BUILTIN_GENERIC_sqrt(x)
```

13839
13840

Generic macros are designed for a useful level of consistency with C++ overloaded math functions.

13841
13842
13843
13844

The great majority of existing C programs are expected to be unaffected when the <tgmath.h> header is included instead of the <math.h> or <complex.h> headers. Generic macros are similar to the ISO/IEC 9899:1999 standard library masking macros, though the semantic types of return values differ.

13845
13846
13847
13848

The ability to overload on integer as well as floating types would have been useful for some functions; for example, *copysign()*. Overloading with different numbers of arguments would have allowed reusing names; for example, *remainder()* for *remquo()*. However, these facilities would have complicated the specification; and their natural consistent use, such as for a floating

13849 *abs()* or a two-argument *atan()*, would have introduced further inconsistencies with the
13850 ISO/IEC 9899:1999 standard for insufficient benefit.

13851 The ISO C standard in no way limits the implementation's options for efficiency, including
13852 inlining library functions.

13853 **FUTURE DIRECTIONS**

13854 None.

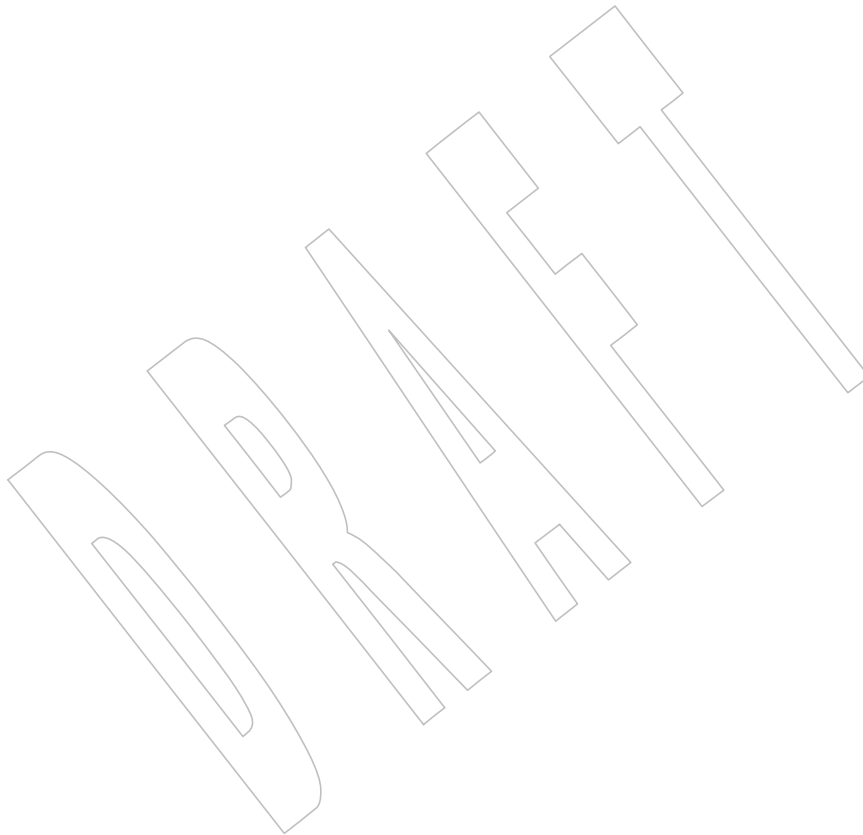
13855 **SEE ALSO**

13856 [<math.h>](#), [<complex.h>](#)

13857 XSH *cabs()*, *fabs()*, *modf()*

13858 **CHANGE HISTORY**

13859 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.



13860 **NAME**

13861 time.h — time types

13862 **SYNOPSIS**

13863 #include <time.h>

13864 **DESCRIPTION**

13865 CX Some of the functionality described on this reference page extends the ISO C standard.
 13866 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 13867 enable the visibility of these symbols in this header.

13868 The <time.h> header shall declare the structure **tm**, which shall include at least the following
 13869 members:

13870 int tm_sec Seconds [0,60].
 13871 int tm_min Minutes [0,59].
 13872 int tm_hour Hour [0,23].
 13873 int tm_mday Day of month [1,31].
 13874 int tm_mon Month of year [0,11].
 13875 int tm_year Years since 1900.
 13876 int tm_wday Day of week [0,6] (Sunday =0).
 13877 int tm_yday Day of year [0,365].
 13878 int tm_isdst Daylight Savings flag.

13879 The value of *tm_isdst* shall be positive if Daylight Savings Time is in effect, 0 if Daylight Savings
 13880 Time is not in effect, and negative if the information is not available.

13881 The <time.h> header shall define the following macros:

13882 NULL As described in <stddef.h>.

13883 CLOCKS_PER_SEC A number used to convert the value returned by the *clock()* function into
 13884 XSI seconds. The value shall be an expression with type **clock_t**. The value of
 13885 CLOCKS_PER_SEC shall be 1 million on XSI-conformant systems.
 13886 However, it may be variable on other systems, and it should not be
 13887 assumed that CLOCKS_PER_SEC is a compile-time constant.

13888 CX The <time.h> header shall define the following symbolic constants. The values shall have a type
 13889 that is assignment-compatible with **clockid_t**.

13890 MON **CLOCK_MONOTONIC**

13891 The identifier for the system-wide monotonic clock, which is defined as a
 13892 realtime clock whose value cannot be set via *clock_settime()* and which
 13893 cannot have backward clock jumps. The maximum possible clock jump
 13894 shall be implementation-defined.

13895 CPT **CLOCK_PROCESS_CPUTIME_ID**

13896 The identifier of the CPU-time clock associated with the process making a
 13897 *clock()* or *timer*()* function call.

13898 CX **CLOCK_REALTIME** The identifier of the system-wide realtime clock.

13899 TCT **CLOCK_THREAD_CPUTIME_ID**

13900 The identifier of the CPU-time clock associated with the thread making a
 13901 *clock()* or *timer*()* function call.

<time.h>

Headers

13902	CX	The <time.h> header shall declare the structure timespec , which has at least the following members:
13903		
13904		time_t tv_sec Seconds.
13905		long tv_nsec Nanoseconds.
13906		The <time.h> header shall also declare the itimerspec structure, which has at least the following members:
13907		
13908		struct timespec it_interval Timer period.
13909		struct timespec it_value Timer expiration.
13910		The <time.h> header shall define the following symbolic constant:
13911		TIMER_ABSTIME Flag indicating time is absolute. For functions taking timer objects, this
13912		refers to the clock associated with the timer.
13913	CX	The clock_t , size_t , time_t , clockid_t , and timer_t types shall be defined as described in <sys/types.h> .
13914		
13915	XSI	The <time.h> header shall provide a declaration for <i>getdate_err</i> .
13916		The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.
13917		
13918	OB	char *asctime(const struct tm *);
13919	OB CX	char *asctime_r(const struct tm *restrict, char *restrict);
13920		clock_t clock(void);
13921	CPT	int clock_getcpuclockid(pid_t, clockid_t *);
13922	CX	int clock_getres(clockid_t, struct timespec *);
13923		int clock_gettime(clockid_t, struct timespec *);
13924		int clock_nanosleep(clockid_t, int, const struct timespec *,
13925		struct timespec *);
13926		int clock_settime(clockid_t, const struct timespec *);
13927	OB	char *ctime(const time_t *);
13928	OB CX	char *ctime_r(const time_t *, char *);
13929		double difftime(time_t, time_t);
13930	XSI	struct tm *getdate(const char *);
13931		struct tm *gmtime(const time_t *);
13932	CX	struct tm *gmtime_r(const time_t *restrict, struct tm *restrict);
13933		struct tm *localtime(const time_t *);
13934	CX	struct tm *localtime_r(const time_t *restrict, struct tm *restrict);
13935		time_t mktime(struct tm *);
13936	CX	int nanosleep(const struct timespec *, struct timespec *);
13937		size_t strftime(char *restrict, size_t, const char *restrict,
13938		const struct tm *restrict);
13939	CX	size_t strftime_l(char *restrict, size_t, const char *restrict,
13940		const struct tm *restrict, locale_t);
13941	XSI	char *strptime(const char *restrict, const char *restrict,
13942		struct tm *restrict);
13943		time_t time(time_t *);
13944	CX	int timer_create(clockid_t, struct sigevent *restrict,
13945		timer_t *restrict);
13946		int timer_delete(timer_t);
13947		int timer_getoverrun(timer_t);
13948		int timer_gettime(timer_t, struct itimerspec *);
13949		int timer_settime(timer_t, int, const struct itimerspec *restrict,
13950		struct itimerspec *restrict);

13951 `void tzset(void);`

13952 The following shall be declared as variables:

13953 XSI `extern int daylight;`
 13954 `extern long timezone;`
 13955 CX `extern char *tzname[];`

13956 CX Inclusion of the <time.h> header may make visible all symbols from the <signal.h> header.

13957 APPLICATION USAGE

13958 The range [0,60] for *tm_sec* allows for the occasional leap second.

13959 *tm_year* is a signed value; therefore, years before 1900 may be represented.

13960 To obtain the number of clock ticks per second returned by the *times()* function, applications
 13961 should call *sysconf(_SC_CLK_TCK)*.

13962 RATIONALE

13963 The range [0,60] seconds allows for positive or negative leap seconds. The formal definition of
 13964 UTC does not permit double leap seconds, so all mention of double leap seconds has been
 13965 removed, and the range shortened from the former [0,61] seconds seen in earlier versions of this
 13966 standard.

13967 FUTURE DIRECTIONS

13968 None.

13969 SEE ALSO

13970 <signal.h>, <stddef.h>, <sys/types.h>

13971 XSH Section 2.2 (on page 448), *asctime()*, *clock()*, *clock_getcpuclockid()*, *clock_getres()*,
 13972 *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *mq_receive()*,
 13973 *mq_send()*, *nanosleep()*, *pthread_getcpuclockid()*, *pthread_mutex_timedlock()*,
 13974 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *sem_timedwait()*, *strptime()*,
 13975 *sysconf()*, *time*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *tzset()*, *utime()*

13976 CHANGE HISTORY

13977 First released in Issue 1. Derived from Issue 1 of the SVID.

13978 Issue 5

13979 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 13980 Threads Extension.

13981 Issue 6

13982 The Open Group Corrigendum U035/6 is applied. In the DESCRIPTION, the types **clockid_t**
 13983 and **timer_t** have been described.

13984 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 13985 • The POSIX timer-related functions are marked as part of the Timers option.

13986 The symbolic name CLK_TCK is removed. Application usage is added describing how its
 13987 equivalent functionality can be obtained using *sysconf()*.

13988 The *clock_getcpuclockid()* function and manifest constants CLOCK_PROCESS_CPUTIME_ID and
 13989 CLOCK_THREAD_CPUTIME_ID are added for alignment with IEEE Std 1003.1d-1999.

13990 The manifest constant CLOCK_MONOTONIC and the *clock_nanosleep()* function are added for
 13991 alignment with IEEE Std 1003.1j-2000.

13992 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

13993

- The range for seconds is changed from [0,61] to [0,60].

13994

13995

- The **restrict** keyword is added to the prototypes for *asctime_r()*, *gmtime_r()*, *localtime_r()*, *strftime()*, *strptime()*, *timer_create()*, and *timer_settime()*.

13996

13997

IEEE PASC Interpretation 1003.1 #84 is applied adding the statement that symbols from the <signal.h> header may be made visible when the <time.h> header is included.

13998

Extensions beyond the ISO C standard are marked.

13999

Issue 7

14000

Austin Group Interpretation 1003.1-2001 #111 is applied. +

14001

SD5-XBD-ERN-74 is applied. +

14002

14003

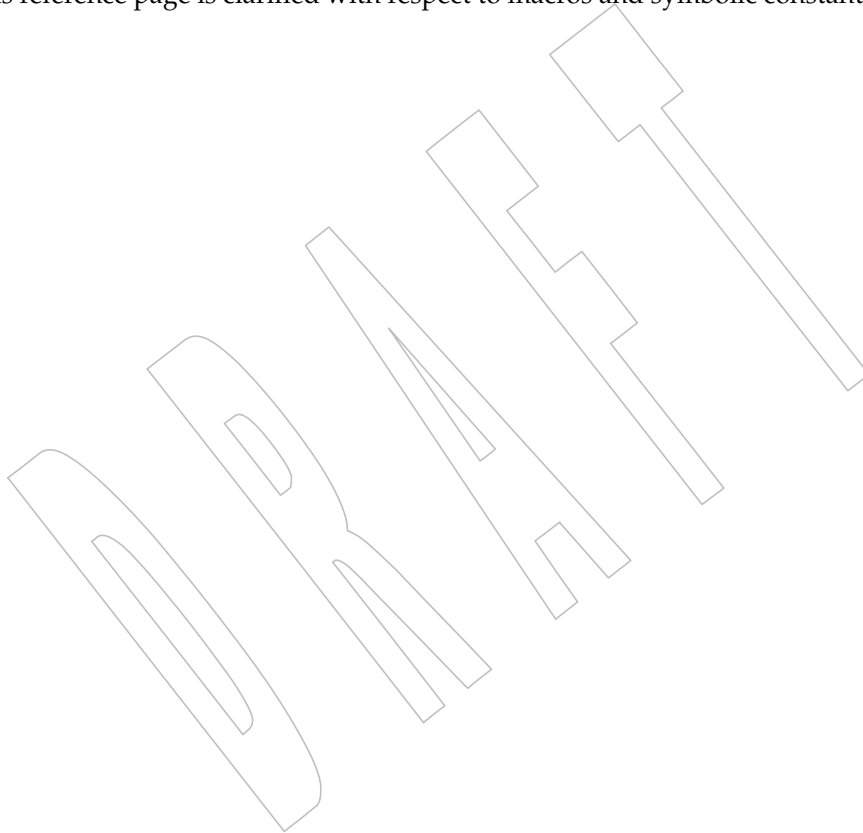
The *strftime_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

14004

Functionality relating to the Timers option is moved to the Base.

14005

This reference page is clarified with respect to macros and symbolic constants. |



14006 **NAME**

14007 trace.h — tracing

14008 **SYNOPSIS**

14009 OB TRC #include <trace.h>

14010 **DESCRIPTION**14011 The <trace.h> header shall define the **posix_trace_event_info** structure that includes at least the
14012 following members:

```

14013     trace_event_id_t  posix_event_id
14014     pid_t             posix_pid
14015     void              *posix_prog_address
14016     pthread_t         posix_thread_id
14017     struct timespec   posix_timestamp
14018     int               posix_truncation_status

```

14019 The <trace.h> header shall define the **posix_trace_status_info** structure that includes at least the
14020 following members:

```

14021     int      posix_stream_full_status
14022     int      posix_stream_overrun_status
14023     int      posix_stream_status
14024     OB TRL  int      posix_log_full_status
14025     int      posix_log_overrun_status
14026     int      posix_stream_flush_error
14027     int      posix_stream_flush_status

```

14028 The <trace.h> header shall define the following symbolic constants: |

```

14029     POSIX_TRACE_ALL_EVENTS
14030     OB TRL  POSIX_TRACE_APPEND
14031     OB TRI  POSIX_TRACE_CLOSE_FOR_CHILD
14032     OB TEF  POSIX_TRACE_FILTER
14033     OB TRL  POSIX_TRACE_FLUSH
14034     POSIX_TRACE_FLUSH_START
14035     POSIX_TRACE_FLUSH_STOP
14036     POSIX_TRACE_FLUSHING
14037     POSIX_TRACE_FULL
14038     POSIX_TRACE_LOOP
14039     POSIX_TRACE_NO_OVERRUN
14040     OB TRL  POSIX_TRACE_NOT_FLUSHING
14041     POSIX_TRACE_NOT_FULL
14042     OB TRI  POSIX_TRACE_INHERITED
14043     POSIX_TRACE_NOT_TRUNCATED
14044     POSIX_TRACE_OVERFLOW
14045     POSIX_TRACE_OVERRUN
14046     POSIX_TRACE_RESUME
14047     POSIX_TRACE_RUNNING
14048     POSIX_TRACE_START
14049     POSIX_TRACE_STOP
14050     POSIX_TRACE_SUSPENDED
14051     POSIX_TRACE_SYSTEM_EVENTS

```

```

14052     POSIX_TRACE_TRUNCATED_READ
14053     POSIX_TRACE_TRUNCATED_RECORD
14054     POSIX_TRACE_UNNAMED_USER_EVENT
14055     POSIX_TRACE_UNTIL_FULL
14056     POSIX_TRACE_WOPID_EVENTS

```

14057 The following types shall be defined as described in <sys/types.h>:

```

14058     size_t
14059     trace_attr_t
14060     trace_event_id_t
14061     OB TEF trace_event_set_t
14062     trace_id_t

```

14063 The following shall be declared as functions and may also be defined as macros. Function
14064 prototypes shall be provided.

```

14065     int  posix_trace_attr_destroy(trace_attr_t *);
14066     int  posix_trace_attr_getclockres(const trace_attr_t *,
14067         struct timespec *);
14068     int  posix_trace_attr_getcreatetime(const trace_attr_t *,
14069         struct timespec *);
14070     int  posix_trace_attr_getgenversion(const trace_attr_t *, char *);
14071     TRI  int  posix_trace_attr_getinherited(const trace_attr_t *restrict,
14072         int *restrict);
14073     TRL  int  posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict,
14074         int *restrict);
14075     int  posix_trace_attr_getlogsize(const trace_attr_t *restrict,
14076         size_t *restrict);
14077     int  posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict,
14078         size_t *restrict);
14079     int  posix_trace_attr_getmaxsystemeventsize(const trace_attr_t *restrict,
14080         size_t *restrict);
14081     int  posix_trace_attr_getmaxusereventsize(const trace_attr_t *restrict,
14082         size_t, size_t *restrict);
14083     int  posix_trace_attr_getname(const trace_attr_t *, char *);
14084     int  posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict,
14085         int *restrict);
14086     int  posix_trace_attr_getstreamsize(const trace_attr_t *restrict,
14087         size_t *restrict);
14088     int  posix_trace_attr_init(trace_attr_t *);
14089     TRI  int  posix_trace_attr_setinherited(trace_attr_t *, int);
14090     TRL  int  posix_trace_attr_setlogfullpolicy(trace_attr_t *, int);
14091     int  posix_trace_attr_setlogsize(trace_attr_t *, size_t);
14092     int  posix_trace_attr_setmaxdatasize(trace_attr_t *, size_t);
14093     int  posix_trace_attr_setname(trace_attr_t *, const char *);
14094     int  posix_trace_attr_setstreamfullpolicy(trace_attr_t *, int);
14095     int  posix_trace_attr_setstreamsize(trace_attr_t *, size_t);
14096     int  posix_trace_clear(trace_id_t);
14097     TRL  int  posix_trace_close(trace_id_t);
14098     int  posix_trace_create(pid_t, const trace_attr_t *restrict,
14099         trace_id_t *restrict);
14100     TRL  int  posix_trace_create_withlog(pid_t, const trace_attr_t *restrict,
14101         int, trace_id_t *restrict);
14102     void posix_trace_event(trace_event_id_t, const void *restrict, size_t);
14103     int  posix_trace_eventid_equal(trace_id_t, trace_event_id_t,

```



```

14104         trace_event_id_t);
14105     int  posix_trace_eventid_get_name(trace_id_t, trace_event_id_t, char *);
14106     int  posix_trace_eventid_open(const char *restrict,
14107         trace_event_id_t *restrict);
14108     TEF  int  posix_trace_eventset_add(trace_event_id_t, trace_event_set_t *);
14109         int  posix_trace_eventset_del(trace_event_id_t, trace_event_set_t *);
14110         int  posix_trace_eventset_empty(trace_event_set_t *);
14111         int  posix_trace_eventset_fill(trace_event_set_t *, int);
14112         int  posix_trace_eventset_ismember(trace_event_id_t,
14113         const trace_event_set_t *restrict, int *restrict);
14114         int  posix_trace_eventtypelist_getnext_id(trace_id_t,
14115         trace_event_id_t *restrict, int *restrict);
14116         int  posix_trace_eventtypelist_rewind(trace_id_t);
14117     TRL  int  posix_trace_flush(trace_id_t);
14118         int  posix_trace_get_attr(trace_id_t, trace_attr_t *);
14119     TEF  int  posix_trace_get_filter(trace_id_t, trace_event_set_t *);
14120         int  posix_trace_get_status(trace_id_t,
14121         struct posix_trace_status_info *);
14122         int  posix_trace_getnext_event(trace_id_t,
14123         struct posix_trace_event_info *restrict, void *restrict,
14124         size_t, size_t *restrict, int *restrict);
14125     TRL  int  posix_trace_open(int, trace_id_t *);
14126         int  posix_trace_rewind(trace_id_t);
14127     TEF  int  posix_trace_set_filter(trace_id_t, const trace_event_set_t *, int);
14128         int  posix_trace_shutdown(trace_id_t);
14129         int  posix_trace_start(trace_id_t);
14130         int  posix_trace_stop(trace_id_t);
14131         int  posix_trace_timedgetnext_event(trace_id_t,
14132         struct posix_trace_event_info *restrict, void *restrict,
14133         size_t, size_t *restrict, int *restrict,
14134         const struct timespec *restrict);
14135     TEF  int  posix_trace_trid_eventid_open(trace_id_t, const char *restrict,
14136         trace_event_id_t *restrict);
14137         int  posix_trace_trygetnext_event(trace_id_t,
14138         struct posix_trace_event_info *restrict, void *restrict, size_t,
14139         size_t *restrict, int *restrict);

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The <trace.h> header may be removed in a future version.

SEE ALSO

<sys/types.h>

XSH Section 2.11 (on page 507), *posix_trace_attr_destroy()*, *posix_trace_attr_getclockres()*, *posix_trace_attr_getinherited()*, *posix_trace_attr_getlogsize()*, *posix_trace_clear()*, *posix_trace_close()*, *posix_trace_create()*, *posix_trace_event()*, *posix_trace_eventid_equal()*, *posix_trace_eventtypelist_getnext_id()*, *posix_trace_eventset_add()*, *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_getnext_event()*, *posix_trace_start()*

14153

CHANGE HISTORY

14154

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

14155

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/40 is applied, adding the TRL margin code to the *posix_trace_flush()* function, for alignment with the System Interfaces volume of POSIX.1-200x.

14156

14157

14158

Issue 7

14159

SD5-XBD-ERN-56 is applied, adding a reference to **<sys/types.h>** for the **size_t** type.

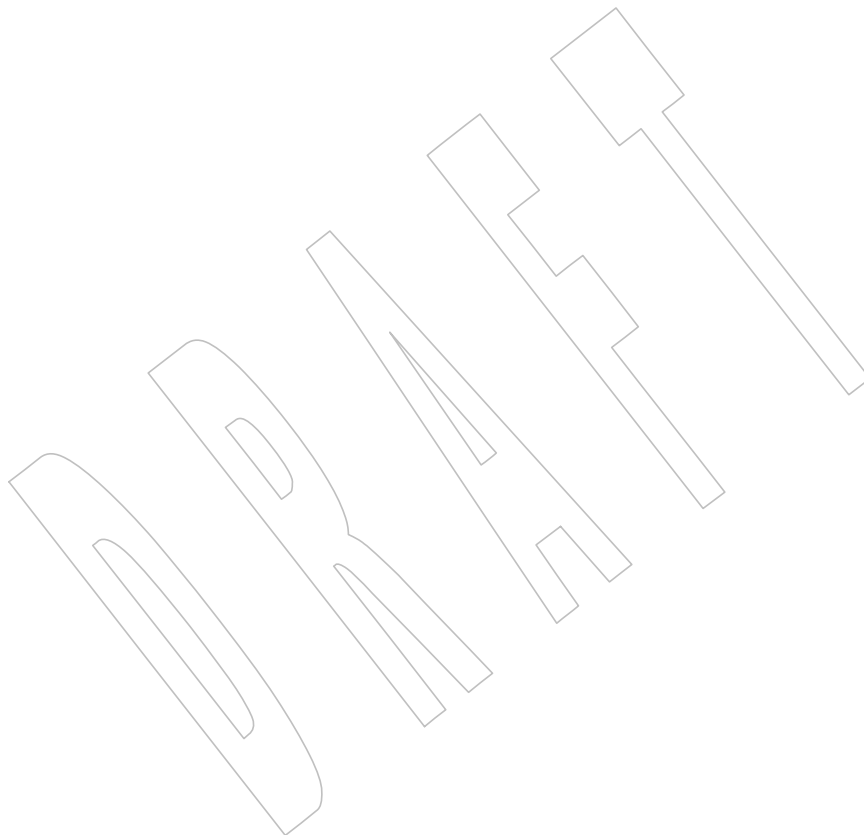
14160

The **<trace.h>** header is marked obsolescent.

14161

This reference page is clarified with respect to macros and symbolic constants.

+



14162 **NAME**14163 `ulimit.h` — ulimit commands14164 **SYNOPSIS**14165 OB XSI `#include <ulimit.h>`14166 **DESCRIPTION**14167 The <ulimit.h> header shall define the symbolic constants used by the *ulimit()* function.

14168 Symbolic constants:

14169 `UL_GETFSIZE` Get maximum file size.14170 `UL_SETFSIZE` Set maximum file size.14171 The following shall be declared as a function and may also be defined as a macro. A function
14172 prototype shall be provided.14173 `long ulimit(int, ...);`14174 **APPLICATION USAGE**14175 See *ulimit()*.14176 **RATIONALE**

14177 None.

14178 **FUTURE DIRECTIONS**

14179 None.

14180 **SEE ALSO**14181 XSH *ulimit*14182 **CHANGE HISTORY**

14183 First released in Issue 3.

14184 **Issue 7**

14185 The <ulimit.h> header is marked obsolescent.

14186 **NAME**
 14187 unistd.h — standard symbolic constants and types

14188 **SYNOPSIS**
 14189 #include <unistd.h>

14190 **DESCRIPTION**
 14191 The <unistd.h> header defines miscellaneous symbolic constants and types, and declares
 14192 miscellaneous functions. The actual values of the constants are unspecified except as shown. The
 14193 contents of this header are shown below.

14194 **Version Test Macros**

14195 The <unistd.h> header shall define the following symbolic constants. The values shall be
 14196 suitable for use in #if preprocessing directives.

14197 **_POSIX_VERSION**
 14198 Integer value indicating version of this standard (C-language binding) to which the
 14199 implementation conforms. For implementations conforming to POSIX.1-200x, the value
 14200 shall be 200xxxL.

14201 **_POSIX2_VERSION**
 14202 Integer value indicating version of the Shell and Utilities volume of POSIX.1 to which the
 14203 implementation conforms. For implementations conforming to POSIX.1-200x, the value
 14204 shall be 200xxxL.

14205 The <unistd.h> header shall define the following symbolic constant only if the implementation
 14206 supports the XSI option; see [Section 2.1.4](#) (on page 19). If defined, its value shall be suitable for
 14207 use in #if preprocessing directives.

14208 XSI **_XOPEN_VERSION**
 14209 Integer value indicating version of the X/Open Portability Guide to which the
 14210 implementation conforms. The value shall be 700.

14211 **Constants for Options and Option Groups**

14212 The following symbolic constants, if defined in <unistd.h>, shall have a value of -1, 0, or
 14213 greater, unless otherwise specified below. The values shall be suitable for use in #if
 14214 preprocessing directives.

14215 If a symbolic constant is not defined or is defined with the value -1, the option is not supported
 14216 for compilation. If it is defined with a value greater than zero, the option shall always be
 14217 supported when the application is executed. If it is defined with the value zero, the option shall
 14218 be supported for compilation and might or might not be supported at runtime. See [Section 2.1.6](#)
 14219 (on page 25) for further information about the conformance requirements of these three
 14220 categories of support.

14221 ADV **_POSIX_ADVISORY_INFO**
 14222 The implementation supports the Advisory Information option. If this symbol is defined in
 14223 <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
 14224 *sysconf()* shall either be -1 or 200xxxL.

14225 **_POSIX_ASYNCHRONOUS_IO**
 14226 The implementation supports asynchronous input and output. This symbol shall always be
 14227 set to the value 200xxxL.

14228		<code>_POSIX_BARRIERS</code>	
14229			The implementation supports barriers. This symbol shall always be set to the value
14230			200xxxL.
14231		<code>_POSIX_CHOWN_RESTRICTED</code>	
14232			The use of <i>chown()</i> and <i>fchown()</i> is restricted to a process with appropriate privileges, and
14233			to changing the group ID of a file only to the effective group ID of the process or to one of
14234			its supplementary group IDs. This symbol shall be defined with a value other than -1.
14235		<code>_POSIX_CLOCK_SELECTION</code>	
14236			The implementation supports clock selection. This symbol shall always be set to the value
14237			200xxxL.
14238	CPT	<code>_POSIX_CPUTIME</code>	
14239			The implementation supports the Process CPU-Time Clocks option. If this symbol is
14240			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14241			reported by <i>sysconf()</i> shall either be -1 or 200xxxL.
14242	FSC	<code>_POSIX_FSYNC</code>	
14243			The implementation supports the File Synchronization option. If this symbol is defined in
14244			<unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
14245			<i>sysconf()</i> shall either be -1 or 200xxxL.
14246		<code>_POSIX_IPV6</code>	
14247			The implementation supports the IPv6 option. If this symbol is defined in <unistd.h>, it
14248			shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by <i>sysconf()</i> shall
14249			either be -1 or 200xxxL.
14250		<code>_POSIX_JOB_CONTROL</code>	
14251			The implementation supports job control. This symbol shall always be set to a value greater
14252			than zero.
14253		<code>_POSIX_MAPPED_FILES</code>	
14254			The implementation supports memory mapped Files. This symbol shall always be set to the
14255			value 200xxxL.
14256	ML	<code>_POSIX_MEMLOCK</code>	
14257			The implementation supports the Process Memory Locking option. If this symbol is defined
14258			in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14259			by <i>sysconf()</i> shall either be -1 or 200xxxL.
14260	MLR	<code>_POSIX_MEMLOCK_RANGE</code>	
14261			The implementation supports the Range Memory Locking option. If this symbol is defined
14262			in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14263			by <i>sysconf()</i> shall either be -1 or 200xxxL.
14264		<code>_POSIX_MEMORY_PROTECTION</code>	
14265			The implementation supports memory protection. This symbol shall always be set to the
14266			value 200xxxL.
14267	MSG	<code>_POSIX_MESSAGE_PASSING</code>	
14268			The implementation supports the Message Passing option. If this symbol is defined in
14269			<unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
14270			<i>sysconf()</i> shall either be -1 or 200xxxL.
14271	MON	<code>_POSIX_MONOTONIC_CLOCK</code>	
14272			The implementation supports the Monotonic Clock option. If this symbol is defined in
14273			<unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
14274			<i>sysconf()</i> shall either be -1 or 200xxxL.

14275		<code>_POSIX_NO_TRUNC</code>	
14276			Pathname components longer than <code>{NAME_MAX}</code> generate an error. This symbol shall be
14277			defined with a value other than <code>-1</code> .
14278	PIO	<code>_POSIX_PRIORITIZED_IO</code>	
14279			The implementation supports the Prioritized Input and Output option. If this symbol is
14280			defined in <code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol
14281			reported by <code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14282	PS	<code>_POSIX_PRIORITY_SCHEDULING</code>	
14283			The implementation supports the Process Scheduling option. If this symbol is defined in
14284			<code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by
14285			<code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14286	RS	<code>_POSIX_RAW_SOCKETS</code>	
14287			The implementation supports the Raw Sockets option. If this symbol is defined in
14288			<code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by
14289			<code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14290		<code>_POSIX_READER_WRITER_LOCKS</code>	
14291			The implementation supports read-write locks. This symbol shall always be set to the value
14292			<code>200xxxL</code> .
14293		<code>_POSIX_REALTIME_SIGNALS</code>	
14294			The implementation supports realtime signals. This symbol shall always be set to the value
14295			<code>200xxxL</code> .
14296		<code>_POSIX_REGEX</code>	
14297			The implementation supports the Regular Expression Handling option. This symbol shall
14298			always be set to a value greater than zero.
14299		<code>_POSIX_SAVED_IDS</code>	
14300			Each process has a saved set-user-ID and a saved set-group-ID. This symbol shall always be
14301			set to a value greater than zero.
14302		<code>_POSIX_SEMAPHORES</code>	
14303			The implementation supports semaphores. This symbol shall always be set to the value
14304			<code>200xxxL</code> .
14305	SHM	<code>_POSIX_SHARED_MEMORY_OBJECTS</code>	
14306			The implementation supports the Shared Memory Objects option. If this symbol is defined
14307			in <code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported
14308			by <code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14309		<code>_POSIX_SHELL</code>	
14310			The implementation supports the POSIX shell. This symbol shall always be set to a value
14311			greater than zero.
14312	SPN	<code>_POSIX_SPAWN</code>	
14313			The implementation supports the Spawn option. If this symbol is defined in <code><unistd.h></code> , it
14314			shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by <code>sysconf()</code> shall
14315			either be <code>-1</code> or <code>200xxxL</code> .
14316		<code>_POSIX_SPIN_LOCKS</code>	
14317			The implementation supports spin locks. This symbol shall always be set to the value
14318			<code>200xxxL</code> .
14319	SS	<code>_POSIX_SPORADIC_SERVER</code>	
14320			The implementation supports the Process Sporadic Server option. If this symbol is defined
14321			in <code><unistd.h></code> , it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported
14322			by <code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .

14323	SIO	<u>POSIX_SYNCHRONIZED_IO</u>
14324		The implementation supports the Synchronized Input and Output option. If this symbol is
14325		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14326		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14327	TSA	<u>POSIX_THREAD_ATTR_STACKADDR</u>
14328		The implementation supports the Thread Stack Address Attribute option. If this symbol is
14329		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14330		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14331	TSS	<u>POSIX_THREAD_ATTR_STACKSIZE</u>
14332		The implementation supports the Thread Stack Size Attribute option. If this symbol is
14333		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14334		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14335	TCT	<u>POSIX_THREAD_CPU_TIME</u>
14336		The implementation supports the Thread CPU-Time Clocks option. If this symbol is
14337		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14338		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14339	TPI	<u>POSIX_THREAD_PRIO_INHERIT</u>
14340		The implementation supports the Non-Robust Mutex Priority Inheritance option. If this
14341		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this
14342		symbol reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14343	TPP	<u>POSIX_THREAD_PRIO_PROTECT</u>
14344		The implementation supports the Non-Robust Mutex Priority Protection option. If this
14345		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this
14346		symbol reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14347	TPS	<u>POSIX_THREAD_PRIORITY_SCHEDULING</u>
14348		The implementation supports the Thread Execution Scheduling option. If this symbol is
14349		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14350		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14351	TSH	<u>POSIX_THREAD_PROCESS_SHARED</u>
14352		The implementation supports the Thread Process-Shared Synchronization option. If this
14353		symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this
14354		symbol reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14355	RPI	<u>POSIX_THREAD_ROBUST_PRIO_INHERIT</u>
14356		The implementation supports the Robust Mutex Priority Inheritance option. If this symbol
14357		is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14358		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14359	RPP	<u>POSIX_THREAD_ROBUST_PRIO_PROTECT</u>
14360		The implementation supports the Robust Mutex Priority Protection option. If this symbol is
14361		defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14362		reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14363		<u>POSIX_THREAD_SAFE_FUNCTIONS</u>
14364		The implementation supports thread-safe functions. This symbol shall always be set to the
14365		value 200xxxL.
14366	TSP	<u>POSIX_THREAD_SPORADIC_SERVER</u>
14367		The implementation supports the Thread Sporadic Server option. If this symbol is defined
14368		in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14369		by <i>sysconf</i> () shall either be -1 or 200xxxL.

14370		<code>_POSIX_THREADS</code>
14371		The implementation supports threads. This symbol shall always be set to the value
14372		<code>200xxxL</code> .
14373		<code>_POSIX_TIMEOUTS</code>
14374		The implementation supports timeouts. This symbol shall always be set to the value
14375		<code>200xxxL</code> .
14376		<code>_POSIX_TIMERS</code>
14377		The implementation supports timers. This symbol shall always be set to the value <code>200xxxL</code> .
14378	OB TRC	<code>_POSIX_TRACE</code>
14379		The implementation supports the Trace option. If this symbol is defined in <unistd.h>, it
14380		shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by <code>sysconf()</code> shall
14381		either be <code>-1</code> or <code>200xxxL</code> .
14382	OB TEF	<code>_POSIX_TRACE_EVENT_FILTER</code>
14383		The implementation supports the Trace Event Filter option. If this symbol is defined in
14384		<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by
14385		<code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14386	OB TRI	<code>_POSIX_TRACE_INHERIT</code>
14387		The implementation supports the Trace Inherit option. If this symbol is defined in
14388		<unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by
14389		<code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14390	OB TRL	<code>_POSIX_TRACE_LOG</code>
14391		The implementation supports the Trace Log option. If this symbol is defined in <unistd.h>,
14392		it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported by <code>sysconf()</code>
14393		shall either be <code>-1</code> or <code>200xxxL</code> .
14394	TYM	<code>_POSIX_TYPED_MEMORY_OBJECTS</code>
14395		The implementation supports the Typed Memory Objects option. If this symbol is defined
14396		in <unistd.h>, it shall be defined to be <code>-1</code> , <code>0</code> , or <code>200xxxL</code> . The value of this symbol reported
14397		by <code>sysconf()</code> shall either be <code>-1</code> or <code>200xxxL</code> .
14398	OB	<code>_POSIX_V6_ILP32_OFF32</code>
14399		The implementation provides a C-language compilation environment with 32-bit int , long ,
14400		pointer , and off_t types.
14401	OB	<code>_POSIX_V6_ILP32_OFFBIG</code>
14402		The implementation provides a C-language compilation environment with 32-bit int , long ,
14403		and pointer types and an off_t type using at least 64 bits.
14404	OB	<code>_POSIX_V6_LP64_OFF64</code>
14405		The implementation provides a C-language compilation environment with 32-bit int and
14406		64-bit long , pointer , and off_t types.
14407	OB	<code>_POSIX_V6_LP64_OFFBIG</code>
14408		The implementation provides a C-language compilation environment with an int type
14409		using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14410		<code>_POSIX_V7_ILP32_OFF32</code>
14411		The implementation provides a C-language compilation environment with 32-bit int , long ,
14412		pointer , and off_t types.
14413		<code>_POSIX_V7_ILP32_OFFBIG</code>
14414		The implementation provides a C-language compilation environment with 32-bit int , long ,
14415		and pointer types and an off_t type using at least 64 bits.

14416		<code>_POSIX_V7_LP64_OFF64</code>	
14417			The implementation provides a C-language compilation environment with 32-bit int and
14418			64-bit long , pointer , and off_t types.
14419		<code>_POSIX_V7_LPBIG_OFFBIG</code>	
14420			The implementation provides a C-language compilation environment with an int type
14421			using at least 32 bits and long , pointer , and off_t types using at least 64 bits.
14422		<code>_POSIX2_C_BIND</code>	
14423			The implementation supports the C-Language Binding option. This symbol shall always
14424			have the value 200xxxL.
14425	CD	<code>_POSIX2_C_DEV</code>	
14426			The implementation supports the C-Language Development Utilities option. If this symbol
14427			is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14428			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14429		<code>_POSIX2_CHAR_TERM</code>	
14430			The implementation supports at least one terminal type.
14431	FD	<code>_POSIX2_FORT_DEV</code>	
14432			The implementation supports the FORTRAN Development Utilities option. If this symbol
14433			is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14434			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14435	FR	<code>_POSIX2_FORT_RUN</code>	
14436			The implementation supports the FORTRAN Runtime Utilities option. If this symbol is
14437			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14438			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14439		<code>_POSIX2_LOCALEDEF</code>	
14440			The implementation supports the creation of locales by the <i>localedef</i> utility. If this symbol is
14441			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14442			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14443	OB BE	<code>_POSIX2_PBS</code>	
14444			The implementation supports the Batch Environment Services and Utilities option. If this
14445			symbol is defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this
14446			symbol reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14447	OB BE	<code>_POSIX2_PBS_ACCOUNTING</code>	
14448			The implementation supports the Batch Accounting option. If this symbol is defined in
14449			<unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
14450			<i>sysconf</i> () shall either be -1 or 200xxxL.
14451	OB BE	<code>_POSIX2_PBS_CHECKPOINT</code>	
14452			The implementation supports the Batch Checkpoint/Restart option. If this symbol is
14453			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14454			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.
14455	OB BE	<code>_POSIX2_PBS_LOCATE</code>	
14456			The implementation supports the Locate Batch Job Request option. If this symbol is defined
14457			in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14458			by <i>sysconf</i> () shall either be -1 or 200xxxL.
14459	OB BE	<code>_POSIX2_PBS_MESSAGE</code>	
14460			The implementation supports the Batch Job Message Request option. If this symbol is
14461			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14462			reported by <i>sysconf</i> () shall either be -1 or 200xxxL.

14463	OB BE	<u>POSIX2_PBS_TRACK</u>	
14464			The implementation supports the Track Batch Job Request option. If this symbol is defined
14465			in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14466			by <i>sysconf()</i> shall either be -1 or 200xxxL.
14467	SD	<u>POSIX2_SW_DEV</u>	
14468			The implementation supports the Software Development Utilities option. If this symbol is
14469			defined in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol
14470			reported by <i>sysconf()</i> shall either be -1 or 200xxxL.
14471	UP	<u>POSIX2_UPE</u>	
14472			The implementation supports the User Portability Utilities option. If this symbol is defined
14473			in <unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported
14474			by <i>sysconf()</i> shall either be -1 or 200xxxL.
14475	XSI	<u>XOPEN_CRYPT</u>	
14476			The implementation supports the X/Open Encryption Option Group.
14477		<u>XOPEN_ENH_I18N</u>	
14478			The implementation supports the Issue 4, Version 2 Enhanced Internationalization Option
14479			Group. This symbol shall always be set to a value other than -1.
14480		<u>XOPEN_REALTIME</u>	
14481			The implementation supports the X/Open Realtime Option Group.
14482		<u>XOPEN_REALTIME_THREADS</u>	
14483			The implementation supports the X/Open Realtime Threads Option Group.
14484		<u>XOPEN_SHM</u>	
14485			The implementation supports the Issue 4, Version 2 Shared Memory Option Group. This
14486			symbol shall always be set to a value other than -1.
14487	OB XSR	<u>XOPEN_STREAMS</u>	
14488			The implementation supports the XSI STREAMS Option Group.
14489	XSI	<u>XOPEN_UNIX</u>	
14490			The implementation supports the XSI option.
14491	UU	<u>XOPEN_UUCP</u>	
14492			The implementation supports the UUCP Utilities option. If this symbol is defined in
14493			<unistd.h>, it shall be defined to be -1, 0, or 200xxxL. The value of this symbol reported by
14494			<i>sysconf()</i> shall be either -1 or 200xxxL.

14495 Execution-Time Symbolic Constants

14496 If any of the following symbolic constants are not defined in the <unistd.h> header, the value
 14497 shall vary depending on the file to which it is applied. If defined, they shall have values suitable
 14498 for use in #if preprocessing directives.

14499 If any of the following symbolic constants are defined to have value -1 in the <unistd.h> header,
 14500 the implementation shall not provide the option on any file; if any are defined to have a value
 14501 other than -1 in the <unistd.h> header, the implementation shall provide the option on all
 14502 applicable files.

14503 All of the following values, whether defined as symbolic constants in <unistd.h> or not, may be
 14504 queried with respect to a specific file using the *pathconf()* or *fpathconf()* functions:

14505 _POSIX_ASYNC_IO
 14506 Asynchronous input or output operations may be performed for the associated file.

14507 _POSIX_PRIO_IO
 14508 Prioritized input or output operations may be performed for the associated file.

14509 _POSIX_SYNC_IO
 14510 Synchronized input or output operations may be performed for the associated file.

14511 If the following symbolic constants are defined in the <unistd.h> header, they apply to files and |
 14512 all paths in all file systems on the implementation:

14513 _POSIX_TIMESTAMP_RESOLUTION
 14514 The resolution in nanoseconds for all file timestamps. |

14515 _POSIX2_SYMLINKS
 14516 Symbolic links can be created. +

14517 **Constants for Functions**

14518 The <unistd.h> header shall define NULL as described in <stddef.h>. |

14519 The <unistd.h> header shall define the following symbolic constants for use with the *access()* |
 14520 function. The values shall be suitable for use in **#if** preprocessing directives.

14521 F_OK Test for existence of file.

14522 R_OK Test for read permission.

14523 W_OK Test for write permission.

14524 X_OK Test for execute (search) permission.

14525 The constants F_OK, R_OK, W_OK, and X_OK and the expressions R_OK|W_OK, R_OK|X_OK,
 14526 and R_OK|W_OK|X_OK shall all have distinct values.

14527 The <unistd.h> header shall define the following symbolic constants for the *confstr()* function: |

14528 _CS_PATH
 14529 This is the value for the *PATH* environment variable that finds all standard utilities.

14530 _CS_POSIX_V7_ILP32_OFF32_CFLAGS
 14531 If *sysconf(SC_V7_ILP32_OFF32)* returns -1 , the meaning of this value is unspecified.
 14532 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an
 14533 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14534 _CS_POSIX_V7_ILP32_OFF32_LDFLAGS
 14535 If *sysconf(SC_V7_ILP32_OFF32)* returns -1 , the meaning of this value is unspecified.
 14536 Otherwise, this value is the set of final options to be given to the *c99* utility to build an
 14537 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14538 _CS_POSIX_V7_ILP32_OFF32_LIBS
 14539 If *sysconf(SC_V7_ILP32_OFF32)* returns -1 , the meaning of this value is unspecified.
 14540 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an
 14541 application using a programming model with 32-bit **int**, **long**, **pointer**, and **off_t** types.

14542 _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
 14543 If *sysconf(SC_V7_ILP32_OFFBIG)* returns -1 , the meaning of this value is unspecified.
 14544 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an
 14545 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14546 **off_t** type using at least 64 bits.

14547 _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
 14548 If *sysconf(SC_V7_ILP32_OFFBIG)* returns -1 , the meaning of this value is unspecified.
 14549 Otherwise, this value is the set of final options to be given to the *c99* utility to build an
 14550 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an

14551 **off_t** type using at least 64 bits.

14552 **_CS_POSIX_V7_ILP32_OFFBIG_LIBS**
 14553 If *sysconf*(**_SC_V7_ILP32_OFFBIG**) returns -1 , the meaning of this value is unspecified.
 14554 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an
 14555 application using a programming model with 32-bit **int**, **long**, and **pointer** types, and an
 14556 **off_t** type using at least 64 bits.

14557 **_CS_POSIX_V7_LP64_OFF64_CFLAGS**
 14558 If *sysconf*(**_SC_V7_LP64_OFF64**) returns -1 , the meaning of this value is unspecified.
 14559 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an
 14560 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14561 types.

14562 **_CS_POSIX_V7_LP64_OFF64_LDFLAGS**
 14563 If *sysconf*(**_SC_V7_LP64_OFF64**) returns -1 , the meaning of this value is unspecified.
 14564 Otherwise, this value is the set of final options to be given to the *c99* utility to build an
 14565 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14566 types.

14567 **_CS_POSIX_V7_LP64_OFF64_LIBS**
 14568 If *sysconf*(**_SC_V7_LP64_OFF64**) returns -1 , the meaning of this value is unspecified.
 14569 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an
 14570 application using a programming model with 32-bit **int** and 64-bit **long**, **pointer**, and **off_t**
 14571 types.

14572 **_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS**
 14573 If *sysconf*(**_SC_V7_LPBIG_OFFBIG**) returns -1 , the meaning of this value is unspecified.
 14574 Otherwise, this value is the set of initial options to be given to the *c99* utility to build an
 14575 application using a programming model with an **int** type using at least 32 bits and **long**,
 14576 **pointer**, and **off_t** types using at least 64 bits.

14577 **_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS**
 14578 If *sysconf*(**_SC_V7_LPBIG_OFFBIG**) returns -1 , the meaning of this value is unspecified.
 14579 Otherwise, this value is the set of final options to be given to the *c99* utility to build an
 14580 application using a programming model with an **int** type using at least 32 bits and **long**,
 14581 **pointer**, and **off_t** types using at least 64 bits.

14582 **_CS_POSIX_V7_LPBIG_OFFBIG_LIBS**
 14583 If *sysconf*(**_SC_V7_LPBIG_OFFBIG**) returns -1 , the meaning of this value is unspecified.
 14584 Otherwise, this value is the set of libraries to be given to the *c99* utility to build an
 14585 application using a programming model with an **int** type using at least 32 bits and **long**,
 14586 **pointer**, and **off_t** types using at least 64 bits.

14587 **_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS**
 14588 This value is a <newline>-separated list of names of programming environments supported
 14589 by the implementation in which the widths of the **blksize_t**, **cc_t**, **mode_t**, **nfds_t**, **pid_t**,
 14590 **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **wchar_t**, and **wint_t** types are no
 14591 greater than the width of type **long**. The format of each name shall be suitable for use with
 14592 the *getconf* $-v$ option.

14593 **_CS_V7_ENV**
 14594 This is the value that provides the environment variable information (other than that
 14595 provided by **_CS_PATH**) that is required by the implementation to create a conforming
 14596 environment, as described in the implementation's conformance documentation.

14597	OB	The following symbolic constants are reserved for compatibility with Issue 6:	
14598		_CS_POSIX_V6_ILP32_OFF32_CFLAGS	
14599		_CS_POSIX_V6_ILP32_OFF32_LDFLAGS	
14600		_CS_POSIX_V6_ILP32_OFF32_LIBS	
14601		_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS	
14602		_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS	
14603		_CS_POSIX_V6_ILP32_OFFBIG_LIBS	
14604		_CS_POSIX_V6_LP64_OFF64_CFLAGS	
14605		_CS_POSIX_V6_LP64_OFF64_LDFLAGS	
14606		_CS_POSIX_V6_LP64_OFF64_LIBS	
14607		_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS	
14608		_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS	
14609		_CS_POSIX_V6_LPBIG_OFFBIG_LIBS	
14610		_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS	
14611		_CS_V6_ENV	
14612			-
14613		The <unistd.h> header shall define SEEK_CUR, SEEK_END, and SEEK_SET as described in	
14614		<stdio.h>.	-
14615		The <unistd.h> header shall define the following symbolic constants as possible values for the	
14616		<i>function</i> argument to the <i>lockf()</i> function:	
14617		F_LOCK Lock a section for exclusive use.	
14618		F_TEST Test section for locks by other processes.	
14619		F_TLOCK Test and lock a section for exclusive use.	
14620		F_ULOCK Unlock locked sections.	
14621		The <unistd.h> header shall define the following symbolic constants for <i>pathconf()</i> :	
14622		_PC_2_SYMLINKS	
14623		_PC_ALLOC_SIZE_MIN	
14624		_PC_ASYNC_IO	
14625		_PC_CHOWN_RESTRICTED	
14626		_PC_FILESIZEBITS	
14627		_PC_LINK_MAX	
14628		_PC_MAX_CANON	
14629		_PC_MAX_INPUT	
14630		_PC_NAME_MAX	
14631		_PC_NO_TRUNC	
14632		_PC_PATH_MAX	
14633		_PC_PIPE_BUF	
14634		_PC_PRIO_IO	
14635		_PC_REC_INCR_XFER_SIZE	
14636		_PC_REC_MAX_XFER_SIZE	+
14637		_PC_REC_MIN_XFER_SIZE	
14638		_PC_REC_XFER_ALIGN	
14639		_PC_SYMLINK_MAX	
14640		_PC_SYNC_IO	
14641		_PC_TIMESTAMP_RESOLUTION	+
14642		_PC_VDISABLE	
14643		The <unistd.h> header shall define the following symbolic constants for <i>sysconf()</i> :	
14644		_SC_2_C_BIND	
14645		_SC_2_C_DEV	

14646 _SC_2_CHAR_TERM
 14647 _SC_2_FORT_DEV
 14648 _SC_2_FORT_RUN
 14649 _SC_2_LOCALEDEF
 14650 _SC_2_PBS
 14651 _SC_2_PBS_ACCOUNTING
 14652 _SC_2_PBS_CHECKPOINT
 14653 _SC_2_PBS_LOCATE
 14654 _SC_2_PBS_MESSAGE
 14655 _SC_2_PBS_TRACK
 14656 _SC_2_SW_DEV
 14657 _SC_2_UPE
 14658 _SC_2_VERSION
 14659 _SC_ADVISORY_INFO
 14660 _SC_AIO_LISTIO_MAX
 14661 _SC_AIO_MAX
 14662 _SC_AIO_PRIO_DELTA_MAX
 14663 _SC_ARG_MAX
 14664 _SC_ASYNCHRONOUS_IO
 14665 _SC_ATEXIT_MAX
 14666 _SC_BARRIERS
 14667 _SC_BC_BASE_MAX
 14668 _SC_BC_DIM_MAX
 14669 _SC_BC_SCALE_MAX
 14670 _SC_BC_STRING_MAX
 14671 _SC_CHILD_MAX
 14672 _SC_CLK_TCK
 14673 _SC_CLOCK_SELECTION
 14674 _SC_COLL_WEIGHTS_MAX
 14675 _SC_CPUTIME
 14676 _SC_DELAYTIMER_MAX
 14677 _SC_EXPR_NEST_MAX
 14678 _SC_FSYNC
 14679 _SC_GETGR_R_SIZE_MAX
 14680 _SC_GETPW_R_SIZE_MAX
 14681 _SC_HOST_NAME_MAX
 14682 _SC_IOV_MAX
 14683 _SC_IPV6
 14684 _SC_JOB_CONTROL
 14685 _SC_LINE_MAX
 14686 _SC_LOGIN_NAME_MAX
 14687 _SC_MAPPED_FILES
 14688 _SC_MEMLOCK
 14689 _SC_MEMLOCK_RANGE
 14690 _SC_MEMORY_PROTECTION
 14691 _SC_MESSAGE_PASSING
 14692 _SC_MONOTONIC_CLOCK
 14693 _SC_MQ_OPEN_MAX
 14694 _SC_MQ_PRIO_MAX
 14695 _SC_NGROUPS_MAX
 14696 _SC_OPEN_MAX
 14697 _SC_PAGE_SIZE
 14698 _SC_PAGESIZE
 14699 _SC_PRIORITIZED_IO

14700 _SC_PRIORITY_SCHEDULING
 14701 _SC_RAW_SOCKETS
 14702 _SC_RE_DUP_MAX
 14703 _SC_READER_WRITER_LOCKS
 14704 _SC_REALTIME_SIGNALS
 14705 _SC_REGEX
 14706 _SC_RTSIG_MAX
 14707 _SC_SAVED_IDS
 14708 _SC_SEM_NSEMS_MAX
 14709 _SC_SEM_VALUE_MAX
 14710 _SC_SEMAPHORES
 14711 _SC_SHARED_MEMORY_OBJECTS
 14712 _SC_SHELL
 14713 _SC_SIGQUEUE_MAX
 14714 _SC_SPAWN
 14715 _SC_SPIN_LOCKS
 14716 _SC_SPORADIC_SERVER
 14717 _SC_SS_REPL_MAX
 14718 _SC_STREAM_MAX
 14719 _SC_SYMLOOP_MAX
 14720 _SC_SYNCHRONIZED_IO
 14721 _SC_THREAD_ATTR_STACKADDR
 14722 _SC_THREAD_ATTR_STACKSIZE
 14723 _SC_THREAD_CPU_TIME
 14724 _SC_THREAD_DESTRUCTOR_ITERATIONS
 14725 _SC_THREAD_KEYS_MAX
 14726 _SC_THREAD_PRIO_INHERIT
 14727 _SC_THREAD_PRIO_PROTECT
 14728 _SC_THREAD_PRIORITY_SCHEDULING
 14729 _SC_THREAD_PROCESS_SHARED
 14730 _SC_THREAD_SAFE_FUNCTIONS
 14731 _SC_THREAD_SPARADIC_SERVER
 14732 _SC_THREAD_STACK_MIN
 14733 _SC_THREAD_THREADS_MAX
 14734 _SC_THREADS
 14735 _SC_TIMEOUTS
 14736 _SC_TIMER_MAX
 14737 _SC_TIMERS
 14738 _SC_TRACE
 14739 _SC_TRACE_EVENT_FILTER
 14740 _SC_TRACE_EVENT_NAME_MAX
 14741 _SC_TRACE_INHERIT
 14742 _SC_TRACE_LOG
 14743 _SC_TRACE_NAME_MAX
 14744 _SC_TRACE_SYS_MAX
 14745 _SC_TRACE_USER_EVENT_MAX
 14746 _SC_TTY_NAME_MAX
 14747 _SC_TYPED_MEMORY_OBJECTS
 14748 _SC_TZNAME_MAX
 14749 _SC_V7_ILP32_OFF32
 14750 _SC_V7_ILP32_OFFBIG
 14751 _SC_V7_LP64_OFF64
 14752 _SC_V7_LP64_OFFBIG

14753 OB `_SC_V6_ILP32_OFF32`
 14754 `_SC_V6_ILP32_OFFBIG`
 14755 `_SC_V6_LP64_OFF64`
 14756 `_SC_V6_LP64_OFFBIG`
 14757 `_SC_VERSION`
 14758 `_SC_XOPEN_CRYPT`
 14759 `_SC_XOPEN_ENH_I18N`
 14760 `_SC_XOPEN_REALTIME`
 14761 `_SC_XOPEN_REALTIME_THREADS`
 14762 `_SC_XOPEN_SHM`
 14763 `_SC_XOPEN_STREAMS`
 14764 `_SC_XOPEN_UNIX`
 14765 `_SC_XOPEN_UUCP`
 14766 `_SC_XOPEN_VERSION`

14767 The two constants `_SC_PAGESIZE` and `_SC_PAGE_SIZE` may be defined to have the same value.

14768 The <unistd.h> header shall define the following symbolic constants for file streams: |

14769 `STDERR_FILENO` File number of *stderr*; 2.
 14770 `STDIN_FILENO` File number of *stdin*; 0.
 14771 `STDOUT_FILENO` File number of *stdout*; 1.

14772 The following symbolic constant shall be defined for terminal special character handling: +

14773 `_POSIX_VDISABLE` This symbol shall be defined to be the value of a character that shall +
 14774 disable terminal special character handling as described in [Section 11.2.6](#) +
 14775 (on page 196). This symbol shall always be set to a value other than -1. +

14776 **Type Definitions**

14777 The `size_t`, `ssize_t`, `uid_t`, `gid_t`, `off_t`, and `pid_t` types shall be defined as described in
 14778 <sys/types.h>.

14779 The `intptr_t` type shall be defined as described in <inttypes.h>.

14780 **Declarations**

14781 The following shall be declared as functions and may also be defined as macros. Function
 14782 prototypes shall be provided.

14783 `int` `access(const char *, int);`
 14784 `unsigned` `alarm(unsigned);`
 14785 `int` `chdir(const char *);`
 14786 `int` `chown(const char *, uid_t, gid_t);`
 14787 `int` `close(int);`
 14788 `size_t` `confstr(int, char *, size_t);`
 14789 XSI `char` `*crypt(const char *, const char *);`
 14790 CX `char` `*ctermid(char *);`
 14791 `int` `dup(int);`

14792		int	dup2(int, int);	
14793	XSI	void	encrypt(char [64], int);	
14794		int	execl(const char *, const char *, ...);	
14795		int	execle(const char *, const char *, ...);	
14796		int	execlp(const char *, const char *, ...);	
14797		int	execv(const char *, char *const []);	
14798		int	execve(const char *, char *const [], char *const []);	
14799		int	execvp(const char *, char *const []);	
14800		void	_exit(int);	
14801		int	faccessat(int, const char *, int, int);	
14802		int	fchdir(int);	
14803		int	fchown(int, uid_t, gid_t);	-
14804		int	fchownat(int, const char *, uid_t, gid_t, int);	
14805	SIO	int	fdatasync(int);	
14806		int	fexecve(int, char *const [], char *const []);	
14807		pid_t	fork(void);	
14808		long	fpathconf(int, int);	
14809	FSC	int	fsync(int);	
14810		int	ftruncate(int, off_t);	
14811		char	*getcwd(char *, size_t);	
14812		gid_t	getegid(void);	
14813		uid_t	geteuid(void);	
14814		gid_t	getgid(void);	
14815		int	getgroups(int, gid_t []);	
14816	XSI	long	gethostid(void);	
14817		int	gethostname(char *, size_t);	
14818		char	*getlogin(void);	
14819		int	getlogin_r(char *, size_t);	
14820		int	getopt(int, char * const [], const char *);	
14821		pid_t	getpgid(pid_t);	
14822		pid_t	getpgrp(void);	
14823		pid_t	getpid(void);	
14824		pid_t	getppid(void);	
14825		pid_t	getsid(pid_t);	
14826		uid_t	getuid(void);	
14827		int	isatty(int);	
14828		int	lchown(const char *, uid_t, gid_t);	
14829		int	link(const char *, const char *);	
14830		int	linkat(int, const char *, int, const char *, int);	
14831	XSI	int	lockf(int, int, off_t);	
14832		off_t	lseek(int, off_t, int);	
14833	XSI	int	nice(int);	
14834		long	pathconf(const char *, int);	
14835		int	pause(void);	
14836		int	pipe(int [2]);	
14837		ssize_t	pread(int, void *, size_t, off_t);	
14838		ssize_t	pwrite(int, const void *, size_t, off_t);	
14839		ssize_t	read(int, void *, size_t);	
14840		ssize_t	readlink(const char *restrict, char *restrict, size_t);	
14841		ssize_t	readlinkat(int, const char *restrict, char *restrict, size_t);	
14842		int	rmdir(const char *);	
14843		int	setegid(gid_t);	
14844		int	seteuid(uid_t);	
14845		int	setgid(gid_t);	

```

14846     int          setpgid(pid_t, pid_t);
14847  OB XSI  pid_t      setpgrp(void);
14848  XSI     int          setregid(gid_t, gid_t);
14849     int          setreuid(uid_t, uid_t);
14850     pid_t        setsid(void);
14851     int          setuid(uid_t);
14852     unsigned     sleep(unsigned);
14853  XSI     void        swab(const void *restrict, void *restrict, ssize_t);
14854     int          symlink(const char *, const char *);
14855     int          symlinkat(const char *, int, const char *);
14856  XSI     void        sync(void);
14857     long         sysconf(int);
14858     pid_t        tcgetpgrp(int);
14859     int          tcsetpgrp(int, pid_t);
14860     int          truncate(const char *, off_t);
14861     char         *ttyname(int);
14862     int          ttyname_r(int, char *, size_t);
14863     int          unlink(const char *);
14864     int          unlinkat(int, const char *, int);
14865     ssize_t      write(int, const void *, size_t);

```

14866 Implementations may also include the *pthread_atfork()* prototype as defined in <pthread.h>.

14867 The following external variables shall be declared:

```

14868     extern char  *optarg;
14869     extern int   opterr, optind, optopt;

```

14870 APPLICATION USAGE

14871 POSIX.1-200x only describes the behavior of systems that claim conformance to it. However,
 14872 application developers who want to write applications that adapt to other versions of this
 14873 standard (or to systems that do not conform to any POSIX standard) may find it useful to code
 14874 them so as to conditionally compile different code depending on the value of
 14875 `_POSIX_VERSION`, for example:

```

14876     #if _POSIX_VERSION >= 200112L
14877     /* Use the newer function that copes with large files. */
14878     off_t pos=ftello(fp);
14879     #else
14880     /* Either this is an old version of POSIX, or _POSIX_VERSION is
14881        not even defined, so use the traditional function. */
14882     long pos=ftell(fp);
14883     #endif

```

14884 Earlier versions of POSIX.1-200x and of the Single UNIX Specification can be identified by the
 14885 following macros:

```

14886     POSIX.1-1988 standard
14887         _POSIX_VERSION == 198808L
14888     POSIX.1-1990 standard
14889         _POSIX_VERSION == 199009L
14890     ISO POSIX-1:1996 standard
14891         _POSIX_VERSION == 199506L
14892     Single UNIX Specification, Version 1
14893         _XOPEN_UNIX and _XOPEN_VERSION == 4

```

14894 Single UNIX Specification, Version 2
 14895 `_XOPEN_UNIX` and `_XOPEN_VERSION == 500` |

14896 ISO POSIX-1:2001 and Single UNIX Specification, Version 3
 14897 `_POSIX_VERSION == 200112L`, plus (if the XSI option is supported) `_XOPEN_UNIX` and |
 14898 `_XOPEN_VERSION == 600`

14899 POSIX.1-200x does not make any attempt to define application binary interaction with the
 14900 underlying operating system. However, application developers may find it useful to query
 14901 `_SC_VERSION` at runtime via `sysconf()` to determine whether the current version of the
 14902 operating system supports the necessary functionality as in the following program fragment:

```
14903 if (sysconf(_SC_VERSION) < 200xxxL) {
14904     fprintf(stderr, "POSIX.1-200x system required, terminating \n");
14905     exit(1);
14906 }
```

14907 New applications should not use `_XOPEN_SHM` or `_XOPEN_ENH_I18N`.

14908 RATIONALE

14909 As POSIX.1-200x evolved, certain options became sufficiently standardized that it was
 14910 concluded that simply requiring one of the option choices was simpler than retaining the option.
 14911 However, for backwards-compatibility, the option flags (with required constant values) are
 14912 retained.

14913 Version Test Macros

14914 The standard developers considered altering the definition of `_POSIX_VERSION` and removing
 14915 `_SC_VERSION` from the specification of `sysconf()` since the utility to an application was deemed
 14916 by some to be minimal, and since the implementation of the functionality is potentially
 14917 problematic. However, they recognized that support for existing application binaries is a
 14918 concern to manufacturers, application developers, and the users of implementations conforming
 14919 to POSIX.1-200x.

14920 While the example using `_SC_VERSION` in the APPLICATION USAGE section does not provide
 14921 the greatest degree of imaginable utility to the application developer or user, it is arguably better
 14922 than a **core** file or some other equally obscure result. (It is also possible for implementations to
 14923 encode and recognize application binaries compiled in various POSIX.1-conforming
 14924 environments, and modify the semantics of the underlying system to conform to the
 14925 expectations of the application.) For the reasons outlined in the preceding paragraphs and in the
 14926 APPLICATION USAGE section, the standard developers elected to retain the `_POSIX_VERSION`
 14927 and `_SC_VERSION` functionality.

14928 Compile-Time Symbolic Constants for System-Wide Options

14929 POSIX.1-200x includes support in certain areas for the newly adopted policy governing options
 14930 and stubs.

14931 This policy provides flexibility for implementations in how they support options. It also
 14932 specifies how conforming applications can adapt to different implementations that support
 14933 different sets of options. It allows the following:

- 14934 1. If an implementation has no interest in supporting an option, it does not have to provide
 14935 anything associated with that option beyond the announcement that it does not support
 14936 it.
- 14937 2. An implementation can support a partial or incompatible version of an option (as a non-
 14938 standard extension) as long as it does not claim to support the option.

- 14939 3. An application can determine whether the option is supported. A strictly conforming
 14940 application must check this announcement mechanism before first using anything
 14941 associated with the option.

14942 There is an important implication of this policy. POSIX.1-200x cannot dictate the behavior of
 14943 interfaces associated with an option when the implementation does not claim to support the
 14944 option. In particular, it cannot require that a function associated with an unsupported option
 14945 will fail if it does not perform as specified. However, this policy does not prevent a standard
 14946 from requiring certain functions to always be present, but that they shall always fail on some
 14947 implementations. The *setpgid()* function in the POSIX.1-1990 standard, for example, is
 14948 considered appropriate.

14949 The POSIX standards include various options, and the C-language binding support for an
 14950 option implies that the implementation must supply data types and function interfaces. An
 14951 application must be able to discover whether the implementation supports each option.

14952 Any application must consider the following three cases for each option:

- 14953 1. Option never supported.

14954 The implementation advertises at compile time that the option will never be supported.
 14955 In this case, it is not necessary for the implementation to supply any of the data types or
 14956 function interfaces that are provided only as part of the option. The implementation
 14957 might provide data types and functions that are similar to those defined by POSIX.1-200x,
 14958 but there is no guarantee for any particular behavior.

- 14959 2. Option always supported.

14960 The implementation advertises at compile time that the option will always be supported.
 14961 In this case, all data types and function interfaces shall be available and shall operate as
 14962 specified.

- 14963 3. Option might or might not be supported.

14964 Some implementations might not provide a mechanism to specify support of options at
 14965 compile time. In addition, the implementation might be unable or unwilling to specify
 14966 support or non-support at compile time. In either case, any application that might use the
 14967 option at runtime must be able to compile and execute. The implementation must
 14968 provide, at compile time, all data types and function interfaces that are necessary to allow
 14969 this. In this situation, there must be a mechanism that allows the application to query, at
 14970 runtime, whether the option is supported. If the application attempts to use the option
 14971 when it is not supported, the result is unspecified unless explicitly specified otherwise in
 14972 POSIX.1-200x.

14973 FUTURE DIRECTIONS

14974 None.

14975 SEE ALSO

14976 <inttypes.h>, <limits.h>, <stddef.h>, <stdio.h>, <sys/socket.h>, <sys/types.h>, <termios.h>, -
 14977 <wctype.h>

14978 XSH *access()*, *alarm()*, *chdir()*, *chown*, *close()*, *crypt()*, *dup()*, *encrypt()*, *exec*, *exit()*, *fchdir()*, |
 14979 *fchown()*, *fcntl()*, *fork()*, *fpathconf()*, *fsync()*, *ftruncate()*, *getcwd()*, *getegid()*, *geteuid()*, *getgid()*, |
 14980 *getgroups()*, *gethostid()*, *gethostname()*, *getlogin()*, *getpgid()*, *getpgrp()*, *getpid()*, *getppid()*, *getsid()*, |
 14981 *getuid()*, *isatty()*, *lchown()*, *link*, *lockf()*, *lseek()*, *nice*, *pause()*, *pipe()*, *read*, *readlink()*, *rmdir*, |
 14982 *setgid()*, *setpgid()*, *setpgrp()*, *setregid()*, *setreuid()*, *setsid()*, *setuid()*, *sleep*, *swab()*, *symlink()*, |
 14983 *sync()*, *sysconf()*, *tcgetpgrp()*, *tcsetpgrp()*, *truncate()*, *ttyname()*, *unlink*, *write* |

CHANGE HISTORY14984
14985

First released in Issue 1. Derived from Issue 1 of the SVID.

14986
14987
14988**Issue 5**

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

14989
14990
14991

The symbolic constants `_XOPEN_REALTIME` and `_XOPEN_REALTIME_THREADS` are added. `_POSIX2_C_BIND`, `_XOPEN_ENH_I18N`, and `_XOPEN_SHM` must now be set to a value other than `-1` by a conforming implementation.

14992

Large File System extensions are added.

14993

The type of the argument to `sbrk()` is changed from `int` to `intptr_t`.

14994
14995
14996

`_XBS_` constants are added to the list of constants for Options and Option Groups, to the list of constants for the `confstr()` function, and to the list of constants to the `sysconf()` function. These are all marked EX.

14997

Issue 6

14998

`_POSIX2_C_VERSION` is removed.

14999

The Open Group Corrigendum U026/4 is applied, adding the prototype for `fdatasync()`.

15000
15001

The Open Group Corrigendum U026/1 is applied, adding the symbols `_SC_XOPEN_LEGACY`, `_SC_XOPEN_REALTIME`, and `_SC_XOPEN_REALTIME_THREADS`.

15002
15003

The symbols `_XOPEN_STREAMS` and `_SC_XOPEN_STREAMS` are added to support the XSI STREAMS Option Group.

15004
15005

Text in the DESCRIPTION relating to conformance requirements is moved elsewhere in IEEE Std 1003.1-2001.

15006

The LEGACY symbol `_SC_PASS_MAX` is removed.

15007
15008

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

15009

- The `_CS_POSIX_*` and `_CS_XBS5_*` constants are added for the `confstr()` function.

15010

- The `_SC_XBS5_*` constants are added for the `sysconf()` function.

15011

- The symbolic constants `F_ULOCK`, `F_LOCK`, `F_TLOCK`, and `F_TEST` are added.

15012

- The `uid_t`, `gid_t`, `off_t`, `pid_t`, and `useconds_t` types are mandated.

15013

The `gethostname()` prototype is added for sockets.

15014

A new section is added for System-Wide Options.

15015

Function prototypes for `setegid()` and `seteuid()` are added.

15016

Option symbolic constants are added for `_POSIX_ADVISORY_INFO`, `_POSIX_CPUTIME`, `_POSIX_SPAWN`, `_POSIX_SPORADIC_SERVER`, `_POSIX_THREAD_CPUTIME`, `_POSIX_THREAD_SPORADIC_SERVER`, and `_POSIX_TIMEOUTS`, and `pathconf()` variables are added for `_PC_ALLOC_SIZE_MIN`, `_PC_REC_INCR_XFER_SIZE`, `_PC_REC_MAX_XFER_SIZE`, `_PC_REC_MIN_XFER_SIZE`, and `_PC_REC_XFER_ALIGN` for alignment with IEEE Std 1003.1d-1999.

15022

The following are added for alignment with IEEE Std 1003.1j-2000:

15023

- Option symbolic constants `_POSIX_BARRIERS`, `_POSIX_CLOCK_SELECTION`, `_POSIX_MONOTONIC_CLOCK`, `_POSIX_READER_WRITER_LOCKS`, `_POSIX_SPIN_LOCKS`, and `_POSIX_TYPED_MEMORY_OBJECTS`

15024

15025

15026 • *sysconf()* variables `_SC_BARRIERS`, `_SC_CLOCK_SELECTION`,
15027 `_SC_MONOTONIC_CLOCK`, `_SC_READER_WRITER_LOCKS`, `_SC_SPIN_LOCKS`, and
15028 `_SC_TYPED_MEMORY_OBJECTS`

15029 The `_SC_XBS5` macros associated with the ISO/IEC 9899:1990 standard are marked LEGACY,
15030 and new equivalent `_SC_V6` macros associated with the ISO/IEC 9899:1999 standard are
15031 introduced.

15032 The *getwd()* function is marked LEGACY.

15033 The **restrict** keyword is added to the prototypes for *readlink()* and *swab()*.

15034 Constants for options are now harmonized, so when supported they take the year of approval of
15035 IEEE Std 1003.1-2001 as the value.

15036 The following are added for alignment with IEEE Std 1003.1q-2000:

15037 • Optional symbolic constants `_POSIX_TRACE`, `_POSIX_TRACE_EVENT_FILTER`,
15038 `_POSIX_TRACE_LOG`, and `_POSIX_TRACE_INHERIT`

15039 • The *sysconf()* symbolic constants `_SC_TRACE`, `_SC_TRACE_EVENT_FILTER`,
15040 `_SC_TRACE_LOG`, and `_SC_TRACE_INHERIT`

15041 The *brk()* and *sbrk()* LEGACY functions are removed.

15042 The Open Group Base Resolution bwg2001-006 is applied, which reworks the XSI versioning
15043 information.

15044 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter for
15045 *gethostname()* from `socklen_t` to `size_t`.

15046 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/2 is applied, changing “Thread Stack
15047 Address Size” to “Thread Stack Size Attribute”.

15048 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/20 is applied, adding the `_POSIX_IPV6`,
15049 `_SC_V6`, and `_SC_RAW_SOCKETS` symbols.

15050 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/21 is applied, correcting the description
15051 in “Constants for Functions” for the `_CS_POSIX_V6_LP64_OFF64_CFLAGS`,
15052 `_CS_POSIX_V6_LP64_OFF64_LDFLAGS`, and `_CS_POSIX_V6_LP64_OFF64_LIBS` symbols.

15053 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/22 is applied, removing the shading for
15054 the `_PC*` and `_SC*` constants, since these are mandatory on all implementations.

15055 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/23 is applied, adding the
15056 `_PC_SYMLINK_MAX` and `_SC_SYMLINK_MAX` constants.

15057 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/24 is applied, correcting the shading and
15058 margin code for the *fsync()* function.

15059 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/25 is applied, adding the following text to
15060 the APPLICATION USAGE section: “New applications should not use `_XOPEN_SHM` or
15061 `_XOPEN_ENH_I18N`.”.

15062 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/29 is applied, clarifying the requirements
15063 for when constants for Options and Option Groups can be defined or undefined.

15064 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/30 is applied, changing the
15065 `_V6_ILP32_OFF32`, `_V6_ILP32_OFFBIG`, `_V6_LP64_OFF64`, and `_V6_LPBIG_OFFBIG` symbols to
15066 `_POSIX_V6_ILP32_OFF32`, `_POSIX_V6_ILP32_OFFBIG`, `_POSIX_V6_LP64_OFF64`, and
15067 `_POSIX_V6_LPBIG_OFFBIG`, respectively. This is for consistency with the *sysconf()* and *c99*
15068 reference pages.

15069 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/31 is applied, adding that the format of

15070 names of programming environments can be obtained using the *getconf* -v option.

15071 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/32 is applied, deleting the
15072 *_SC_FILE_LOCKING*, *_SC_2_C_VERSION*, and *_SC_XOPEN_XCU_VERSION* constants.

15073 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/33 is applied, adding
15074 *_SC_SS_REPL_MAX*, *_SC_TRACE_EVENT_NAME_MAX*, *_SC_TRACE_NAME_MAX*,
15075 *_SC_TRACE_SYS_MAX*, and *_SC_TRACE_USER_EVENT_MAX* to the list of symbolic constants
15076 for *sysconf*().

15077 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/34 is applied, updating the prototype for
15078 the *symlink*() function to match that in the System Interfaces volume of IEEE Std 1003.1-2001.

15079 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/35 is applied, adding *_PC_2_SYMLINKS*
15080 to the symbolic constants list for *pathconf*(). This corresponds to the definition of
15081 *POSIX2_SYMLINKS* in the Shell and Utilities volume of IEEE Std 1003.1-2001.

15082 **Issue 7**

15083 Austin Group Interpretations 1003.1-2001 #026 and #047 are applied. +

15084 SD5-XBD-ERN-41 is applied, adding the *_POSIX2_SYMLINKS* constant.

15085 SD5-XBD-ERN-76 and SD5-XBD-ERN-77 are applied. |

15086 Symbols to support the UUCP Utilities option are added.

15087 The variables for the supported programming environments are updated to be V7.

15088 The LEGACY and obsolescent symbols are removed.

15089 The *faccessat*(), *fchownat*(), *fexecve*(), *linkat*(), *readlinkat*(), *symlinkat*(), and *unlinkat*() functions -
15090 are added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

15091 The *_POSIX_TRACE** constants from the Trace option are marked obsolescent.

15092 The *_POSIX2_PBS** constants from the Batch Environment Services and Utilities option are
15093 marked obsolescent.

15094 Functionality relating to the Asynchronous Input and Output, Barriers, Clock Selection, Memory
15095 Mapped Files, Memory Protection, Realtime Signals Extension, Semaphores, Spin Locks,
15096 Threads, Timeouts, and Timers options is moved to the Base.

15097 Functionality relating to the Thread Priority Protection and Thread Priority Inheritance options
15098 is changed to be Non-Robust Mutex or Robust Mutex Priority Protection and Non-Robust Mutex
15099 or Robust Mutex Priority Inheritance, respectively.

15100 This reference page is clarified with respect to macros and symbolic constants. +

15101 Changes are made related to support for finegrained timestamps and the +
15102 *_POSIX_TIMESTAMP_RESOLUTION* constant is added.

15103 **NAME**
 15104 `utime.h` — access and modification times structure

15105 **SYNOPSIS**
 15106 OB `#include <utime.h>`

15107 **DESCRIPTION**
 15108 The <**utime.h**> header shall declare the structure **utimbuf**, which shall include the following
 15109 members:

15110 `time_t` `actime` Access time.
 15111 `time_t` `modtime` Modification time.

15112 The times shall be measured in seconds since the Epoch.

15113 The type **time_t** shall be defined as described in <**sys/types.h**>.

15114 The following shall be declared as a function and may also be defined as a macro. A function
 15115 prototype shall be provided.

15116 `int utime(const char *, const struct utimbuf *);`

15117 **APPLICATION USAGE**
 15118 The `utime()` function only allows setting file timestamps to the nearest second. Applications |
 15119 should use the `utimensat()` function instead. See <**sys/stat.h**>.

15120 **RATIONALE**
 15121 None.

15122 **FUTURE DIRECTIONS**
 15123 The <**utime.h**> header may be removed in a future version. |

15124 **SEE ALSO**
 15125 [<sys/stat.h>](#), [<sys/types.h>](#)
 15126 XSH [futimens\(\)](#), [utime\(\)](#) |

15127 **CHANGE HISTORY**
 15128 First released in Issue 3.

15129 **Issue 6**
 15130 The following new requirements on POSIX implementations derive from alignment with the
 15131 Single UNIX Specification:
 15132 • The **time_t** type is defined.

15133 **Issue 7**
 15134 The <**utime.h**> header is marked obsolescent. +

15135 **NAME**
 15136 utmpx.h — user accounting database definitions

15137 **SYNOPSIS**
 15138 XSI `#include <utmpx.h>`

15139 **DESCRIPTION**
 15140 The <utmpx.h> header shall define the **utmpx** structure that shall include at least the following
 15141 members:

15142	char	ut_user[]	User login name.
15143	char	ut_id[]	Unspecified initialization process identifier.
15144	char	ut_line[]	Device name.
15145	pid_t	ut_pid	Process ID.
15146	short	ut_type	Type of entry.
15147	struct timeval	ut_tv	Time entry was made.

15148 The **pid_t** type shall be defined through **typedef** as described in <sys/types.h>.

15149 The **timeval** structure shall be defined as described in <sys/time.h>.

15150 Inclusion of the <utmpx.h> header may also make visible all symbols from <sys/time.h>.

15151 The following symbolic constants shall be defined as possible values for the *ut_type* member of
 15152 the **utmpx** structure:

15153	EMPTY	No valid user accounting information.
15154	BOOT_TIME	Identifies time of system boot.
15155	OLD_TIME	Identifies time when system clock changed.
15156	NEW_TIME	Identifies time after system clock changed.
15157	USER_PROCESS	Identifies a process.
15158	INIT_PROCESS	Identifies a process spawned by the init process.
15159	LOGIN_PROCESS	Identifies the session leader of a logged-in user.
15160	DEAD_PROCESS	Identifies a session leader who has exited.

15161 The following shall be declared as functions and may also be defined as macros. Function
 15162 prototypes shall be provided.

```

15163 void          endutxent(void);
15164 struct utmpx *getutxent(void);
15165 struct utmpx *getutxid(const struct utmpx *);
15166 struct utmpx *getutxline(const struct utmpx *);
15167 struct utmpx *pututxline(const struct utmpx *);
15168 void          setutxent(void);

```

<utmpx.h>*Headers*

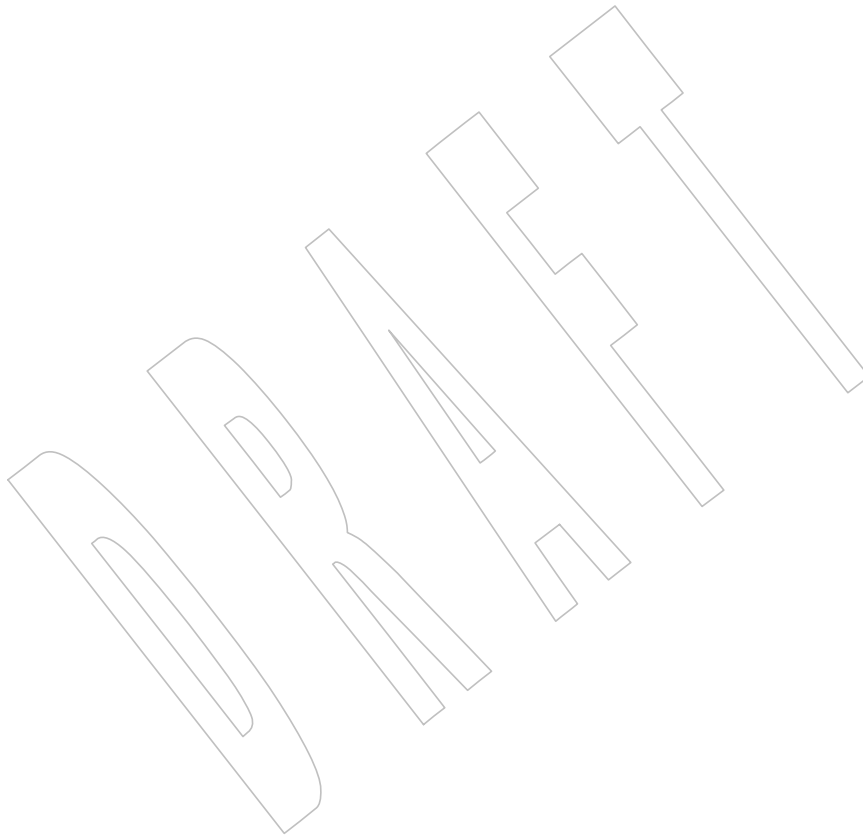
15169 **APPLICATION USAGE**
15170 None.

15171 **RATIONALE**
15172 None.

15173 **FUTURE DIRECTIONS**
15174 None.

15175 **SEE ALSO**
15176 [<sys/time.h>](#), [<sys/types.h>](#)
15177 XSH *endutxent()*

15178 **CHANGE HISTORY**
15179 First released in Issue 4, Version 2.



15180 **NAME**

15181 wchar.h — wide-character handling

15182 **SYNOPSIS**

15183 #include <wchar.h>

15184 **DESCRIPTION**

15185 CX Some of the functionality described on this reference page extends the ISO C standard.
 15186 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 15187 enable the visibility of these symbols in this header.

15188 The <wchar.h> header shall define the following types:

15189 **wchar_t** As described in <stddef.h>.15190 **wint_t** An integer type capable of storing any valid value of **wchar_t** or WEOF.

15191 OB XSI **wctype_t** A scalar type of a data object that can hold values which represent locale-
 15192 specific character classification.

15193 **mbstate_t** An object type other than an array type that can hold the conversion state
 15194 information necessary to convert between sequences of (possibly multi-byte)
 15195 CX characters and wide characters. If a codeset is being used such that an
 15196 **mbstate_t** needs to preserve more than two levels of reserved state, the results
 15197 are unspecified.

15198 CX **FILE** As described in <stdio.h>.15199 **size_t** As described in <stddef.h>.15200 CX **va_list** As described in <stdarg.h>.

15201 The implementation shall support one or more programming environments in which the width
 15202 of **wint_t** is no greater than the width of type **long**. The names of these programming
 15203 environments can be obtained using the *confstr()* function or the *getconf* utility.

15204 The <wchar.h> header shall define the following macros: +

15205 WCHAR_MAX As described in <stdint.h>. +

15206 WCHAR_MIN As described in <stdint.h>. +

15207 WEOF Constant expression of type **wint_t** that is returned by several WP functions to +
 15208 indicate end-of-file. +

15209 NULL As described in <stddef.h>. +

15210 The tag **tm** shall be declared as naming an incomplete structure type, the contents of which are +
 15211 described in the header <time.h>. +

15212 CX Inclusion of the <wchar.h> header may make visible all symbols from the headers <ctype.h>, +
 15213 <string.h>, <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, and <time.h>. +

15214 The following shall be declared as functions and may also be defined as macros. Function
 15215 prototypes shall be provided for use with ISO C standard compilers.

15216 wint_t btowc(int);

15217 wint_t fgetwc(FILE *);

15218 wchar_t *fgetws(wchar_t *restrict, int, FILE *restrict);

15219 wint_t fputwc(wchar_t, FILE *);

15220 int fputws(const wchar_t *restrict, FILE *restrict);

15221 int fwide(FILE *, int);

<wchar.h>

Headers

```

15222         int          fwprintf(FILE *restrict, const wchar_t *restrict, ...);
15223         int          fwscanf(FILE *restrict, const wchar_t *restrict, ...);
15224         wint_t       getwc(FILE *);
15225         wint_t       getwchar(void);
15226     OB XSI         int          iswalnum(wint_t);
15227                 int          iswalpha(wint_t);
15228                 int          iswcntrl(wint_t);
15229                 int          iswctype(wint_t, wctype_t);
15230                 int          iswdigit(wint_t);
15231                 int          iswgraph(wint_t);
15232                 int          iswlower(wint_t);
15233                 int          iswprint(wint_t);
15234                 int          iswpunct(wint_t);
15235                 int          iswspace(wint_t);
15236                 int          iswupper(wint_t);
15237                 int          iswxdigit(wint_t);
15238         size_t       mbrlen(const char *restrict, size_t, mbstate_t *restrict);
15239         size_t       mbrtowc(wchar_t *restrict, const char *restrict, size_t,
15240                             mbstate_t *restrict);
15241         int          mbsinit(const mbstate_t *);
15242     CX             size_t       mbsnrto wcs(wchar_t *restrict, const char **restrict,
15243                                     size_t, size_t, mbstate_t *restrict);
15244         size_t       mbsrtowcs(wchar_t *restrict, const char **restrict, size_t,
15245                             mbstate_t *restrict);
15246     CX             FILE         *open_wmemstream(wchar_t **, size_t *);
15247         wint_t       putwc(wchar_t, FILE *);
15248         wint_t       putwchar(wchar_t);
15249         int          swprintf(wchar_t *restrict, size_t,
15250                             const wchar_t *restrict, ...);
15251         int          swscanf(const wchar_t *restrict,
15252                             const wchar_t *restrict, ...);
15253     OB XSI         wint_t       tolower(wint_t);
15254                 wint_t       toupper(wint_t);
15255         wint_t       ungetwc(wint_t, FILE *);
15256         int          vfwprintf(FILE *restrict, const wchar_t *restrict, va_list);
15257         int          vfwscanf(FILE *restrict, const wchar_t *restrict, va_list);
15258         int          vwprintf(const wchar_t *restrict, va_list);
15259         int          vswprintf(wchar_t *restrict, size_t,
15260                             const wchar_t *restrict, va_list);
15261         int          vswscanf(const wchar_t *restrict, const wchar_t *restrict,
15262                             va_list);
15263         int          vwscanf(const wchar_t *restrict, va_list);
15264     CX             wchar_t     *wcpcpy(wchar_t restrict*, const wchar_t *restrict);
15265                 wchar_t     *wcpncpy(wchar_t restrict *, const wchar_t *restrict, size_t);
15266         size_t       wcrto mb(char *restrict, wchar_t, mbstate_t *restrict);
15267     CX             int          wcscasecmp(const wchar_t *, const wchar_t *);
15268                 int          wcscasecmp_l(const wchar_t *, const wchar_t *, locale_t);
15269         wchar_t     *wcscat(wchar_t *restrict, const wchar_t *restrict);
15270         wchar_t     *wcschr(const wchar_t *, wchar_t);
15271         int          wcscmp(const wchar_t *, const wchar_t *);
15272         int          wcscoll(const wchar_t *, const wchar_t *);
15273     CX             int          wcscoll_l(const wchar_t *, const wchar_t *, locale_t);
15274         wchar_t     *wcscpy(wchar_t *restrict, const wchar_t *restrict);
15275         size_t       wcsncpy(const wchar_t *, const wchar_t *);

```

```

15276 CX    wchar_t    *wcsdup(const wchar_t *);
15277      size_t    wcsftime(wchar_t *restrict, size_t,
15278                        const wchar_t *restrict, const struct tm *restrict);
15279      size_t    wcslen(const wchar_t *);
15280 CX    int      wcsncasecmp(const wchar_t *, const wchar_t *, size_t);
15281      int      wcsncasecmp_l(const wchar_t *, const wchar_t *, size_t,
15282                             locale_t);
15283      wchar_t  *wcsncat(wchar_t *restrict, const wchar_t *restrict, size_t);
15284      int      wcsncmp(const wchar_t *, const wchar_t *, size_t);
15285      wchar_t  *wcsncpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15286 CX    size_t    wcsnlen(const wchar_t *, size_t);
15287      size_t    wcsnrtombs(char *, const wchar_t **, size_t, size_t,
15288                           mbstate_t *);
15289      wchar_t  *wcpbrk(const wchar_t *, const wchar_t *);
15290      wchar_t  *wcsrchr(const wchar_t *, wchar_t);
15291      size_t    wcsrtombs(char *restrict, const wchar_t **restrict,
15292                          size_t, mbstate_t *restrict);
15293      size_t    wcsspncpy(const wchar_t *, const wchar_t *);
15294      wchar_t  *wcsstr(const wchar_t *restrict, const wchar_t *restrict);
15295      double   wcstod(const wchar_t *restrict, wchar_t **restrict);
15296      float    wcstof(const wchar_t *restrict, wchar_t **restrict);
15297      wchar_t  *wcstok(wchar_t *restrict, const wchar_t *restrict,
15298                       wchar_t **restrict);
15299      long     wcstol(const wchar_t *restrict, wchar_t **restrict, int);
15300      long double wcstold(const wchar_t *restrict, wchar_t **restrict);
15301      long long wcstoll(const wchar_t *restrict, wchar_t **restrict, int);
15302      unsigned long wcstoul(const wchar_t *restrict, wchar_t **restrict, int);
15303      unsigned long long wcstoull(const wchar_t *restrict, wchar_t **restrict, int);
15304      int      wcswidth(const wchar_t *, size_t);
15305 XSI    size_t    wcsxfrm(wchar_t *restrict, const wchar_t *restrict, size_t);
15306 CX    size_t    wcsxfrm_l(wchar_t *restrict, const wchar_t *restrict,
15307                           size_t, locale_t);
15308      int      wctob(wint_t);
15309      wchar_t  *wctype(const char *);
15310 OB XSI int      wctype(const char *);
15311 XSI    int      wcwidth(wchar_t);
15312      wchar_t  *wmemchr(const wchar_t *, wchar_t, size_t);
15313      int      wmemcmp(const wchar_t *, const wchar_t *, size_t);
15314      wchar_t  *wmemcpy(wchar_t *restrict, const wchar_t *restrict, size_t);
15315      wchar_t  *wmemmove(wchar_t *, const wchar_t *, size_t);
15316      wchar_t  *wmemset(wchar_t *, wchar_t, size_t);
15317      int      wprintf(const wchar_t *restrict, ...);
15318      int      wscanf(const wchar_t *restrict, ...);

```

APPLICATION USAGE

15319 The *iswblank()* function was a late addition to the ISO C standard and was introduced at the
15320 same time as the ISO C standard introduced <wctype.h>, which contains all of the *isw*()*
15321 functions. The Open Group Base Specifications had previously aligned with the MSE working
15322 draft and had introduced the rest of the *isw*()* functions into <wchar.h>. For backwards-
15323 compatibility, the original set of *isw*()* functions, without *iswblank()*, are permitted (as part of
15324 the XSI option) in <wchar.h>. For maximum portability, applications should include
15325 <wctype.h> in order to obtain declarations for the *isw*()* functions. This compatibility has been
15326 made obsolescent.
15327

15328
15329
15330
15331
15332
15333
15334
15335
15336
15337
15338
15339
15340
15341
15342
15343
15344
15345
15346
15347
15348
15349
15350
15351
15352
15353
15354
15355
15356
15357
15358
15359
15360
15361
15362
15363
15364
15365
15366
15367
15368
15369**RATIONALE**

In the ISO C standard, the symbols referenced as XSI extensions are in <wctype.h>. Their presence here is thus an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

<ctype.h>, <stdarg.h>, <stddef.h>, <stdint.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, <wctype.h>

XSH Section 2.2 (on page 448), *btowc()*, *confstr()*, *fgetwc()*, *fgetws()*, *fputwc()*, *fputws()*, *fwide()*, *fwprintf()*, *fwscanf()*, *getwc()*, *getwchar()*, *iswalnum()*, *iswalpna()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *iswctype()*, *mbsinit()*, *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *putwc()*, *putwchar()*, *towlower()*, *towupper()*, *ungetwc()*, *vwprintf()*, *vwscanf()*, *wcrtomb()*, *wcsrtombs()*, *wcscat()*, *wcschr()*, *wcscmp()*, *wscoll()*, *wscpy()*, *wscspn()*, *wcsftime()*, *wcslen()*, *wcsncat()*, *wcsncmp()*, *wcsncpy()*, *wcspbrk()*, *wcsrchr()*, *wcsspn()*, *wcsstr()*, *wcstod()*, *wcstok()*, *wcstol()*, *wcstoul()*, *wcswidth()*, *wcsxfrm()*, *wctob()*, *wctype()*, *wcwidth()*, *wmemchr()*, *wmemcmp()*, *wmemncpy()*, *wmemmove()*, *wmemset()*

XCU *getconf*

CHANGE HISTORY

First released in Issue 4.

Issue 5

Aligned with the ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

The Open Group Corrigendum U021/10 is applied. The prototypes for *wcswidth()* and *wcwidth()* are marked as extensions.

The Open Group Corrigendum U028/5 is applied, correcting the prototype for the *mbsinit()* function.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- Various function prototypes are updated to add the **restrict** keyword.
- The functions *vwscanf()*, *vwscanf()*, *wcstof()*, *wcstold()*, *wcstoll()*, and *wcstoull()* are added.

The type **wctype_t**, the *isw*()*, *to*()*, and *wctype()* functions are marked as XSI extensions.

IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/26 is applied, adding the APPLICATION USAGE section.

Issue 7

The *mbsnrtowcs()*, *open_wmemstream()*, *wcpcpy()*, *wcpncpy()*, *wcscasecmp()*, *wcsdup()*, *wcsncasecmp()*, *wcsnlen()*, and *wcsnrtombs()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The *wcscasecmp_l()*, *wcsncasecmp_l()*, *wscoll_l()*, and *wcsxfrm_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

The **wctype_t** type, and the *isw**, *towlower()*, and *towupper()* functions are marked obsolescent in <wchar.h> since the ISO C standard requires the declarations to be in <wctype.h>.

This reference page is clarified with respect to macros and symbolic constants.

15370 **NAME**

15371 wctype.h — wide-character classification and mapping utilities

15372 **SYNOPSIS**

15373 #include <wctype.h>

15374 **DESCRIPTION**

15375 CX Some of the functionality described on this reference page extends the ISO C standard.
 15376 Applications shall define the appropriate feature test macro (see XSH Section 2.2, on page 448) to
 15377 enable the visibility of these symbols in this header.

15378 The <wctype.h> header shall define the following types:

15379 **wint_t** As described in <wchar.h>.15380 **wctrans_t** A scalar type that can hold values which represent locale-specific character
15381 mappings.15382 **wctype_t** As described in <wchar.h>.15383 CX The <ctype.h> header shall provide a definition for a type **locale_t** as defined in <locale.h>.

15384 The <wctype.h> header shall define the following macro: +

15385 WEOF As described in <wchar.h>. +

15386 For all functions described in this header that accept an argument of type **wint_t**, the value is +
 15387 representable as a **wchar_t** or equals the value of WEOF. If this argument has any other value, +
 15388 the behavior is undefined. +

15389 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale. +

15390 CX Inclusion of the <wctype.h> header may make visible all symbols from the headers <ctype.h>, +
 15391 <stdarg.h>, <stddef.h>, <stdio.h>, <stdlib.h>, <string.h>, <time.h>, and <wchar.h>. +

15392 The following shall be declared as functions and may also be defined as macros. Function
 15393 prototypes shall be provided for use with ISO C standard compilers.

15394 int iswalnum(wint_t);

15395 CX int iswalnum_l(wint_t, locale_t);

15396 int iswalpha(wint_t);

15397 CX int iswalpha_l(wint_t, locale_t);

15398 int iswblank(wint_t);

15399 CX int iswblank_l(wint_t, locale_t);

15400 int iswcntrl(wint_t);

15401 CX int iswcntrl_l(wint_t, locale_t);

15402 int iswctype(wint_t, wctype_t); +

15403 CX int iswctype_l(wint_t, wctype_t, locale_t); +

15404 int iswdigit(wint_t);

15405 CX int iswdigit_l(wint_t, locale_t);

15406 int iswgraph(wint_t);

15407 CX int iswgraph_l(wint_t, locale_t);

15408 int iswlower(wint_t);

15409 CX int iswlower_l(wint_t, locale_t);

15410 int iswprint(wint_t);

15411 CX int iswprint_l(wint_t, locale_t);

15412 int iswpunct(wint_t);

15413 CX int iswpunct_l(wint_t, locale_t);

15414 int iswspace(wint_t);

<wctype.h>

Headers

```

15415 CX      int      iswspace_l(wint_t, locale_t);
15416      int      iswupper(wint_t);
15417 CX      int      iswupper_l(wint_t, locale_t);
15418      int      iswxdigit(wint_t);
15419 CX      int      iswxdigit_l(wint_t, locale_t);
15420      wint_t   towctrans(wint_t, wctrans_t);
15421 CX      wint_t   towctrans_l(wint_t, wctrans_t, locale_t);
15422      wint_t   towlower(wint_t);
15423 CX      wint_t   towlower_l(wint_t, locale_t);
15424      wint_t   towupper(wint_t);
15425 CX      wint_t   towupper_l(wint_t, locale_t);
15426      wctrans_t wctrans(const char *);
15427 CX      wctrans_t wctrans_l(const char *, locale_t);
15428      wctype_t  wctype(const char *);
15429 CX      wctype_t  wctype_l(const char *, locale_t);

```

15430 **APPLICATION USAGE**

15431 None.

15432 **RATIONALE**

15433 None.

15434 **FUTURE DIRECTIONS**

15435 None.

15436 **SEE ALSO**

15437 [<ctype.h>](#), [<locale.h>](#), [<stdarg.h>](#), [<stddef.h>](#), [<stdio.h>](#), [<stdlib.h>](#), [<string.h>](#), [<time.h>](#),
15438 [<wchar.h>](#)

15439 XSH Section 2.2 (on page 448), [iswalnum\(\)](#), [iswalphalpha\(\)](#), [iswblank\(\)](#), [iswcntrl\(\)](#), [iswctype\(\)](#),
15440 [iswdigit\(\)](#), [iswgraph\(\)](#), [iswlower\(\)](#), [iswprint\(\)](#), [iswpunct\(\)](#), [iswspace\(\)](#), [iswupper\(\)](#), [iswxdigit\(\)](#),
15441 [setlocale\(\)](#), [towctrans\(\)](#), [towlower\(\)](#), [towupper\(\)](#), [wctrans\(\)](#), [wctype\(\)](#)

15442 **CHANGE HISTORY**

15443 First released in Issue 5. Derived from the ISO/IEC 9899:1990/Amendment 1:1995 (E).

15444 **Issue 6**15445 The [iswblank\(\)](#) function is added for alignment with the ISO/IEC 9899:1999 standard.15446 **Issue 7**

15447 SD5-XBD-ERN-6 is applied.

15448 The [*_l\(\)](#) functions are added from The Open Group Technical Standard, 2006, Extended API Set
15449 Part 4.

15450 This reference page is clarified with respect to macros and symbolic constants.

15451 **NAME**
 15452 wordexp.h — word-expansion types

15453 **SYNOPSIS**
 15454 #include <wordexp.h>

15455 **DESCRIPTION**
 15456 The <wordexp.h> header shall define the structures and symbolic constants used by the
 15457 *wordexp()* and *wordfree()* functions.

15458 The structure type **wordexp_t** shall contain at least the following members:

15459 `size_t we_wordc` Count of words matched by *words*.
 15460 `char **we_wordv` Pointer to list of expanded words.
 15461 `size_t we_offs` Slots to reserve at the beginning of *we_wordv*.

15462 The <wordexp.h> header shall define the following symbolic constants for use as flags for the
 15463 *wordexp()* function:

15464 **WRDE_APPEND** Append words to those previously generated.
 15465 **WRDE_DOOFFS** Number of null pointers to prepend to *we_wordv*.
 15466 **WRDE_NOCMD** Fail if command substitution is requested.
 15467 **WRDE_REUSE** The *pwordexp* argument was passed to a previous successful call to
 15468 *wordexp()*, and has not been passed to *wordfree()*. The result is the same
 15469 as if the application had called *wordfree()* and then called *wordexp()*
 15470 without **WRDE_REUSE**.
 15471 **WRDE_SHOWERR** Do not redirect *stderr* to **/dev/null**.
 15472 **WRDE_UNDEF** Report error on an attempt to expand an undefined shell variable.

15473 The <wordexp.h> header shall define the following symbolic constants as error return values:

15474 **WRDE_BADCHAR** One of the unquoted characters—<newline>, '|', '&', ';', '<', '>',
 15475 '(', ')', '{', '}'—appears in *words* in an inappropriate context.
 15476 **WRDE_BADVAL** Reference to undefined shell variable when **WRDE_UNDEF** is set in *flags*.
 15477 **WRDE_CMDSUB** Command substitution requested when **WRDE_NOCMD** was set in *flags*.
 15478 **WRDE_NOSPACE** Attempt to allocate memory failed.
 15479 **WRDE_SYNTAX** Shell syntax error, such as unbalanced parentheses or unterminated
 15480 string.

15481 The <wordexp.h> header shall define the following type:

15482 **size_t** As described in <stddef.h>.

15483 The following shall be declared as functions and may also be defined as macros. Function
 15484 prototypes shall be provided.

15485 `int wordexp(const char *restrict, wordexp_t *restrict, int);`
 15486 `void wordfree(wordexp_t *);`

<wordexp.h>*Headers*

15487	APPLICATION USAGE		-
15488	None.		
15489	RATIONALE		
15490	None.		
15491	FUTURE DIRECTIONS		
15492	None.		
15493	SEE ALSO		
15494	<stddef.h>		
15495	XSH Section 2.6		
15496	CHANGE HISTORY		
15497	First released in Issue 4. Derived from the ISO POSIX-2 standard.		
15498	Issue 6		
15499	The restrict keyword is added to the prototype for <i>wordexp()</i> .		
15500	The WRDE_NOSYS constant is marked obsolescent.		
15501	Issue 7		
15502	The obsolescent WRDE_NOSYS constant is removed.		
15503	This reference page is clarified with respect to macros and symbolic constants.		+

15504

Technical Standard

15505

Volume 2:

15506

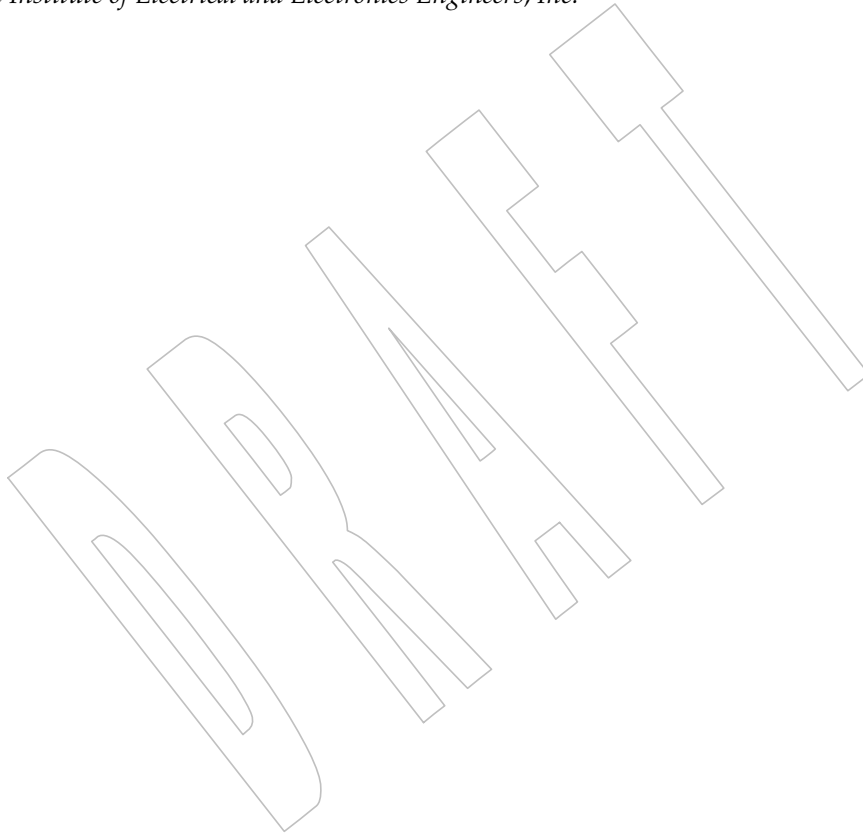
System Interfaces, Issue 7

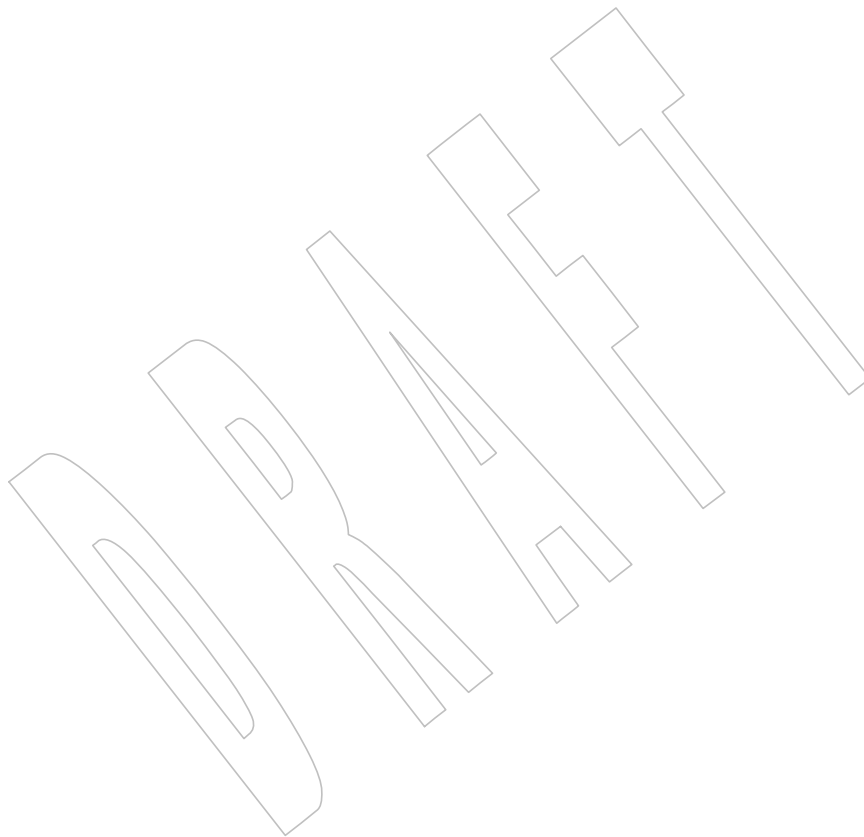
15507

The Open Group

15508

The Institute of Electrical and Electronics Engineers, Inc.





The System Interfaces volume of POSIX.1-200x describes the interfaces offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Formal Standards

Great care has been taken to ensure that this volume of POSIX.1-200x is fully aligned with the following standards:

ISO C (1999)

ISO/IEC 9899:1999, Programming Languages — C.

Parts of the ISO/IEC 9899:1999 standard (hereinafter referred to as the ISO C standard) are referenced to describe requirements also mandated by this volume of POSIX.1-200x. Some functions and headers included within this volume of POSIX.1-200x have a version in the ISO C standard; in this case CX markings are added as appropriate to show where the ISO C standard has been extended (see [Section 1.7.1](#), on page 7). Any conflict between this volume of POSIX.1-200x and the ISO C standard is unintentional.

This volume of POSIX.1-200x also allows, but does not require, mathematics functions to support IEEE Std 754-1985 and IEEE Std 854-1987.

1.2 Format of Entries

The entries in [Chapter 3](#) are based on a common format as follows. The only sections relating to conformance are the SYNOPSIS, DESCRIPTION, RETURN VALUE, and ERRORS sections.

NAME

This section gives the name or names of the entry and briefly states its purpose.

SYNOPSIS

This section summarizes the use of the entry being described. If it is necessary to include a header to use this function, the names of such headers are shown, for example:

```
#include <stdio.h>
```

DESCRIPTION

This section describes the functionality of the function or header.

RETURN VALUE

This section indicates the possible return values, if any.

If the implementation can detect errors, “successful completion” means that no error has been detected during execution of the function. If the implementation does detect an error, the error is indicated.

For functions where no errors are defined, “successful completion” means that if the

15545 implementation checks for errors, no error has been detected. If the implementation can
15546 detect errors, and an error is detected, the indicated return value is returned and *errno*
15547 may be set.

15548 **ERRORS**

15549 This section gives the symbolic names of the error values returned by a function or
15550 stored into a variable accessed through the symbol *errno* if an error occurs.

15551 “No errors are defined” means that error values returned by a function or stored into a
15552 variable accessed through the symbol *errno*, if any, depend on the implementation.

15553 **EXAMPLES**

15554 This section is informative.

15555 This section gives examples of usage, where appropriate. In the event of conflict
15556 between an example and a normative part of this volume of POSIX.1-200x, the
15557 normative material is to be taken as correct.

15558 **APPLICATION USAGE**

15559 This section is informative.

15560 This section gives warnings and advice to application developers about the entry. In the
15561 event of conflict between warnings and advice and a normative part of this volume of
15562 POSIX.1-200x, the normative material is to be taken as correct.

15563 **RATIONALE**

15564 This section is informative.

15565 This section contains historical information concerning the contents of this volume of
15566 POSIX.1-200x and why features were included or discarded by the standard
15567 developers.

15568 **FUTURE DIRECTIONS**

15569 This section is informative.

15570 This section provides comments which should be used as a guide to current thinking;
15571 there is not necessarily a commitment to adopt these future directions.

15572 **SEE ALSO**

15573 This section is informative.

15574 This section gives references to related information.

15575 **CHANGE HISTORY**

15576 This section is informative.

15577 This section shows the derivation of the entry and any significant changes that have
15578 been made to it.

15579

15580

15581

15582

This chapter covers information that is relevant to all the functions specified in [Chapter 3](#) and [XBD Chapter 13](#) (on page 205).

15583

2.1 Use and Implementation of Functions

15584

15585

Each of the following statements shall apply unless explicitly stated otherwise in the detailed descriptions that follow:

15586

15587

15588

1. If an argument to a function has an invalid value (such as a value outside the domain of the function, or a pointer outside the address space of the program, or a null pointer), the behavior is undefined.

15589

15590

15591

15592

15593

15594

15595

15596

2. Any function declared in a header may also be implemented as a macro defined in the header, so a function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is then not followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a function even if it is also defined as a macro. The use of the C-language `#undef` construct to remove any such macro definition shall also ensure that an actual function is referred to.

15597

15598

15599

15600

15601

3. Any invocation of a function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments. Likewise, those function-like macros described in the following sections may be invoked in an expression anywhere a function with a compatible return type could be called.

15602

15603

15604

4. Provided that a function can be declared without reference to any type defined in a header, it is also permissible to declare the function explicitly and use it without including its associated header.

15605

15606

5. If a function that accepts a variable number of arguments is not declared (explicitly or by including its associated header), the behavior is undefined.

15607 2.2 The Compilation Environment

15608 2.2.1 POSIX.1 Symbols

15609 Certain symbols in this volume of POSIX.1-200x are defined in headers (see XBD [Chapter 13](#), on
15610 page 205). Some of those headers could also define symbols other than those defined by
15611 POSIX.1-200x, potentially conflicting with symbols used by the application. Also, POSIX.1-200x
15612 defines symbols that are not permitted by other standards to appear in those headers without
15613 some control on the visibility of those symbols.

15614 Symbols called “feature test macros” are used to control the visibility of symbols that might be
15615 included in a header. Implementations, future versions of this standard, and other standards
15616 may define additional feature test macros.

15617 In the compilation of an application that **#defines** a feature test macro specified by
15618 POSIX.1-200x, no header defined by POSIX.1-200x shall be included prior to the definition of the
15619 feature test macro. This restriction also applies to any implementation-provided header in
15620 which these feature test macros are used. If the definition of the macro does not precede the
15621 **#include**, the result is undefined.

15622 Feature test macros shall begin with the underscore character (‘_’).

15623 2.2.1.1 The `_POSIX_C_SOURCE` Feature Test Macro

15624 A POSIX-conforming application should ensure that the feature test macro `_POSIX_C_SOURCE`
15625 is defined before inclusion of any header.

15626 When an application includes a header described by POSIX.1-200x, and when this feature test
15627 macro is defined to have the value `200xxxL`:

- 15628 1. All symbols required by POSIX.1-200x to appear when the header is included shall be
15629 made visible.
- 15630 2. Symbols that are explicitly permitted, but not required, by POSIX.1-200x to appear in that
15631 header (including those in reserved name spaces) may be made visible.
- 15632 3. Additional symbols not required or explicitly permitted by POSIX.1-200x to be in that
15633 header shall not be made visible, except when enabled by another feature test macro.

15634 Identifiers in POSIX.1-200x may only be undefined using the **#undef** directive as described in
15635 [Section 2.1](#) (on page 447) or [Section 2.2.2](#) (on page 449). These **#undef** directives shall follow all
15636 **#include** directives of any header in POSIX.1-200x.

15637 **Note:** The POSIX.1-1990 standard specified a macro called `_POSIX_SOURCE`. This has been
15638 superseded by `_POSIX_C_SOURCE`.

15639 2.2.1.2 The `_XOPEN_SOURCE` Feature Test Macro

15640 XSI An XSI-conforming application should ensure that the feature test macro `_XOPEN_SOURCE` is
15641 defined with the value `700` before inclusion of any header. This is needed to enable the
15642 functionality described in [Section 2.2.1.1](#) and to ensure that the XSI option is enabled.

15643 Since this volume of POSIX.1-200x is aligned with the ISO C standard, and since all functionality
15644 enabled by `_POSIX_C_SOURCE` set equal to `200xxxL` is enabled by `_XOPEN_SOURCE` set equal
15645 to `700`, there should be no need to define `_POSIX_C_SOURCE` if `_XOPEN_SOURCE` is so
15646 defined. Therefore, if `_XOPEN_SOURCE` is set equal to `700` and `_POSIX_C_SOURCE` is set equal
15647 to `200xxxL`, the behavior is the same as if only `_XOPEN_SOURCE` is defined and set equal to

15648 700. However, should `_POSIX_C_SOURCE` be set to a value greater than 200xxxL, the behavior
 15649 is unspecified.

15650 If `_XOPEN_SOURCE` is defined with the value 700 and `_POSIX_C_SOURCE` is undefined before
 15651 inclusion of any header, then the header may define the `_POSIX_C_SOURCE` macro with the
 15652 value 200xxxL.

15653 2.2.2 The Name Space

15654 All identifiers in this volume of POSIX.1-200x, except *environ*, are defined in at least one of the
 15655 XSI headers, as shown in XBD Chapter 13 (on page 205). When `_XOPEN_SOURCE` or
 15656 `_POSIX_C_SOURCE` is defined, each header defines or declares some identifiers, potentially
 15657 conflicting with identifiers used by the application. The set of identifiers visible to the
 15658 application consists of precisely those identifiers from the header pages of the included headers,
 15659 as well as additional identifiers reserved for the implementation. In addition, some headers may
 15660 make visible identifiers from other headers as indicated on the relevant header pages.

15661 Implementations may also add members to a structure or union without controlling the
 15662 visibility of those members with a feature test macro, as long as a user-defined macro with the
 15663 same name cannot interfere with the correct interpretation of the program. The identifiers
 15664 reserved for use by the implementation are described below:

- 15665 1. Each identifier with external linkage described in the header section is reserved for use as
 15666 an identifier with external linkage if the header is included.
- 15667 2. Each macro described in the header section is reserved for any use if the header is
 15668 included.
- 15669 3. Each identifier with file scope described in the header section is reserved for use as an
 15670 identifier with file scope in the same name space if the header is included.

15671 The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by POSIX.1-200x and other
 15672 POSIX standards. Implementations may add symbols to the headers shown in the following
 15673 table, provided the identifiers for those symbols either:

- 15674 1. Begin with the corresponding reserved prefixes in the table, or
- 15675 2. Have one of the corresponding complete names in the table, or
- 15676 3. End in the string indicated as a reserved suffix in the table and do not use the reserved
 15677 prefixes `posix_`, `POSIX_`, or `_POSIX_`, as long as the reserved suffix is in that part of the
 15678 name considered significant by the implementation.

15679 Symbols that use the reserved prefix `_POSIX_` may be made visible by implementations in any
 15680 header defined by POSIX.1-200x.

	Header	Prefix	Suffix	Complete Name
15681	<aio.h>	aio_, lio_, AIO_, LIO_		
15682	<arpa/inet.h>	in_, inet_		
15683	<ctype.h>	to[a-z], is[a-z]		
15684	<dlfcn.h>	RTLD_		
15685	<dirent.h>	d_		
15686	<fcntl.h>	l_		
15687	<fmtmsg.h>	MM_		
15688	<fnmatch.h>	FNM_		
15689	XSI			
15690	<ftw.h>	FTW		
15691	XSI			
15692	<glob.h>	gl_, GLOB_		
15693	<grp.h>	gr_		
15694	<inttypes.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
15695				
15696	<limits.h>		_MAX, _MIN	
15697	MSG	mq_, MQ_		
15698	XSI	dbm_, DBM_		
15699		ai_, h_, n_, p_, s_		
15700	<net/if.h>	if_, IF_		
15701	<netinet/in.h>	in_, ip_, s_, sin_, INADDR_, IPPROTO_		
15702	IP6	in6_, s6_, sin6_, IPV6_		
15703	<netinet/tcp.h>	TCP_		
15704	<nl_types.h>	NL_		
15705	XSI	pd_, ph_, ps_, POLL		
15706	<pthread.h>	pthread_, PTHREAD_		
15707	<pwd.h>	pw_		
15708	<regex.h>	re_, rm_, REG_		
15709	PS	sched_, SCHED_		
15710	<semaphore.h>	sem_, SEM_		
15711	<signal.h>	sa_, si_, sigev_, sival_, uc_, BUS_, CLD_, FPE_, ILL_, SA_, SEGV_, SI_, SIGEV_		
15712		ss_, sv_, SS_, TRAP_		
15713	XSI			
15714	OB XSR	POLL_		
15715	<stropts.h>	bi_, ic_, l_, sl_, str_		
15716		FLUSH[A-Z], I_, S_, SND[A-Z]		
15717	<stdint.h>			int[0-9a-z]*_t, uint[0-9a-z]*_t
15718				
15719	<stdlib.h>	str[a-z]		
15720	<string.h>	str[a-z], mem[a-z], wcs[a-z]		
15721	XSI	ipc_, IPC_		key, pad, seq
15722	<sys/mman.h>	shm_, MAP_, MCL_, MS_, PROT_		
15723				
15724	XSI	msg, MSG_[A-Z]		msg
15725	XSI	rlim_, ru_, PRIO_, RLIMIT_, RUSAGE_		
15726	<sys/select.h>	fd_, fds_, FD_		

		Header	Prefix	Suffix	Complete Name
15727		<sys/sem.h>	sem, SEM_		sem
15728		<sys/shm.h>	shm, SHM[A-Z], SHM_[A-Z]		
15729	XSI	<sys/socket.h>	cmsg_, if_, ifc_, ifra_, ifru_, infu_, l_, msg_, sa_, ss_, AF_, MSG_, PF_, SCM_, SHUT_, SO		
15730	XSI	<sys/stat.h>	st_		
15731		<sys/statvfs.h>	f_, ST_		
15732		<sys/time.h>	it_, tv_, ITIMER_		
15733	XSI	<sys/times.h>	tms_		
15734		<sys/unistd.h>	io_		UIO_MAXIOV
15735		<sys/un.h>	sun_		
15736		<sys/utsname.h>	uts_		
15737	XSI	<sys/wait.h>	si_, P_, W[A-Z]		
15738	XSI	<syslog.h>	LOG_		
15739		<termios.h>	c_, B[0-9], TC		
15740		<time.h>	tm_ clock_, it_, timer_, tv_, CLOCK_, TIMER_		
15741		<ulimit.h>	UL_		
15742	XSI	<utime.h>	utim_		
15743	XSI	<utmpx.h>	ut_	_LVL, _PROCESS, _TIME	
15744		<wchar.h>	wcs[a-z]		
15745		<wctype.h>	is[a-z], to[a-z]		
15746		<wordexp.h>	we_, WRDE_		
15747		ANY header		_t	

Note: The notation [A-Z] indicates any uppercase letter in the portable character set. The notation [a-z] indicates any lowercase letter in the portable character set. Commas and spaces in the lists of prefixes and complete names in the above table are not part of any prefix or complete name.

15759 If any header in the following table is included, macros with the prefixes shown may be defined.
 15760 After the last inclusion of a given header, an application may use identifiers with the
 15761 corresponding prefixes for its own purpose, provided their use is preceded by a **#undef** of the
 15762 corresponding macro.

Header	Prefix
<errno.h>	E[0-9], E[A-Z]
<fcntl.h>	F_, O_, S_, SEEK_
<fenv.h>	FE_[A-Z]
<inttypes.h>	PRI[Xa-z], SCN[Xa-z]
<locale.h>	LC_[A-Z]
<math.h>	FP_[A-Z]
<netinet/in.h>	IMPLINK_, IN_, IP_, IPPORT_, SOCK_
IP6	IN6_
<signal.h>	SIG_, SIG[A-Z],
XSI	SV_
CX	<stdio.h> SEEK_
OB XSR	<stropts.h> M_, MUXID_R[A-Z], STR
XSI	<sys/resource.h> RLIM_
XSI	<sys/socket.h> CMMSG_
	<sys/stat.h> S_
XSI	<sys/uio.h> IOV_
XSI	<sys/wait.h> BUS_, CLD_, FPE_, ILL_, POLL_, SEGV_, SI_, TRAP_
	<termios.h> I, O, V (See below.)
	<unistd.h> SEEK_

15783 The following are used to reserve complete names for the **<stdint.h>** header:

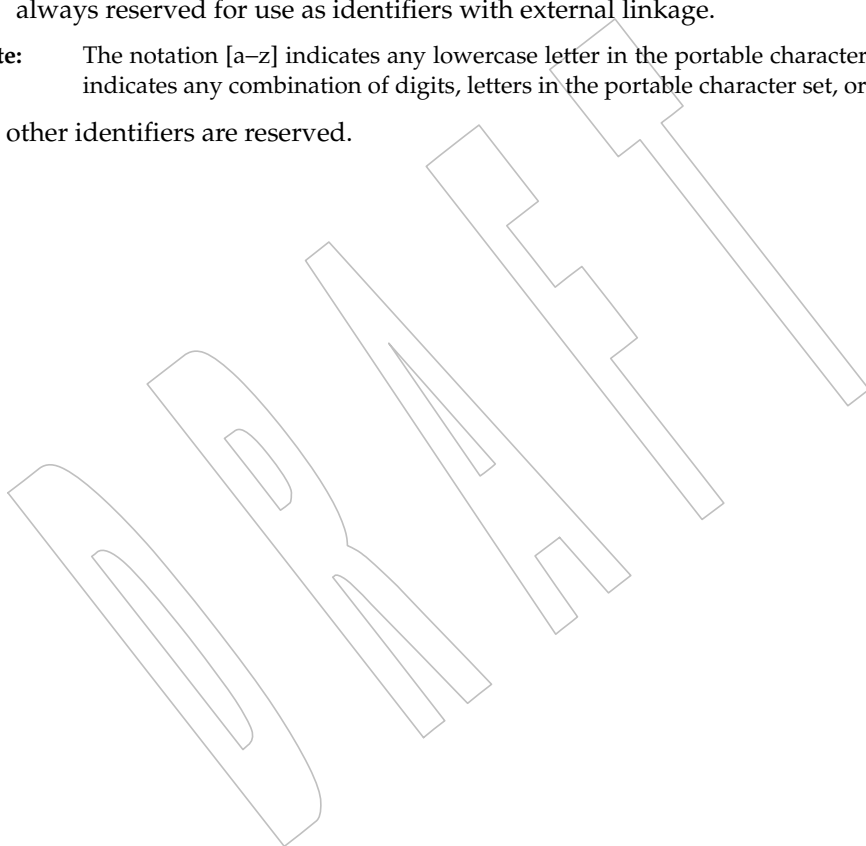
15784 INT[0-9A-Za-z_]*_MIN
 15785 INT[0-9A-Za-z_]*_MAX
 15786 INT[0-9A-Za-z_]*_C
 15787 UINT[0-9A-Za-z_]*_MIN
 15788 UINT[0-9A-Za-z_]*_MAX
 15789 UINT[0-9A-Za-z_]*_C

15790 **Note:** The notation [0–9] indicates any digit. The notation [A–Z] indicates any uppercase letter in the
 15791 portable character set. The notation [0–9a–z_] indicates any digit, any lowercase letter in the
 15792 portable character set, or underscore.

15793 XSI The following reserved names are used as exact matches for **<termios.h>**:

15794	CBAUD	EXTB	VDSUSP
15795	DEFECHO	FLUSHO	VLNEXT
15796	ECHOCTL	LOBLK	VREPRINT
15797	ECHOKE	PENDIN	VSTATUS
15798	ECHOPRT	SWTCH	VWERASE
15799	EXTA	VDISCARD	

- 15800 The following identifiers are reserved regardless of the inclusion of headers:
- 15801 1. With the exception of identifiers beginning with the prefix `_POSIX_`, all identifiers that
- 15802 begin with an underscore and either an uppercase letter or another underscore are always
- 15803 reserved for any use by the implementation.
- 15804 2. All identifiers that begin with an underscore are always reserved for use as identifiers with
- 15805 file scope in both the ordinary identifier and tag name spaces.
- 15806 3. All identifiers in the table below are reserved for use as identifiers with external linkage.
- 15807 Some of these identifiers do not appear in this volume of POSIX.1-200x, but are reserved for
- 15808 future use by the ISO C standard.
- 15809 4. All functions and external identifiers defined in XBD [Chapter 13](#) (on page 205) are reserved +
- 15810 for use as identifiers with external linkage. +
- 15811 5. All the identifiers defined in this volume of POSIX.1-200x that have external linkage are +
- 15812 always reserved for use as identifiers with external linkage. +
- 15813 **Note:** The notation `[a-z]` indicates any lowercase letter in the portable character set. The notation `'*'` +
- 15814 indicates any combination of digits, letters in the portable character set, or underscore. +
- 15815 No other identifiers are reserved. +



15816	_Exit	casinh	clog10	erfcl	fminl
15817	abort	casinhf	clog10f	erff	fmod
15818	abs	casinhl	clog10l	erfl	fmodf
15819	acos	casinl	clog1p	errno	fmodl
15820	acosf	catan	clog1pf	exit	fopen
15821	acosh	catanf	clog1pl	exp	fprintf
15822	acoshf	catanh	clog2	exp2	fputc
15823	acoshl	catanh	clog2f	exp2f	fputs
15824	acosl	catanhf	clog2l	exp2l	fputwc
15825	acosl	catanhf	clogf	expf	fputws
15826	asctime	catanhl	clogl	expl	fread
15827	asin	catanhl	conj	expm1	free
15828	asinf	catanl	conjf	expm1f	freopen
15829	asinh	cbirt	conjl	expm1l	frexp
15830	asinhf	cbirtf	copysign	fabs	frexpf
15831	asinhhl	cbirtl	copysignf	fabsf	frexpl
15832	asinl	ccos	copysignl	fabsl	fscanf
15833	asinl	ccosf	cos	fclose	fseek
15834	atan	ccosh	cosf	fdim	fsetpos
15835	atan2	ccoshf	cosh	fdimf	ftell
15836	atan2f	ccoshl	coshf	fdiml	fwide
15837	atan2l	ccosl	coshl	feclearexcept	fwprintf
15838	atanf	ceil	cosl	fegetenv	fwrite
15839	atanf	ceilf	cpow	fegetexceptflag	fwscanf
15840	atanh	ceilf	cpowf	fegetround	getc
15841	atanh	ceill	cpowl	feholdexcept	getchar
15842	atanhf	ceill	cproj	feof	getenv
15843	atanhl	cerf	cprojf	feraiseexcept	gets
15844	atanl	cerfc	cprojl	ferror	getwc
15845	atanl	cerfcf	creal	fesetenv	getwchar
15846	atexit	cerfcf	crealf	fesetexceptflag	gmtime
15847	atof	cerff	creall	fesetround	hypotf
15848	atoi	cerfl	csin	fetestexcept	hypotl
15849	atol	cexmp1	csinf	feupdateenv	ilogb
15850	atoll	cexmp1f	csinh	fflush	ilogbf
15851	bsearch	cexmp1l	csinhf	fgetc	ilogbl
15852	cabs	cexp	csinhhl	fgetpos	imaxabs
15853	cabsf	cexp2	csinl	fgets	imaxdiv
15854	cabsl	cexp2f	csqrt	fgetwc	is[a-z]*
15855	cacos	cexp2l	csqrtf	fgetws	isblank
15856	cacosf	cexpf	csqrtl	floor	iswblank
15857	cacosh	cexpl	ctan	floorf	labs
15858	cacoshf	cimag	ctanf	floorl	ldexp
15859	cacoshl	cimagf	ctanl	fma	ldexpf
15860	cacosl	cimagl	ctgamma	fmaf	ldexpl
15861	calloc	clearerr	ctgammaf	fmal	ldiv
15862	carg	clgamma	ctgamma	fmax	ldiv
15863	cargf	clgammaf	ltime	fmaxf	lgammaf
15864	cargl	clgamma	difftime	fmaxl	lgammal
15865	casin	clock	div	fmin	llabs
15866	casinf	clog	erfcf	fminf	llrint

15867	llrintf	mbsinit	realloc	sprintf	ungetwc
15868	llrintl	mbsrtowcs	remainderf	sqrt	va_end
15869	llround	mbstowcs	remainderl	sqrtrf	vfprintf
15870	llroundf	mbtowlc	remove	sqrtrl	vfscanf
15871	llroundl	mem[a-z]*	remquo	srand	vwprintf
15872	localeconv	mktime	remquof	sscanf	vwscanf
15873	localtime	modf	remquol	str[a-z]*	vprintf
15874	log	modff	rename	strtof	vscanf
15875	log10	modfl	rewind	strtoimax	vsprintf
15876	log10f	nan	rint	strtold	vsscanf
15877	log10l	nanf	rintf	strtoll	vswprintf
15878	log1p	nanl	rintl	strtoull	vswscanf
15879	log1pf	nearbyint	round	strtoumax	vwprintf
15880	log1pl	nearbyintf	roundf	swprintf	vwscanf
15881	log2	nearbyintl	roundl	swscanf	wcrtomb
15882	log2f	nextafterf	scalbln	system	wcs[a-z]*
15883	log2l	nextafterl	scalblnf	tan	wcstof
15884	logb	nexttoward	scalblnl	tanf	wcstoimax
15885	logbf	nexttowardf	scalbn	tanh	wcstold
15886	logbl	nexttowardl	scalbnf	tanhf	wcstoll
15887	logf	perror	scalbnl	tanhf	wcstoull
15888	logl	pow	scanf	tanl	wcstoumax
15889	longjmp	powf	setbuf	tgamma	wctob
15890	lrint	powl	setjmp	tgammaf	wctomb
15891	lrintf	printf	setlocale	tgammaf	wctrans
15892	lrintl	putc	setvbuf	time	wctype
15893	lround	putchar	signal	tmpfile	wcwidth
15894	lroundf	puts	sin	tmpnam	wmem[a-z]*
15895	lroundl	putwc	sinf	to[a-z]*	wprintf
15896	malloc	putwchar	sinh	trunc	wscanf
15897	mblen	qsort	sinhf	truncf	
15898	mbrlen	raise	sinhl	truncl	
15899	mbrtowc	rand	sinl	ungetc	

15900 Applications shall not declare or define identifiers with the same name as an identifier reserved -
 15901 in the same context. Since macro names are replaced whenever found, independent of scope and
 15902 name space, macro names matching any of the reserved identifier names shall not be defined by
 15903 an application if any associated header is included.

15904 Except that the effect of each inclusion of `<assert.h>` depends on the definition of `NDEBUG`,
 15905 headers may be included in any order, and each may be included more than once in a given
 15906 scope, with no difference in effect from that of being included only once.

15907 If used, the application shall ensure that a header is included outside of any external declaration
 15908 or definition, and it shall be first included before the first reference to any type or macro it
 15909 defines, or to any function or object it declares. However, if an identifier is declared or defined in
 15910 more than one header, the second and subsequent associated headers may be included after the
 15911 initial reference to the identifier. Prior to the inclusion of a header, the application shall not
 15912 define any macros with names lexically identical to symbols defined by that header.

2.3 Error Numbers

15913

15914

15915

Most functions can provide an error number. The means by which each function provides its error numbers is specified in its description.

15916

15917

15918

15919

15920

Some functions provide the error number in a variable accessed through the symbol *errno*, defined by including the `<errno.h>` header. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. No function in this volume of POSIX.1-200x shall set *errno* to zero. For each thread of a process, the value of *errno* shall not be affected by function calls or assignments to *errno* by other threads.

15921

15922

Some functions return an error number directly as the function value. These functions return a value of zero to indicate success.

15923

15924

If more than one error occurs in processing a function call, any one of the possible errors may be returned, as the order of detection is undefined.

15925

15926

15927

15928

15929

15930

15931

Implementations may support additional errors not included in this list, may generate errors included in this list under circumstances other than those described here, or may contain extensions or limitations that prevent some errors from occurring. The ERRORS section on each reference page specifies whether an error shall be returned, or whether it may be returned. Implementations shall not generate a different error number from the ones described here for error conditions described in this volume of POSIX.1-200x, but may generate additional errors unless explicitly disallowed for a particular function.

15932

15933

15934

Each implementation shall document, in the conformance document, situations in which each of the optional conditions defined in POSIX.1-200x is detected. The conformance document may also contain statements that one or more of the optional error conditions are not detected.

15935

15936

Certain threads-related functions are not allowed to return an error code of [EINTR]. Where this applies it is stated in the ERRORS section on the individual function pages.

15937

15938

15939

15940

15941

15942

15943

The following macro names identify the possible error numbers, in the context of the functions specifically defined in this volume of POSIX.1-200x; these general descriptions are more precisely defined in the ERRORS sections of the functions that return them. Only these macro names should be used in programs, since the actual value of the error number is unspecified. All values listed in this section shall be unique, except as noted below. The values for all these macros shall be found in the `<errno.h>` header defined in the Base Definitions volume of POSIX.1-200x. The actual values are unspecified by this volume of POSIX.1-200x.

15944

15945

15946

15947

[E2BIG]

Argument list too long. The sum of the number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

15948

or:

15949

Lack of space in an output buffer.

15950

or:

15951

Argument is greater than the system-imposed maximum.

15952

15953

15954

[EACCES]

Permission denied. An attempt was made to access a file in a way forbidden by its file access permissions.

15955

15956

[EADDRINUSE]

Address in use. The specified address is in use.

15957		[EADDRNOTAVAIL]
15958		Address not available. The specified address is not available from the local system.
15959		[EAFNOSUPPORT]
15960		Address family not supported. The implementation does not support the specified address
15961		family, or the specified address is not a valid address for the address family of the specified
15962		socket.
15963		[EAGAIN]
15964		Resource temporarily unavailable. This is a temporary condition and later calls to the same
15965		routine may complete normally.
15966		[EALREADY]
15967		Connection already in progress. A connection request is already in progress for the specified
15968		socket.
15969		[EBADF]
15970		Bad file descriptor. A file descriptor argument is out of range, refers to no open file, or a
15971		read (write) request is made to a file that is only open for writing (reading).
15972		[EBADMSG]
15973	OB XSR	Bad message. During a <i>read()</i> , <i>getmsg()</i> , <i>getpmsg()</i> , or <i>ioctl()</i> I_RECVFD request to a
15974		STREAMS device, a message arrived at the head of the STREAM that is inappropriate for
15975		the function receiving the message.
15976		<i>read()</i> Message waiting to be read on a STREAM is not a data message.
15977		<i>getmsg()</i> or <i>getpmsg()</i>
15978		A file descriptor was received instead of a control message.
15979		<i>ioctl()</i> Control or data information was received instead of a file descriptor when
15980		I_RECVFD was specified.
15981		or:
15982		Bad Message. The implementation has detected a corrupted message.
15983		[EBUSY]
15984		Resource busy. An attempt was made to make use of a system resource that is not currently
15985		available, as it is being used by another process in a manner that would have conflicted
15986		with the request being made by this process.
15987		[ECANCELED]
15988		Operation canceled. The associated asynchronous operation was canceled before
15989		completion.
15990		[ECHILD]
15991		No child process. A <i>wait()</i> , <i>waitid()</i> , or <i>waitpid()</i> function was executed by a process that
15992		had no existing or unwaited-for child process.
15993		[ECONNABORTED]
15994		Connection aborted. The connection has been aborted.
15995		[ECONNREFUSED]
15996		Connection refused. An attempt to connect to a socket was refused because there was no
15997		process listening or because the queue of connection requests was full and the underlying
15998		protocol does not support retransmissions.
15999		[ECONNRESET]
16000		Connection reset. The connection was forcibly closed by the peer.

16001	[EDEADLK]
16002	Resource deadlock would occur. An attempt was made to lock a system resource that would
16003	have resulted in a deadlock situation.
16004	[EDESTADDRREQ]
16005	Destination address required. No bind address was established.
16006	[EDOM]
16007	Domain error. An input argument is outside the defined domain of the mathematical
16008	function (defined in the ISO C standard).
16009	[EDQUOT]
16010	Reserved.
16011	[EEXIST]
16012	File exists. An existing file was mentioned in an inappropriate context; for example, as a
16013	new link name in the <i>link()</i> function.
16014	[EFAULT]
16015	Bad address. The system detected an invalid address in attempting to use an argument of a
16016	call. The reliable detection of this error cannot be guaranteed, and when not detected may
16017	result in the generation of a signal, indicating an address violation, which is sent to the
16018	process.
16019	[EFBIG]
16020	File too large. The size of a file would exceed the maximum file size of an implementation
16021	or offset maximum established in the corresponding file description.
16022	[EHOSTUNREACH]
16023	Host is unreachable. The destination host cannot be reached (probably because the host is
16024	down or a remote router cannot reach it).
16025	[EIDRM]
16026	Identifier removed. Returned during XSI interprocess communication if an identifier has
16027	been removed from the system.
16028	[EILSEQ]
16029	Illegal byte sequence. A wide-character code has been detected that does not correspond to
16030	a valid character, or a byte sequence does not form a valid wide-character code (defined in
16031	the ISO C standard).
16032	[EINPROGRESS]
16033	Operation in progress. This code is used to indicate that an asynchronous operation has not
16034	yet completed.
16035	or:
16036	O_NONBLOCK is set for the socket file descriptor and the connection cannot be
16037	immediately established.
16038	[EINTR]
16039	Interrupted function call. An asynchronous signal was caught by the process during the
16040	execution of an interruptible function. If the signal handler performs a normal return, the
16041	interrupted function call may return this condition (see the Base Definitions volume of
16042	POSIX.1-200x, <signal.h>).
16043	[EINVAL]
16044	Invalid argument. Some invalid argument was supplied; for example, specifying an
16045	undefined signal in a <i>signal()</i> function or a <i>kill()</i> function.

16046		[EIO]
16047		Input/output error. Some physical input or output error has occurred. This error may be
16048		reported on a subsequent operation on the same file descriptor. Any other error-causing
16049		operation on the same file descriptor may cause the [EIO] error indication to be lost.
16050		[EISCONN]
16051		Socket is connected. The specified socket is already connected.
16052		[EISDIR]
16053		Is a directory. An attempt was made to open a directory with write mode specified.
16054		[ELOOP]
16055		Symbolic link loop. A loop exists in symbolic links encountered during pathname
16056		resolution. This error may also be returned if more than {SYMLOOP_MAX} symbolic links
16057		are encountered during pathname resolution.
16058		[EMFILE]
16059		File descriptor value too large. An attempt was made to open a file descriptor with a value
16060	XSI	greater than or equal to {OPEN_MAX}, or greater than or equal to the soft limit
16061		RLIMIT_NOFILE for the process (if smaller than {OPEN_MAX}).
16062		[EMLINK]
16063		Too many links. An attempt was made to have the link count of a single file exceed
16064		{LINK_MAX}.
16065		[EMSGSIZE]
16066		Message too large. A message sent on a transport provider was larger than an internal
16067		message buffer or some other network limit.
16068		or:
16069		Inappropriate message buffer length.
16070		[EMULTIHOP]
16071		◁Reserved.
16072		[ENAMETOOLONG]
16073		Filename too long. The length of a pathname exceeds {PATH_MAX}, or a pathname
16074		component is longer than {NAME_MAX}. This error may also occur when pathname
16075		substitution, as a result of encountering a symbolic link during pathname resolution, results
16076		in a pathname string the size of which exceeds {PATH_MAX}.
16077		[ENETDOWN]
16078		Network is down. The local network interface used to reach the destination is down.
16079		[ENETRESET]
16080		The connection was aborted by the network.
16081		[ENETUNREACH]
16082		Network unreachable. No route to the network is present.
16083		[ENFILE]
16084		Too many files open in system. Too many files are currently open in the system. The system
16085		has reached its predefined limit for simultaneously open files and temporarily cannot
16086		accept requests to open another one.
16087		[ENOBUFS]
16088		No buffer space available. Insufficient buffer resources were available in the system to
16089		perform the socket operation.

16090	OB XSR	[ENODATA]	
16091			No message available. No message is available on the STREAM head read queue.
16092		[ENODEV]	
16093			No such device. An attempt was made to apply an inappropriate function to a device; for example, trying to read a write-only device such as a printer.
16094			
16095		[ENOENT]	
16096			No such file or directory. A component of a specified pathname does not exist, or the pathname is an empty string.
16097			
16098		[ENOEXEC]	
16099			Executable file format error. A request is made to execute a file that, although it has the appropriate permissions, is not in the format required by the implementation for executable files.
16100			
16101			
16102		[ENOLCK]	
16103			No locks available. A system-imposed limit on the number of simultaneous file and record locks has been reached and no more are currently available.
16104			
16105		[ENOLINK]	
16106			Reserved.
16107		[ENOMEM]	
16108			Not enough space. The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.
16109			
16110		[ENOMSG]	
16111			No message of the desired type. The message queue does not contain a message of the required type during XSI interprocess communication.
16112			
16113		[ENOPROTOOPT]	
16114			Protocol not available. The protocol option specified to <i>setsockopt()</i> is not supported by the implementation.
16115			
16116		[ENOSPC]	
16117			No space left on a device. During the <i>write()</i> function on a regular file or when extending a directory, there is no free space left on the device.
16118			
16119	OB XSR	[ENOSR]	
16120			No STREAM resources. Insufficient STREAMS memory resources are available to perform a STREAMS-related function. This is a temporary condition; it may be recovered from if other processes release resources.
16121			
16122			
16123	OB XSR	[ENOSTR]	
16124			Not a STREAM. A STREAM function was attempted on a file descriptor that was not associated with a STREAMS device.
16125			
16126		[ENOSYS]	
16127			Function not implemented. An attempt was made to use a function that is not available in this implementation.
16128			
16129		[ENOTCONN]	
16130			Socket not connected. The socket is not connected.
16131		[ENOTDIR]	
16132			Not a directory. A component of the specified pathname exists, but it is not a directory, when a directory was expected.
16133			

16134	[ENOTEMPTY]
16135	Directory not empty. A directory other than an empty directory was supplied when an
16136	empty directory was expected.
16137	[ENOTSOCK]
16138	Not a socket. The file descriptor does not refer to a socket.
16139	[ENOTSUP]
16140	Not supported. The implementation does not support this feature of the Realtime Option
16141	Group.
16142	[ENOTTY]
16143	Inappropriate I/O control operation. A control function has been attempted for a file or
16144	special file for which the operation is inappropriate.
16145	[ENXIO]
16146	No such device or address. Input or output on a special file refers to a device that does not
16147	exist, or makes a request beyond the capabilities of the device. It may also occur when, for
16148	example, a tape drive is not on-line.
16149	[EOPNOTSUPP]
16150	Operation not supported on socket. The type of socket (address family or protocol) does not
16151	support the requested operation. A conforming implementation may assign the same values
16152	for [EOPNOTSUP] and [ENOTSUP].
16153	[EOVERFLOW]
16154	Value too large to be stored in data type. An operation was attempted which would
16155	generate a value that is outside the range of values that can be represented in the relevant
16156	data type or that are allowed for a given data item.
16157	[EPERM]
16158	Operation not permitted. An attempt was made to perform an operation limited to
16159	processes with appropriate privileges or to the owner of a file or other resource.
16160	[EPIPE]
16161	Broken pipe. A write was attempted on a socket, pipe, or FIFO for which there is no process
16162	to read the data.
16163	[EPROTO]
16164	Protocol error. Some protocol error occurred. This error is device-specific, but is generally
16165	not related to a hardware failure.
16166	[EPROTONOSUPPORT]
16167	Protocol not supported. The protocol is not supported by the address family, or the protocol
16168	is not supported by the implementation.
16169	[EPROTOTYPE]
16170	Protocol wrong type for socket. The socket type is not supported by the protocol.
16171	[ERANGE]
16172	Result too large or too small. The result of the function is too large (overflow) or too small
16173	(underflow) to be represented in the available space (defined in the ISO C standard).
16174	[EROFS]
16175	Read-only file system. An attempt was made to modify a file or directory on a file system
16176	that is read-only.
16177	[ESPIPE]
16178	Invalid seek. An attempt was made to access the file offset associated with a pipe or FIFO.

16179		[ESRCH]
16180		No such process. No process can be found corresponding to that specified by the given
16181		process ID.
16182		[ESTALE]
16183		Reserved.
16184	OB XSR	[ETIME]
16185		STREAM <i>ioctl()</i> timeout. The timer set for a STREAMS <i>ioctl()</i> call has expired. The cause of
16186		this error is device-specific and could indicate either a hardware or software failure, or a
16187		timeout value that is too short for the specific operation. The status of the <i>ioctl()</i> operation is
16188		unspecified.
16189		[ETIMEDOUT]
16190		Connection timed out. The connection to a remote machine has timed out. If the connection
16191		timed out during execution of the function that reported this error (as opposed to timing
16192		out prior to the function being called), it is unspecified whether the function has completed
16193		some or all of the documented behavior associated with a successful completion of the
16194		function.
16195		or:
16196		Operation timed out. The time limit associated with the operation was exceeded before the
16197		operation completed.
16198		[ETXTBSY]
16199		Text file busy. An attempt was made to execute a pure-procedure program that is currently
16200		open for writing, or an attempt has been made to open for writing a pure-procedure
16201		program that is being executed.
16202		[EWOULDBLOCK]
16203		Operation would block. An operation on a socket marked as non-blocking has encountered
16204		a situation such as no data available that otherwise would have caused the function to
16205		<suspend execution.
16206		A conforming implementation may assign the same values for [EWOULDBLOCK] and
16207		[EAGAIN].
16208		[EXDEV]
16209		Improper link. A link to a file on another file system was attempted.

2.3.1 Additional Error Numbers

16210 Additional implementation-defined error numbers may be defined in `<errno.h>`.

16211

16212 2.4 Signal Concepts

16213 2.4.1 Signal Generation and Delivery

16214 A signal is said to be “generated” for (or sent to) a process or thread when the event that causes
 16215 the signal first occurs. Examples of such events include detection of hardware faults, timer
 16216 expiration, signals generated via the **sigevent** structure and terminal activity, as well as
 16217 invocations of the *kill()* and *sigqueue()* functions. In some circumstances, the same event
 16218 generates signals for multiple processes.

16219 At the time of generation, a determination shall be made whether the signal has been generated
 16220 for the process or for a specific thread within the process. Signals which are generated by some
 16221 action attributable to a particular thread, such as a hardware fault, shall be generated for the
 16222 thread that caused the signal to be generated. Signals that are generated in association with a
 16223 process ID or process group ID or an asynchronous event, such as terminal activity, shall be
 16224 generated for the process.

16225 Each process has an action to be taken in response to each signal defined by the system (see
 16226 [Section 2.4.3](#), on page 465). A signal is said to be “delivered” to a process when the appropriate
 16227 action for the process and signal is taken. A signal is said to be “accepted” by a process when the
 16228 signal is selected and returned by one of the *sigwait()* functions.

16229 During the time between the generation of a signal and its delivery or acceptance, the signal is
 16230 said to be “pending”. Ordinarily, this interval cannot be detected by an application. However, a
 16231 signal can be “blocked” from delivery to a thread. If the action associated with a blocked signal
 16232 is anything other than to ignore the signal, and if that signal is generated for the thread, the
 16233 signal shall remain pending until it is unblocked, it is accepted when it is selected and returned
 16234 by a call to the *sigwait()* function, or the action associated with it is set to ignore the signal.
 16235 Signals generated for the process shall be delivered to exactly one of those threads within the
 16236 process which is in a call to a *sigwait()* function selecting that signal or has not blocked delivery
 16237 of the signal. If there are no threads in a call to a *sigwait()* function selecting that signal, and if all
 16238 threads within the process block delivery of the signal, the signal shall remain pending on the
 16239 process until a thread calls a *sigwait()* function selecting that signal, a thread unblocks delivery
 16240 of the signal, or the action associated with the signal is set to ignore the signal. If the action
 16241 associated with a blocked signal is to ignore the signal and if that signal is generated for the
 16242 process, it is unspecified whether the signal is discarded immediately upon generation or
 16243 remains pending.

16244 Each thread has a “signal mask” that defines the set of signals currently blocked from delivery
 16245 to it. The signal mask for a thread shall be initialized from that of its parent or creating thread,
 16246 or from the corresponding thread in the parent process if the thread was created as the result of a
 16247 call to *fork()*. The *pthread_sigmask()*, *sigaction()*, *sigprocmask()*, and *sigsuspend()* functions control
 16248 the manipulation of the signal mask.

16249 The determination of which action is taken in response to a signal is made at the time the signal
 16250 is delivered, allowing for any changes since the time of generation. This determination is
 16251 independent of the means by which the signal was originally generated. If a subsequent
 16252 occurrence of a pending signal is generated, it is implementation-defined as to whether the
 16253 signal is delivered or accepted more than once in circumstances other than those in which
 16254 queuing is required. The order in which multiple, simultaneously pending signals outside the
 16255 range SIGRTMIN to SIGRTMAX are delivered to or accepted by a process is unspecified.

16256 When any stop signal (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is generated for a process, any
 16257 pending SIGCONT signals for that process shall be discarded. Conversely, when SIGCONT is

16258 generated for a process, all pending stop signals for that process shall be discarded. When
 16259 SIGCONT is generated for a process that is stopped, the process shall be continued, even if the
 16260 SIGCONT signal is blocked or ignored. If SIGCONT is blocked and not ignored, it shall remain
 16261 pending until it is either unblocked or a stop signal is generated for the process.

16262 An implementation shall document any condition not specified by this volume of POSIX.1-200x
 16263 under which the implementation generates signals.

16264 2.4.2 Realtime Signal Generation and Delivery

16265 This section describes functionality to support realtime signal generation and delivery.

16266 Some signal-generating functions, such as high-resolution timer expiration, asynchronous I/O
 16267 completion, interprocess message arrival, and the *sigqueue()* function, support the specification
 16268 of an application-defined value, either explicitly as a parameter to the function or in a **sigevent**
 16269 structure parameter. The **sigevent** structure is defined in **<signal.h>** and contains at least the
 16270 following members:

Member Type	Member Name	Description
int	<i>sigev_notify</i>	Notification type.
int	<i>sigev_signo</i>	Signal number.
union signal	<i>sigev_value</i>	Signal value.
void(*) (union signal)	<i>sigev_notify_function</i>	Notification function.
(pthread_attr_t*)	<i>sigev_notify_attributes</i>	Notification attributes.

16271 The *sigev_notify* member specifies the notification mechanism to use when an asynchronous
 16272 event occurs. This volume of POSIX.1-200x defines the following values for the *sigev_notify*
 16273 member:
 16274

16280 SIGEV_NONE No asynchronous notification shall be delivered when the event of
 16281 interest occurs.

16282 SIGEV_SIGNAL The signal specified in *sigev_signo* shall be generated for the process when
 16283 the event of interest occurs. If the implementation supports the Realtime
 16284 Signals Extension option and if the SA_SIGINFO flag is set for that signal
 16285 number, then the signal shall be queued to the process and the value
 16286 specified in *sigev_value* shall be the *si_value* component of the generated
 16287 signal. If SA_SIGINFO is not set for that signal number, it is unspecified
 16288 whether the signal is queued and what value, if any, is sent.

16289 SIGEV_THREAD A notification function shall be called to perform notification.

16290 An implementation may define additional notification mechanisms.

16291 The *sigev_signo* member specifies the signal to be generated. The *sigev_value* member is the
 16292 application-defined value to be passed to the signal-catching function at the time of the signal
 16293 delivery or to be returned at signal acceptance as the *si_value* member of the **siginfo_t** structure.

16294 The **signal** union is defined in **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	<i>sival_int</i>	Integer signal value.
void*	<i>sival_ptr</i>	Pointer signal value.

16298 The *sival_int* member shall be used when the application-defined value is of type **int**; the
 16299 *sival_ptr* member shall be used when the application-defined value is a pointer.

16300 When a signal is generated by the *sigqueue()* function or any signal-generating function that

16301 supports the specification of an application-defined value, the signal shall be marked pending
 16302 and, if the SA_SIGINFO flag is set for that signal, the signal shall be queued to the process along
 16303 with the application-specified signal value. Multiple occurrences of signals so generated are
 16304 queued in FIFO order. It is unspecified whether signals so generated are queued when the
 16305 SA_SIGINFO flag is not set for that signal.

16306 Signals generated by the *kill()* function or other events that cause signals to occur, such as
 16307 detection of hardware faults, *alarm()* timer expiration, or terminal activity, and for which the
 16308 implementation does not support queuing, shall have no effect on signals already queued for the
 16309 same signal number.

16310 When multiple unblocked signals, all in the range SIGRTMIN to SIGRTMAX, are pending, the
 16311 behavior shall be as if the implementation delivers the pending unblocked signal with the
 16312 lowest signal number within that range. No other ordering of signal delivery is specified.

16313 If, when a pending signal is delivered, there are additional signals queued to that signal number,
 16314 the signal shall remain pending. Otherwise, the pending indication shall be reset.

16315 Multi-threaded programs can use an alternate event notification mechanism. When a
 16316 notification is processed, and the *sigev_notify* member of the **sigevent** structure has the value
 16317 SIGEV_THREAD, the function *sigev_notify_function* is called with parameter *sigev_value*.

16318 The function shall be executed in an environment as if it were the *start_routine* for a newly
 16319 created thread with thread attributes specified by *sigev_notify_attributes*. If *sigev_notify_attributes*
 16320 is NULL, the behavior shall be as if the thread were created with the *detachstate* attribute set to
 16321 PTHREAD_CREATE_DETACHED. Supplying an attributes structure with a *detachstate* attribute
 16322 of PTHREAD_CREATE_JOINABLE results in undefined behavior. The signal mask of this
 16323 thread is implementation-defined.

16324 2.4.3 Signal Actions

16325 There are three types of action that can be associated with a signal: SIG_DFL, SIG_IGN, or a
 16326 pointer to a function. Initially, all signals shall be set to SIG_DFL or SIG_IGN prior to entry of
 16327 the *main()* routine (see the *exec* functions). The actions prescribed by these values are as follows.

16328 SIG_DFL

16329 Signal-specific default action.

16330 The default actions for the signals defined in this volume of POSIX.1-200x are specified under
 16331 **<signal.h>**. The default actions for the realtime signals in the range SIGRTMIN to SIGRTMAX
 16332 shall be to terminate the process abnormally.

16333 If the default action is to terminate the process abnormally, the process is terminated as if by a +
 16334 call to *_exit()*, except that the status made available to *wait()*, *waitid()*, and *waitpid()* indicates +
 16335 abnormal termination by the signal. If the default action is to terminate the process abnormally +
 16336 with additional actions, implementation-defined abnormal termination actions, such as creation +
 16337 of a core file, may also occur. +

16338 If the default action is to stop the process, the execution of that process is temporarily
 16339 suspended. When a process stops, a SIGCHLD signal shall be generated for its parent process,
 16340 unless the parent process has set the SA_NOCLDSTOP flag. While a process is stopped, any
 16341 additional signals that are sent to the process shall not be delivered until the process is
 16342 continued, except SIGKILL which always terminates the receiving process. A process that is a
 16343 member of an orphaned process group shall not be allowed to stop in response to the SIGTSTP,
 16344 SIGTTIN, or SIGTTOU signals. In cases where delivery of one of these signals would stop such a
 16345 process, the signal shall be discarded.

16346 If the default action is to ignore the signal, delivery of the signal shall have no effect on the +
 16347 process. +

16348 Setting a signal action to SIG_DFL for a signal that is pending, and whose default action is to
 16349 ignore the signal (for example, SIGCHLD), shall cause the pending signal to be discarded,
 16350 whether or not it is blocked. Any queued values pending shall be discarded and the resources
 16351 used to queue them shall be released and returned to the system for other use.

16352 The default action for SIGCONT is to resume execution at the point where the process was
 16353 stopped, after first handling any pending unblocked signals.

16354 XSI When a stopped process is continued, a SIGCHLD signal may be generated for its parent
 16355 process, unless the parent process has set the SA_NOCLDSTOP flag.

16356 SIG_IGN

16357 Ignore signal.

16358 Delivery of the signal shall have no effect on the process. The behavior of a process is undefined
 16359 after it ignores a SIGFPE, SIGILL, SIGSEGV, or SIGBUS signal that was not generated by *kill()*,
 16360 *sigqueue()*, or *raise()*.

16361 The system shall not allow the action for the signals SIGKILL or SIGSTOP to be set to SIG_IGN.

16362 Setting a signal action to SIG_IGN for a signal that is pending shall cause the pending signal to
 16363 be discarded, whether or not it is blocked.

16364 XSI If a process sets the action for the SIGCHLD signal to SIG_IGN, the behavior is unspecified,
 16365 except as specified below.

16366 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes
 16367 shall not be transformed into zombie processes when they terminate. If the calling process
 16368 subsequently waits for its children, and the process has no unwaited-for children that were
 16369 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,
 16370 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

16371 Any queued values pending shall be discarded and the resources used to queue them shall be
 16372 released and made available to queue other signals.

16373 Pointer to a Function

16374 Catch signal.

16375 On delivery of the signal, the receiving process is to execute the signal-catching function at the
 16376 specified address. After returning from the signal-catching function, the receiving process shall
 16377 resume execution at the point at which it was interrupted.

16378 If the SA_SIGINFO flag for the signal is cleared, the signal-catching function shall be entered as
 16379 a C-language function call as follows:

```
16380 void func(int signo);
```

16381 If the SA_SIGINFO flag for the signal is set, the signal-catching function shall be entered as a C-
 16382 language function call as follows:

```
16383 void func(int signo, siginfo_t *info, void *context);
```

16384 where *func* is the specified signal-catching function, *signo* is the signal number of the signal
 16385 being delivered, and *info* is a pointer to a **siginfo_t** structure defined in **<signal.h>** containing at
 16386 least the following members:

16387
16388
16389
16390

Member Type	Member Name	Description
int	<i>si_signo</i>	Signal number.
int	<i>si_code</i>	Cause of the signal.
union signal	<i>si_value</i>	Signal value.

16391
16392
16393

The *si_signo* member shall contain the signal number. This shall be the same as the *signo* parameter. The *si_code* member shall contain a code identifying the cause of the signal. The following values are defined for *si_code*:

16394
16395
16396

SI_USER The signal was sent by the *kill()* function. The implementation may set *si_code* to SI_USER if the signal was sent by the *raise()* or *abort()* functions or any similar functions provided as implementation extensions.

16397

SI_QUEUE The signal was sent by the *sigqueue()* function.

16398

SI_TIMER The signal was generated by the expiration of a timer set by *timer_settime()*.

16399

SI_ASYNCIO The signal was generated by the completion of an asynchronous I/O request.

16400

MSG

SI_MESGQ The signal was generated by the arrival of a message on an empty message queue.

16401

16402

If the signal was not generated by one of the functions or events listed above, the *si_code* shall be set to an implementation-defined value that is not equal to any of the values defined above.

16403

16404

If *si_code* is one of SI_QUEUE, SI_TIMER, SI_ASYNCIO, or SI_MESGQ, then *si_value* shall contain the application-specified signal value. Otherwise, the contents of *si_value* are undefined.

16405

16406

The behavior of a process is undefined after it returns normally from a signal-catching function for a SIGBUS, SIGFPE, SIGILL, or SIGSEGV signal that was not generated by *kill()*, *sigqueue()*, or *raise()*.

16407

16408

16409

The system shall not allow a process to catch the signals SIGKILL and SIGSTOP.

16410

If a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited, it is unspecified whether a SIGCHLD signal is generated to indicate that child process.

16411

16412

16413

When signal-catching functions are invoked asynchronously with process execution, the behavior of some of the functions defined by this volume of POSIX.1-200x is unspecified if they are called from a signal-catching function.

16414

16415

16416

The following table defines a set of functions that shall be either reentrant or non-interruptible by signals and shall be async-signal-safe. Therefore, applications may invoke them, without restriction, from signal-catching functions:

16417

16418

16419	<code>_Exit()</code>	<code>fork()</code>	<code>poll()</code>	<code>sigset()</code>
16420	<code>_exit()</code>	<code>fpathconf()</code>	<code>posix_trace_event()</code>	<code>sigsuspend()</code>
16421	<code>abort()</code>	<code>fstat()</code>	<code>pselect()</code>	<code>sleep()</code>
16422	<code>accept()</code>	<code>fstatat()</code>	<code>raise()</code>	<code>socketatmark()</code>
16423	<code>access()</code>	<code>fsync()</code>	<code>read()</code>	<code>socket()</code>
16424	<code>aio_error()</code>	<code>ftruncate()</code>	<code>readlink()</code>	<code>socketpair()</code>
16425	<code>aio_return()</code>	<code>futimens()</code>	<code>readlinkat()</code>	<code>stat()</code>
16426	<code>aio_suspend()</code>	<code>getegid()</code>	<code>recv()</code>	<code>symlink()</code>
16427	<code>alarm()</code>	<code>geteuid()</code>	<code>recvfrom()</code>	<code>symlinkat()</code>
16428	<code>bind()</code>	<code>getgid()</code>	<code>recvmsg()</code>	<code>sysconf()</code>
16429	<code>cfgetispeed()</code>	<code>getgroups()</code>	<code>rename()</code>	<code>tcdrain()</code>
16430	<code>cfgetospeed()</code>	<code>getpeername()</code>	<code>renameat()</code>	<code>tcflow()</code>
16431	<code>cfsetispeed()</code>	<code>getpgrp()</code>	<code>rmdir()</code>	<code>tcflush()</code>
16432	<code>cfsetospeed()</code>	<code>getpid()</code>	<code>select()</code>	<code>tcgetattr()</code>
16433	<code>chdir()</code>	<code>getppid()</code>	<code>sem_post()</code>	<code>tcgetpgrp()</code>
16434	<code>chmod()</code>	<code>getsockname()</code>	<code>send()</code>	<code>tcsendbreak()</code>
16435	<code>chown()</code>	<code>getsockopt()</code>	<code>sendmsg()</code>	<code>tcsetattr()</code>
16436	<code>clock_gettime()</code>	<code>getuid()</code>	<code>sendto()</code>	<code>tcsetpgrp()</code>
16437	<code>close()</code>	<code>kill()</code>	<code>setgid()</code>	<code>time()</code>
16438	<code>connect()</code>	<code>link()</code>	<code>setpgid()</code>	<code>timer_getoverrun()</code>
16439	<code>creat()</code>	<code>linkat()</code>	<code>setpid()</code>	<code>timer_gettime()</code>
16440	<code>dup()</code>	<code>listen()</code>	<code>setsid()</code>	<code>timer_settime()</code>
16441	<code>dup2()</code>	<code>lseek()</code>	<code>setsockopt()</code>	<code>times()</code>
16442	<code>execl()</code>	<code>lstat()</code>	<code>setuid()</code>	<code>times()</code>
16443	<code>execle()</code>	<code>mkdir()</code>	<code>shutdown()</code>	<code>umask()</code>
16444	<code>execv()</code>	<code>mkdirat()</code>	<code>sigaction()</code>	<code>uname()</code>
16445	<code>execve()</code>	<code>mkfifo()</code>	<code>sigaddset()</code>	<code>unlink()</code>
16446	<code>faccessat()</code>	<code>mkfifoat()</code>	<code>sigdelset()</code>	<code>unlinkat()</code>
16447	<code>fchmod()</code>	<code>mkfifoat()</code>	<code>sigemptyset()</code>	<code>utime()</code>
16448	<code>fchmodat()</code>	<code>mknod()</code>	<code>sigfillset()</code>	<code>utimensat()</code>
16449	<code>fchown()</code>	<code>mknodat()</code>	<code>sigismember()</code>	<code>utimes()</code>
16450	<code>fchownat()</code>	<code>open()</code>	<code>signal()</code>	<code>wait()</code>
16451	<code>fcntl()</code>	<code>openat()</code>	<code>sigpause()</code>	<code>waitpid()</code>
16452	<code>fdatasync()</code>	<code>pathconf()</code>	<code>sigpending()</code>	<code>write()</code>
16453	<code>fexecve()</code>	<code>pause()</code>	<code>sigprocmask()</code>	
		<code>pipe()</code>	<code>sigqueue()</code>	

16454 All functions not in the above table are considered to be unsafe with respect to signals. In the
 16455 presence of signals, all functions defined by this volume of POSIX.1-200x shall behave as defined
 16456 when called from or interrupted by a signal-catching function, with a single exception: when a
 16457 signal interrupts an unsafe function and the signal-catching function calls an unsafe function,
 16458 the behavior is undefined.

16459 Operations which obtain the value of *errno* and operations which assign a value to *errno* shall be +
 16460 *async-signal-safe*. +

16461 When a signal is delivered to a thread, if the action of that signal specifies termination, stop, or
 16462 continue, the entire process shall be terminated, stopped, or continued, respectively.

16463 2.4.4 Signal Effects on Other Functions

16464 Signals affect the behavior of certain functions defined by this volume of POSIX.1-200x if
 16465 delivered to a process while it is executing such a function. If the action of the signal is to
 16466 terminate the process, the process shall be terminated and the function shall not return. If the
 16467 action of the signal is to stop the process, the process shall stop until continued or terminated.
 16468 Generation of a SIGCONT signal for the process shall cause the process to be continued, and the
 16469 original function shall continue at the point the process was stopped. If the action of the signal is
 16470 to invoke a signal-catching function, the signal-catching function shall be invoked; in this case
 16471 the original function is said to be “interrupted” by the signal. If the signal-catching function
 16472 executes a **return** statement, the behavior of the interrupted function shall be as described
 16473 individually for that function, except as noted for unsafe functions. Signals that are ignored shall
 16474 not affect the behavior of any function; signals that are blocked shall not affect the behavior of
 16475 any function until they are unblocked and then delivered, except as specified for the *sigpending()*
 16476 and *sigwait()* functions.

16477 2.5 Standard I/O Streams

16478 CX A stream is associated with an external file (which may be a physical device) or memory buffer
 16479 CX by “opening” a file or buffer. This may involve “creating” a new file. Creating an existing file
 16480 causes its former contents to be discarded if necessary. If a file can support positioning requests
 16481 (such as a disk file, as opposed to a terminal), then a “file position indicator” associated with the
 16482 stream is positioned at the start (byte number 0) of the file, unless the file is opened with append
 16483 mode, in which case it is implementation-defined whether the file position indicator is initially
 16484 positioned at the beginning or end of the file. The file position indicator is maintained by
 16485 subsequent reads, writes, and positioning requests, to facilitate an orderly progression through
 16486 the file. All input takes place as if bytes were read by successive calls to *fgetc()*; all output takes
 16487 place as if bytes were written by successive calls to *fputc()*.

16488 When a stream is “unbuffered”, bytes are intended to appear from the source or at the
 16489 destination as soon as possible; otherwise, bytes may be accumulated and transmitted as a
 16490 block. When a stream is “fully buffered”, bytes are intended to be transmitted as a block when a
 16491 buffer is filled. When a stream is “line buffered”, bytes are intended to be transmitted as a block
 16492 when a newline byte is encountered. Furthermore, bytes are intended to be transmitted as a
 16493 block when a buffer is filled, when input is requested on an unbuffered stream, or when input is
 16494 requested on a line-buffered stream that requires the transmission of bytes. Support for these
 16495 characteristics is implementation-defined, and may be affected via *setbuf()* and *setobuf()*.

16496 A file may be disassociated from a controlling stream by “closing” the file. Output streams are
 16497 flushed (any unwritten buffer contents are transmitted) before the stream is disassociated from
 16498 the file. The value of a pointer to a **FILE** object is unspecified after the associated file is closed
 16499 (including the standard streams).

16500 A file may be subsequently reopened, by the same or another program execution, and its
 16501 contents reclaimed or modified (if it can be repositioned at its start). If the *main()* function
 16502 returns to its original caller, or if the *exit()* function is called, all open files are closed (hence all
 16503 output streams are flushed) before program termination. Other paths to program termination,
 16504 such as calling *abort()*, need not close all files properly.

16505 The address of the **FILE** object used to control a stream may be significant; a copy of a **FILE**
 16506 object need not necessarily serve in place of the original.

16507 At program start-up, three streams are predefined and need not be opened explicitly: *standard*
 16508 *input* (for reading conventional input), *standard output* (for writing conventional output), and
 16509 *standard error* (for writing diagnostic output). When opened, the standard error stream is not

16510 fully buffered; the standard input and standard output streams are fully buffered if and only if
16511 the stream can be determined not to refer to an interactive device.

16512 CX A stream associated with a memory buffer shall have the same operations for text files that a
16513 stream associated with an external file would have. In addition, the stream orientation shall be
16514 determined in exactly the same fashion.

16515 Input and output operations on a stream associated with a memory buffer by a call to
16516 *fmemopen()* shall be constrained by the implementation to take place within the bounds of the
16517 memory buffer. In the case of a stream opened by *open_memstream()* or *open_wmemstream()*, the
16518 memory area shall grow dynamically to accommodate write operations as necessary. For output,
16519 data is moved from the buffer provided by *setvbuf()* to the memory stream during a flush or
16520 close operation.

16521 2.5.1 Interaction of File Descriptors and Standard I/O Streams

16522 CX This section describes the interaction of file descriptors and standard I/O streams. The
16523 functionality described in this section is an extension to the ISO C standard (and the rest of this
16524 section is not further CX shaded).

16525 An open file description may be accessed through a file descriptor, which is created using
16526 functions such as *open()* or *pipe()*, or through a stream, which is created using functions such as
16527 *fopen()* or *popen()*. Either a file descriptor or a stream is called a “handle” on the open file
16528 description to which it refers; an open file description may have several handles.

16529 Handles can be created or destroyed by explicit user action, without affecting the underlying
16530 open file description. Some of the ways to create them include *fcntl()*, *dup()*, *fdopen()*, *fileno()*,
16531 and *fork()*. They can be destroyed by at least *fclose()*, *close()*, and the *exec* functions.

16532 A file descriptor that is never used in an operation that could affect the file offset (for example,
16533 *read()*, *write()*, or *lseek()*) is not considered a handle for this discussion, but could give rise to
16534 one (for example, as a consequence of *fdopen()*, *dup()*, or *fork()*). This exception does not include
16535 the file descriptor underlying a stream, whether created with *fopen()* or *fdopen()*, so long as it is
16536 not used directly by the application to affect the file offset. The *read()* and *write()* functions
16537 implicitly affect the file offset; *lseek()* explicitly affects it.

16538 The result of function calls involving any one handle (the “active handle”) is defined elsewhere
16539 in this volume of POSIX.1-200x, but if two or more handles are used, and any one of them is a
16540 stream, the application shall ensure that their actions are coordinated as described below. If this
16541 is not done, the result is undefined.

16542 A handle which is a stream is considered to be closed when either an *fclose()* or *freopen()* is
16543 executed on it (the result of *freopen()* is a new stream, which cannot be a handle on the same
16544 open file description as its previous value), or when the process owning that stream terminates
16545 with *exit()*, *abort()*, or due to a signal. A file descriptor is closed by *close()*, *_exit()*, or the *exec*
16546 functions when *FD_CLOEXEC* is set on that file descriptor.

16547 For a handle to become the active handle, the application shall ensure that the actions below are
16548 performed between the last use of the handle (the current active handle) and the first use of the
16549 second handle (the future active handle). The second handle then becomes the active handle. All
16550 activity by the application affecting the file offset on the first handle shall be suspended until it
16551 again becomes the active file handle. (If a stream function has as an underlying function one that
16552 affects the file offset, the stream function shall be considered to affect the file offset.)

16553 The handles need not be in the same process for these rules to apply.

16554 Note that after a *fork()*, two handles exist where one existed before. The application shall ensure
16555 that, if both handles can ever be accessed, they are both in a state where the other could become

16556 the active handle first. The application shall prepare for a *fork()* exactly as if it were a change of
 16557 active handle. (If the only action performed by one of the processes is one of the *exec* functions or
 16558 *_exit()* (not *exit()*), the handle is never accessed in that process.)

16559 For the first handle, the first applicable condition below applies. After the actions required
 16560 below are taken, if the handle is still open, the application can close it.

- 16561 • If it is a file descriptor, no action is required.
- 16562 • If the only further action to be performed on any handle to this open file descriptor is to
 16563 close it, no action need be taken.
- 16564 • If it is a stream which is unbuffered, no action need be taken.
- 16565 • If it is a stream which is line buffered, and the last byte written to the stream was a
 16566 <newline> (that is, as if a:

16567 `putc('\n')`

16568 was the most recent operation on that stream), no action need be taken.

- 16569 • If it is a stream which is open for writing or appending (but not also open for reading), the
 16570 application shall either perform an *fflush()*, or the stream shall be closed.
- 16571 • If the stream is open for reading and it is at the end of the file (*feof()* is true), no action need
 16572 be taken.
- 16573 • If the stream is open with a mode that allows reading and the underlying open file
 16574 description refers to a device that is capable of seeking, the application shall either perform
 16575 an *fflush()*, or the stream shall be closed.

16576 Otherwise, the result is undefined.

16577 For the second handle:

- 16578 • If any previous active handle has been used by a function that explicitly changed the file
 16579 offset, except as required above for the first handle, the application shall perform an *lseek()*
 16580 or *fseek()* (as appropriate to the type of handle) to an appropriate location.

16581 If the active handle ceases to be accessible before the requirements on the first handle, above,
 16582 have been met, the state of the open file description becomes undefined. This might occur
 16583 during functions such as a *fork()* or *_exit()*.

16584 The *exec* functions make inaccessible all streams that are open at the time they are called,
 16585 independent of which streams or file descriptors may be available to the new process image.

16586 When these rules are followed, regardless of the sequence of handles used, implementations
 16587 shall ensure that an application, even one consisting of several processes, shall yield correct
 16588 results: no data shall be lost or duplicated when writing, and all data shall be written in order,
 16589 except as requested by seeks. It is implementation-defined whether, and under what conditions,
 16590 all input is seen exactly once.

16591 If the rules above are not followed, the result is unspecified.

16592 Each function that operates on a stream is said to have zero or more “underlying functions”.
 16593 This means that the stream function shares certain traits with the underlying functions, but does
 16594 not require that there be any relation between the implementations of the stream function and its
 16595 underlying functions.

2.5.2 Stream Orientation and Encoding Rules

For conformance to the ISO/IEC 9899:1999 standard, the definition of a stream includes an “orientation”. After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide-character input/output function has been applied to a stream without orientation, the stream shall become “wide-oriented”. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream shall become “byte-oriented”. Only a call to the *freopen()* function or the *fwide()* function can otherwise alter the orientation of a stream.

A successful call to *freopen()* shall remove any orientation. The three predefined streams *standard input*, *standard output*, and *standard error* shall be unoriented at program start-up.

Byte input/output functions cannot be applied to a wide-oriented stream, and wide-character input/output functions cannot be applied to a byte-oriented stream. The remaining stream operations shall not affect and shall not be affected by a stream’s orientation, except for the following additional restriction:

- For wide-oriented streams, after a successful call to a file-positioning function that leaves the file position indicator prior to the end-of-file, a wide-character output function can overwrite a partial character; any file contents beyond the byte(s) written are henceforth undefined.

Each wide-oriented stream has an associated **mbstate_t** object that stores the current parse state of the stream. A successful call to *fgetpos()* shall store a representation of the value of this **mbstate_t** object as part of the value of the **fpos_t** object. A later successful call to *fsetpos()* using the same stored **fpos_t** value shall restore the value of the associated **mbstate_t** object as well as the position within the controlled stream.

Implementations that support multiple encoding rules associate an encoding rule with the stream. The encoding rule shall be determined by the setting of the *LC_CTYPE* category in the current locale at the time when the stream becomes wide-oriented. As with the stream’s orientation, the encoding rule associated with a stream cannot be changed once it has been set, except by a successful call to *freopen()* which clears the encoding rule and resets the orientation to unoriented.

Although wide-oriented streams are conceptually sequences of wide characters, the external file associated with a wide-oriented stream is a sequence of (possibly multi-byte) characters generalized as follows:

- Multi-byte encodings within files may contain embedded null bytes (unlike multi-byte encodings valid for use internal to the program).
- A file need not begin nor end in the initial shift state.

Moreover, the encodings used for characters may differ among files. Both the nature and choice of such encodings are implementation-defined.

The wide-character input functions read characters from the stream and convert them to wide characters as if they were read by successive calls to the *fgetwc()* function. Each conversion shall occur as if by a call to the *mbrtowc()* function, with the conversion state described by the stream’s own **mbstate_t** object, **except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current locale.**

The wide-character output functions convert wide characters to (possibly multi-byte) characters and write them to the stream as if they were written by successive calls to the *fputwc()* function. Each conversion shall occur as if by a call to the *wcrtomb()* function, with the conversion state described by the stream’s own **mbstate_t** object, **except the encoding rule associated with the stream is used instead of the encoding rule implied by the *LC_CTYPE* category of the current**

16643

locale.

16644

16645

16646

16647

16648

An “encoding error” shall occur if the character sequence presented to the underlying `mbrtowc()` function does not form a valid (generalized) character, or if the code value passed to the underlying `wcrtomb()` function does not correspond to a valid (generalized) character. The wide-character input/output functions and the byte input/output functions store the value of the macro `[EILSEQ]` in `errno` if and only if an encoding error occurs.

16649

2.6 STREAMS

16650

OB XSR

16651

16652

STREAMS functionality is provided on implementations supporting the XSI STREAMS Option Group. The functionality described in this section is dependent on support of the XSI STREAMS option (and the rest of this section is not further shaded for this option).

16653

16654

16655

16656

16657

STREAMS provides a uniform mechanism for implementing networking services and other character-based I/O. The STREAMS function provides direct access to protocol modules. STREAMS modules are unspecified objects. Access to STREAMS modules is provided by interfaces in POSIX.1-200x. Creation of STREAMS modules is outside the scope of POSIX.1-200x.

16658

16659

16660

16661

16662

16663

A STREAM is typically a full-duplex connection between a process and an open device or pseudo-device. However, since pipes may be STREAMS-based, a STREAM can be a full-duplex connection between two processes. The STREAM itself exists entirely within the implementation and provides a general character I/O function for processes. It optionally includes one or more intermediate processing modules that are interposed between the process end of the STREAM (STREAM head) and a device driver at the end of the STREAM (STREAM end).

16664

STREAMS I/O is based on messages. There are three types of message:

16665

- *Data messages* containing actual data for input or output
- *Control data* containing instructions for the STREAMS modules and underlying implementation
- Other messages, which include file descriptors

16666

16667

16668

16669

16670

16671

16672

16673

16674

16675

The interface between the STREAM and the rest of the implementation is provided by a set of functions at the STREAM head. When a process calls `write()`, `writew()`, `putmsg()`, `putpmsg()`, or `ioctl()`, messages are sent down the STREAM, and `read()`, `readv()`, `getmsg()`, or `getpmsg()` accepts data from the STREAM and passes it to a process. Data intended for the device at the downstream end of the STREAM is packaged into messages and sent downstream, while data and signals from the device are composed into messages by the device driver and sent upstream to the STREAM head.

16676

16677

16678

16679

16680

16681

When a STREAMS-based device is opened, a STREAM shall be created that contains the STREAM head and the STREAM end (driver). If pipes are STREAMS-based in an implementation, when a pipe is created, two STREAMS shall be created, each containing a STREAM head. Other modules are added to the STREAM using `ioctl()`. New modules are “pushed” onto the STREAM one at a time in last-in, first-out (LIFO) style, as though the STREAM was a push-down stack.

16682

Priority

16683

16684

16685

16686

16687

16688

16689

16690

16691

16692

16693

Message types are classified according to their queuing priority and may be *normal* (non-priority), *priority*, or *high-priority* messages. A message belongs to a particular priority band that determines its ordering when placed on a queue. Normal messages have a priority band of 0 and shall always be placed at the end of the queue following all other messages in the queue. High-priority messages are always placed at the head of a queue, but shall be discarded if there is already a high-priority message in the queue. Their priority band shall be ignored; they are high-priority by virtue of their type. Priority messages have a priority band greater than 0. Priority messages are always placed after any messages of the same or higher priority. High-priority and priority messages are used to send control and data information outside the normal flow of control. By convention, high-priority messages shall not be affected by flow control. Normal and priority messages have separate flow controls.

16694

Message Parts

16695

16696

16697

16698

16699

16700

16701

A process may access STREAMS messages that contain a data part, control part, or both. The data part is that information which is transmitted over the communication medium and the control information is used by the local STREAMS modules. The other types of messages are used between modules and are not accessible to processes. Messages containing only a data part are accessible via *putmsg()*, *putpmsg()*, *getmsg()*, *getpmsg()*, *read()*, *readv()*, *write()*, or *writv()*. Messages containing a control part with or without a data part are accessible via calls to *putmsg()*, *putpmsg()*, *getmsg()*, or *getpmsg()*.

16702

2.6.1 Accessing STREAMS

16703

16704

16705

A process accesses STREAMS-based files using the standard functions *close()*, *ioctl()*, *getmsg()*, *getpmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *putpmsg()*, *read()*, or *write()*. Refer to the applicable function definitions for general properties and errors.

16706

16707

16708

Calls to *ioctl()* shall perform control functions on the STREAM associated with the file descriptor *fdes*. The control functions may be performed by the STREAM head, a STREAMS module, or the STREAMS driver for the STREAM.

16709

16710

16711

16712

16713

STREAMS modules and drivers can detect errors, sending an error message to the STREAM head, thus causing subsequent functions to fail and set *errno* to the value specified in the message. In addition, STREAMS modules and drivers can elect to fail a particular *ioctl()* request alone by sending a negative acknowledgement message to the STREAM head. This shall cause just the pending *ioctl()* request to fail and set *errno* to the value specified in the message.

16714

2.7 XSI Interprocess Communication

16715

16716

16717

XSI

This section describes extensions to support interprocess communication. The functionality described in this section shall be provided on implementations that support the XSI option (and the rest of this section is not further shaded).

16718

16719

16720

The following message passing, semaphore, and shared memory services form an XSI interprocess communication facility. Certain aspects of their operation are common, and are defined as follows.

16721
16722
16723
16724
16725

IPC Functions		
<i>msgctl()</i>	<i>semctl()</i>	<i>shmctl()</i>
<i>msgget()</i>	<i>semget()</i>	<i>shmdt()</i>
<i>msgrcv()</i>	<i>semop()</i>	<i>shmget()</i>
<i>msgsnd()</i>	<i>shmat()</i>	

16726
16727

Another interprocess communication facility is provided by functions in the Realtime Option Group; see [Section 2.8](#) (on page 476).

16728

2.7.1 IPC General Description

16729
16730
16731
16732

Each individual shared memory segment, message queue, and semaphore set shall be identified by a unique positive integer, called, respectively, a shared memory identifier, *shm_{id}*, a semaphore identifier, *sem_{id}*, and a message queue identifier, *msg_{id}*. The identifiers shall be returned by calls to *shmget()*, *semget()*, and *msgget()*, respectively.

16733
16734
16735

Associated with each identifier is a data structure which contains data related to the operations which may be or may have been performed; see the Base Definitions volume of POSIX.1-200x, [<sys/shm.h>](#), [<sys/sem.h>](#), and [<sys/msg.h>](#) for their descriptions.

16736
16737
16738
16739
16740

Each of the data structures contains both ownership information and an **ipc_perm** structure (see the Base Definitions volume of POSIX.1-200x, [<sys/ipc.h>](#)) which are used in conjunction to determine whether or not read/write (read/alter for semaphores) permissions should be granted to processes using the IPC facilities. The *mode* member of the **ipc_perm** structure acts as a bit field which determines the permissions.

16741

The values of the bits are given below in octal notation.

16742
16743
16744
16745
16746
16747
16748

Bit	Meaning
0400	Read by user.
0200	Write by user.
0040	Read by group.
0020	Write by group.
0004	Read by others.
0002	Write by others.

16749
16750
16751
16752

The name of the **ipc_perm** structure is *shm_perm*, *sem_perm*, or *msg_perm*, depending on which service is being used. In each case, read and write/alter permissions shall be granted to a process if one or more of the following are true ("xxx" is replaced by *shm*, *sem*, or *msg*, as appropriate):

16753
16754
16755
16756

- The process has appropriate privileges.
- The effective user ID of the process matches *xxx_perm.cuid* or *xxx_perm.uid* in the data structure associated with the IPC identifier, and the appropriate bit of the *user* field in *xxx_perm.mode* is set.

16757
16758
16759
16760

- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* but the effective group ID of the process matches *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, and the appropriate bit of the *group* field in *xxx_perm.mode* is set.

16761
16762
16763
16764

- The effective user ID of the process does not match *xxx_perm.cuid* or *xxx_perm.uid* and the effective group ID of the process does not match *xxx_perm.cgid* or *xxx_perm.gid* in the data structure associated with the IPC identifier, but the appropriate bit of the *other* field in *xxx_perm.mode* is set.

16765

Otherwise, the permission shall be denied.

2.8 Realtime

This section defines functions to support the source portability of applications with realtime requirements. The presence of some of these functions is dependent on support for implementation options described in the text.

The specific functional areas included in this section and their scope include the following. Full definitions of these terms can be found in XBD [Chapter 3](#) (on page 33).

- Semaphores
- Process Memory Locking
- Memory Mapped Files and Shared Memory Objects
- Priority Scheduling
- Realtime Signal Extension
- Timers
- Interprocess Communication
- Synchronized Input and Output
- Asynchronous Input and Output

All the realtime functions defined in this volume of POSIX.1-200x are portable, although some of the numeric parameters used by an implementation may have hardware dependencies.

2.8.1 Realtime Signals

See [Section 2.4.2](#) (on page 464).

2.8.2 Asynchronous I/O

An asynchronous I/O control block structure `aiocb` is used in many asynchronous I/O functions. It is defined in the Base Definitions volume of POSIX.1-200x, `<aio.h>` and has at least the following members:

Member Type	Member Name	Description
<code>int</code>	<code>aio_fildes</code>	File descriptor.
<code>off_t</code>	<code>aio_offset</code>	File offset.
<code>volatile void*</code>	<code>aio_buf</code>	Location of buffer.
<code>size_t</code>	<code>aio_nbytes</code>	Length of transfer.
<code>int</code>	<code>aio_repprio</code>	Request priority offset.
<code>struct sigevent</code>	<code>aio_sigevent</code>	Signal number and value.
<code>int</code>	<code>aio_lio_opcode</code>	Operation to be performed.

The `aio_fildes` element is the file descriptor on which the asynchronous operation is performed.

If `O_APPEND` is not set for the file descriptor `aio_fildes` and if `aio_fildes` is associated with a device that is capable of seeking, then the requested operation takes place at the absolute position in the file as given by `aio_offset`, as if `lseek()` were called immediately prior to the operation with an `offset` argument equal to `aio_offset` and a `whence` argument equal to `SEEK_SET`. If `O_APPEND` is set for the file descriptor, or if `aio_fildes` is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, with the following exception: under implementation-defined circumstances, such as operation on a multi-processor or when requests of differing priorities are submitted at the same time, the ordering restriction may be relaxed. Since there is no way for a strictly conforming

16807 application to determine whether this relaxation applies, all strictly conforming applications
 16808 which rely on ordering of output shall be written in such a way that they will operate correctly if
 16809 the relaxation applies. After a successful call to enqueue an asynchronous I/O operation, the
 16810 value of the file offset for the file is unspecified. The *aio_nbytes* and *aio_buf* elements are the same
 16811 as the *nbyte* and *buf* arguments defined by *read()* and *write()*, respectively.

16812 If `_POSIX_PRIORITIZED_IO` and `_POSIX_PRIORITY_SCHEDULING` are defined, then
 16813 asynchronous I/O is queued in priority order, with the priority of each asynchronous operation
 16814 based on the current scheduling priority of the calling process. The *aio_reqprio* member can be
 16815 used to lower (but not raise) the asynchronous I/O operation priority and is within the range
 16816 zero through `{AIO_PRIO_DELTA_MAX}`, inclusive. Unless both `_POSIX_PRIORITIZED_IO` and
 16817 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing asynchronous I/O
 16818 requests is unspecified. When both `_POSIX_PRIORITIZED_IO` and
 16819 `_POSIX_PRIORITY_SCHEDULING` are defined, the order of processing of requests submitted
 16820 by processes whose schedulers are not `SCHED_FIFO`, `SCHED_RR`, or `SCHED_SPORADIC` is
 16821 unspecified. The priority of an asynchronous request is computed as (process scheduling
 16822 priority) minus *aio_reqprio*. The priority assigned to each asynchronous I/O request is an
 16823 indication of the desired order of execution of the request relative to other asynchronous I/O
 16824 requests for this file. If `_POSIX_PRIORITIZED_IO` is defined, requests issued with the same
 16825 priority to a character special file are processed by the underlying device in FIFO order; the
 16826 order of processing of requests of the same priority issued to files that are not character special
 16827 files is unspecified. Numerically higher priority values indicate requests of higher priority. The
 16828 value of *aio_reqprio* has no effect on process scheduling priority. When prioritized asynchronous
 16829 I/O requests to the same file are blocked waiting for a resource required for that I/O operation,
 16830 the higher-priority I/O requests shall be granted the resource before lower-priority I/O requests
 16831 are granted the resource. The relative priority of asynchronous I/O and synchronous I/O is
 16832 implementation-defined. If `_POSIX_PRIORITIZED_IO` is defined, the implementation shall
 16833 define for which files I/O prioritization is supported.

16834 The *aio_sigevent* determines how the calling process shall be notified upon I/O completion, as
 16835 specified in [Section 2.4.1](#) (on page 463). If *aio_sigevent.sigev_notify* is `SIGEV_NONE`, then no
 16836 signal shall be posted upon I/O completion, but the error status for the operation and the return
 16837 status for the operation shall be set appropriately.

16838 The *aio_lio_opcode* field is used only by the *lio_listio()* call. The *lio_listio()* call allows multiple
 16839 asynchronous I/O operations to be submitted at a single time. The function takes as an
 16840 argument an array of pointers to **aiocb** structures. Each **aiocb** structure indicates the operation to
 16841 be performed (read or write) via the *aio_lio_opcode* field.

16842 The address of the **aiocb** structure is used as a handle for retrieving the error status and return
 16843 status of the asynchronous operation while it is in progress.

16844 The **aiocb** structure and the data buffers associated with the asynchronous I/O operation are
 16845 being used by the system for asynchronous I/O while, and only while, the error status of the
 16846 asynchronous operation is equal to `[EINPROGRESS]`. Applications shall not modify the **aiocb**
 16847 structure while the structure is being used by the system for asynchronous I/O.

16848 The return status of the asynchronous operation is the number of bytes transferred by the I/O
 16849 operation. If the error status is set to indicate an error completion, then the return status is set to
 16850 the return value that the corresponding *read()*, *write()*, or *fsync()* call would have returned.
 16851 When the error status is not equal to `[EINPROGRESS]`, the return status shall reflect the return
 16852 status of the corresponding synchronous operation.

16853 **2.8.3 Memory Management**16854 **2.8.3.1 Memory Locking**

16855 MLR Range memory locking operations are defined in terms of pages. Implementations may restrict
 16856 the size and alignment of range lockings to be on page-size boundaries. The page size, in bytes,
 16857 is the value of the configurable system variable {PAGESIZE}. If an implementation has no
 16858 restrictions on size or alignment, it may specify a 1-byte page size.

16859 ML | MLR Memory locking guarantees the residence of portions of the address space. It is implementation-
 16860 defined whether locking memory guarantees fixed translation between virtual addresses (as
 16861 seen by the process) and physical addresses. Per-process memory locks are not inherited across a
 16862 *fork()*, and all memory locks owned by a process are unlocked upon *exec* or process termination.
 16863 Unmapping of an address range removes any memory locks established on that address range
 16864 by this process.

16865 **2.8.3.2 Memory Mapped Files**

16866 Range memory mapping operations are defined in terms of pages. Implementations may
 16867 restrict the size and alignment of range mappings to be on page-size boundaries. The page size,
 16868 in bytes, is the value of the configurable system variable {PAGESIZE}. If an implementation has
 16869 no restrictions on size or alignment, it may specify a 1-byte page size.

16870 Memory mapped files provide a mechanism that allows a process to access files by directly
 16871 incorporating file data into its address space. Once a file is mapped into a process address space,
 16872 the data can be manipulated as memory. If more than one process maps a file, its contents are
 16873 shared among them. If the mappings allow shared write access, then data written into the
 16874 memory object through the address space of one process appears in the address spaces of all
 16875 processes that similarly map the same portion of the memory object.

16876 SHM Shared memory objects are named regions of storage that may be independent of the file system
 16877 and can be mapped into the address space of one or more processes to allow them to share the
 16878 associated memory.

16879 SHM An *unlink()* of a file or *shm_unlink()* of a shared memory object, while causing the removal of
 16880 the name, does not unmap any mappings established for the object. Once the name has been
 16881 removed, the contents of the memory object are preserved as long as it is referenced. The
 16882 memory object remains referenced as long as a process has the memory object open or has some
 16883 area of the memory object mapped.

16884 **2.8.3.3 Memory Protection**

16885 When an object is mapped, various application accesses to the mapped region may result in
 16886 signals. In this context, SIGBUS is used to indicate an error using the mapped object, and
 16887 SIGSEGV is used to indicate a protection violation or misuse of an address:

- 16888 • A mapping may be restricted to disallow some types of access.
- 16889 • Write attempts to memory that was mapped without write access, or any access to
 16890 memory mapped PROT_NONE, shall result in a SIGSEGV signal.
- 16891 • References to unmapped addresses shall result in a SIGSEGV signal.
- 16892 • Reference to whole pages within the mapping, but beyond the current length of the object,
 16893 shall result in a SIGBUS signal.

- 16894 • The size of the object is unaffected by access beyond the end of the object (even if a
16895 SIGBUS is not generated).

16896 2.8.3.4 Typed Memory Objects

16897 TYM The functionality described in this section shall be provided on implementations that support
16898 the Typed Memory Objects option (and the rest of this section is not further shaded for this
16899 option).

16900 Implementations may support the Typed Memory Objects option independently of support for
16901 memory mapped files or shared memory objects. Typed memory objects are implementation-
16902 configurable named storage pools accessible from one or more processors in a system, each via
16903 one or more ports, such as backplane buses, LANs, I/O channels, and so on. Each valid
16904 combination of a storage pool and a port is identified through a name that is defined at system
16905 configuration time, in an implementation-defined manner; the name may be independent of the
16906 file system. Using this name, a typed memory object can be opened and mapped into process
16907 address space. For a given storage pool and port, it is necessary to support both dynamic
16908 allocation from the pool as well as mapping at an application-supplied offset within the pool;
16909 when dynamic allocation has been performed, subsequent deallocation must be supported.
16910 Lastly, accessing typed memory objects from different ports requires a method for obtaining the
16911 offset and length of contiguous storage of a region of typed memory (dynamically allocated or
16912 not); this allows typed memory to be shared among processes and/or processors while being
16913 accessed from the desired port.

16914 2.8.4 Process Scheduling

16915 PS The functionality described in this section shall be provided on implementations that support
16916 the Process Scheduling option (and the rest of this section is not further shaded for this option).

16917 Scheduling Policies

16918 The scheduling semantics described in this volume of POSIX.1-200x are defined in terms of a
16919 conceptual model that contains a set of thread lists. No implementation structures are
16920 necessarily implied by the use of this conceptual model. It is assumed that no time elapses
16921 during operations described using this model, and therefore no simultaneous operations are
16922 possible. This model discusses only processor scheduling for runnable threads, but it should be
16923 noted that greatly enhanced predictability of realtime applications results if the sequencing of
16924 other resources takes processor scheduling policy into account.

16925 There is, conceptually, one thread list for each priority. A runnable thread will be on the thread
16926 list for that thread's priority. Multiple scheduling policies shall be provided. Each non-empty
16927 thread list is ordered, contains a head as one end of its order, and a tail as the other. The purpose
16928 of a scheduling policy is to define the allowable operations on this set of lists (for example,
16929 moving threads between and within lists).

16930 The POSIX model treats a "process" as an aggregation of system resources, including one or
16931 more threads that may be scheduled by the operating system on the processor(s) it controls.
16932 Although a process has its own set of scheduling attributes, these have an indirect effect (if any)
16933 on the scheduling behavior of individual threads as described below.

16934 Each thread shall be controlled by an associated scheduling policy and priority. These
16935 parameters may be specified by explicit application execution of the *pthread_setschedparam()*
16936 function. Additionally, the scheduling parameters of a thread (but not its scheduling policy) may
16937 be changed by application execution of the *pthread_setschedprio()* function.

16938 Each process shall be controlled by an associated scheduling policy and priority. These
16939 parameters may be specified by explicit application execution of the *sched_setscheduler()* or

16940 *sched_setparam()* functions.

16941 The effect of the process scheduling attributes on individual threads in the process is dependent
16942 on the scheduling contention scope of the threads (see [Section 2.9.4](#), on page 488):

- 16943 • For threads with system scheduling contention scope, the process scheduling attributes
16944 shall have no effect on the scheduling attributes or behavior either of the thread or an
16945 underlying kernel scheduling entity dedicated to that thread.
- 16946 • For threads with process scheduling contention scope, the process scheduling attributes
16947 shall have no effect on the scheduling attributes of the thread. However, any underlying
16948 kernel scheduling entity used by these threads shall at all times behave as specified by the
16949 scheduling attributes of the containing process, and this behavior may affect the
16950 scheduling behavior of the process contention scope threads. For example, a process
16951 contention scope thread with scheduling policy `SCHED_FIFO` and the system maximum
16952 priority *H* (the value returned by *sched_get_priority_max(SCHED_FIFO)*) in a process with
16953 scheduling policy `SCHED_RR` and system minimum priority *L* (the value returned by
16954 *sched_get_priority_min(SCHED_RR)*) shall be subject to timeslicing and to preemption by
16955 any thread with an effective priority higher than *L*.

16956 Associated with each policy is a priority range. Each policy definition shall specify the minimum
16957 priority range for that policy. The priority ranges for each policy may but need not overlap the
16958 priority ranges of other policies.

16959 A conforming implementation shall select the thread that is defined as being at the head of the
16960 highest priority non-empty thread list to become a running thread, regardless of its associated
16961 policy. This thread is then removed from its thread list.

16962 Four scheduling policies are specifically required. Other implementation-defined scheduling
16963 policies may be defined. The following symbols are defined in the Base Definitions volume of
16964 POSIX.1-200x, `<sched.h>`:

16965 `SCHED_FIFO` First in, first out (FIFO) scheduling policy.

16966 `SCHED_RR` Round robin scheduling policy.

16967 SS `SCHED_SPORADIC` Sporadic server scheduling policy.

16968 `SCHED_OTHER` Another scheduling policy.

16969 The values of these symbols shall be distinct.

16970 **SCHED_FIFO**

16971 Conforming implementations shall include a scheduling policy called the FIFO scheduling
16972 policy.

16973 Threads scheduled under this policy are chosen from a thread list that is ordered by the time its
16974 threads have been on the list without being executed; generally, the head of the list is the thread
16975 that has been on the list the longest time, and the tail is the thread that has been on the list the
16976 shortest time.

16977 Under the `SCHED_FIFO` policy, the modification of the definitional thread lists is as follows:

- 16978 1. When a running thread becomes a preempted thread, it becomes the head of the thread
16979 list for its priority.
- 16980 2. When a blocked thread becomes a runnable thread, it becomes the tail of the thread list
16981 for its priority.

- 16982 3. When a running thread calls the *sched_setscheduler()* function, the process specified in the
 16983 function call is modified to the specified policy and the priority specified by the *param*
 16984 argument.
- 16985 4. When a running thread calls the *sched_setparam()* function, the priority of the process
 16986 specified in the function call is modified to the priority specified by the *param* argument.
- 16987 5. When a running thread calls the *pthread_setschedparam()* function, the thread specified in
 16988 the function call is modified to the specified policy and the priority specified by the *param*
 16989 argument.
- 16990 6. When a running thread calls the *pthread_setschedprio()* function, the thread specified in the
 16991 function call is modified to the priority specified by the *prio* argument.
- 16992 7. If a thread whose policy or priority has been modified other than by *pthread_setschedprio()*
 16993 is a running thread or is runnable, it then becomes the tail of the thread list for its new
 16994 priority.
- 16995 8. If a thread whose priority has been modified by *pthread_setschedprio()* is a running thread
 16996 or is runnable, the effect on its position in the thread list depends on the direction of the
 16997 modification, as follows:
- 16998 a. If the priority is raised, the thread becomes the tail of the thread list.
- 16999 b. If the priority is unchanged, the thread does not change position in the thread list.
- 17000 c. If the priority is lowered, the thread becomes the head of the thread list.
- 17001 9. When a running thread issues the *sched_yield()* function, the thread becomes the tail of
 17002 the thread list for its priority.
- 17003 10. At no other time is the position of a thread with this scheduling policy within the thread
 17004 lists affected.

17005 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
 17006 and *sched_get_priority_min()* functions when SCHED_FIFO is provided as the parameter.
 17007 Conforming implementations shall provide a priority range of at least 32 priorities for this
 17008 policy.

17009 SCHED_RR

17010 Conforming implementations shall include a scheduling policy called the “round robin”
 17011 scheduling policy. This policy shall be identical to the SCHED_FIFO policy with the additional
 17012 condition that when the implementation detects that a running thread has been executing as a
 17013 running thread for a time period of the length returned by the *sched_rr_get_interval()* function or
 17014 longer, the thread shall become the tail of its thread list and the head of that thread list shall be
 17015 removed and made a running thread.

17016 The effect of this policy is to ensure that if there are multiple SCHED_RR threads at the same
 17017 priority, one of them does not monopolize the processor. An application should not rely only on
 17018 the use of SCHED_RR to ensure application progress among multiple threads if the application
 17019 includes threads using the SCHED_FIFO policy at the same or higher priority levels or
 17020 SCHED_RR threads at a higher priority level.

17021 A thread under this policy that is preempted and subsequently resumes execution as a running
 17022 thread completes the unexpired portion of its round robin interval time period.

17023 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_max()*
 17024 and *sched_get_priority_min()* functions when SCHED_RR is provided as the parameter.
 17025 Conforming implementations shall provide a priority range of at least 32 priorities for this

17026 policy.

17027 SCHED_SPORADIC

17028 SS|TSP The functionality described in this section shall be provided on implementations that support
17029 the Process Sporadic Server or Thread Sporadic Server options (and the rest of this section is not
17030 further shaded for these options).

17031 If `_POSIX_SPORADIC_SERVER` or `_POSIX_THREAD_SPORADIC_SERVER` is defined, the
17032 implementation shall include a scheduling policy identified by the value `SCHED_SPORADIC`.

17033 The sporadic server policy is based primarily on two time parameters: the replenishment period
17034 and the available execution capacity. The replenishment period is given by the
17035 `sched_ss_repl_period` member of the `sched_param` structure. The available execution capacity is
17036 initialized to the value given by the `sched_ss_init_budget` member of the same parameter. The
17037 sporadic server policy is identical to the `SCHED_FIFO` policy with some additional conditions
17038 that cause the thread's assigned priority to be switched between the values specified by the
17039 `sched_priority` and `sched_ss_low_priority` members of the `sched_param` structure.

17040 The priority assigned to a thread using the sporadic server scheduling policy is determined in
17041 the following manner: if the available execution capacity is greater than zero and the number of
17042 pending replenishment operations is strictly less than `sched_ss_max_repl`, the thread is assigned
17043 the priority specified by `sched_priority`; otherwise, the assigned priority shall be
17044 `sched_ss_low_priority`. If the value of `sched_priority` is less than or equal to the value of
17045 `sched_ss_low_priority`, the results are undefined. When active, the thread shall belong to the
17046 thread list corresponding to its assigned priority level, according to the mentioned priority
17047 assignment. The modification of the available execution capacity and, consequently of the
17048 assigned priority, is done as follows:

- 17049 1. When the thread at the head of the `sched_priority` list becomes a running thread, its
17050 execution time shall be limited to at most its available execution capacity, plus the
17051 resolution of the execution time clock used for this scheduling policy. This resolution shall
17052 be implementation-defined.
- 17053 2. Each time the thread is inserted at the tail of the list associated with `sched_priority`—
17054 because as a blocked thread it became runnable with priority `sched_priority` or because a
17055 replenishment operation was performed—the time at which this operation is done is
17056 posted as the `activation_time`.
- 17057 3. When the running thread with assigned priority equal to `sched_priority` becomes a
17058 preempted thread, it becomes the head of the thread list for its priority, and the execution
17059 time consumed is subtracted from the available execution capacity. If the available
17060 execution capacity would become negative by this operation, it shall be set to zero.
- 17061 4. When the running thread with assigned priority equal to `sched_priority` becomes a blocked
17062 thread, the execution time consumed is subtracted from the available execution capacity,
17063 and a replenishment operation is scheduled, as described in 6 and 7. If the available
17064 execution capacity would become negative by this operation, it shall be set to zero.
- 17065 5. When the running thread with assigned priority equal to `sched_priority` reaches the limit
17066 imposed on its execution time, it becomes the tail of the thread list for
17067 `sched_ss_low_priority`, the execution time consumed is subtracted from the available
17068 execution capacity (which becomes zero), and a replenishment operation is scheduled, as
17069 described in 6 and 7.
- 17070 6. Each time a replenishment operation is scheduled, the amount of execution capacity to be
17071 replenished, `replenish_amount`, is set equal to the execution time consumed by the thread
17072 since the `activation_time`. The replenishment is scheduled to occur at `activation_time` plus

17073 *sched_ss_repl_period*. If the scheduled time obtained is before the current time, the
 17074 replenishment operation is carried out immediately. Several replenishment operations
 17075 may be pending at the same time, each of which will be serviced at its respective
 17076 scheduled time. With the above rules, the number of replenishment operations
 17077 simultaneously pending for a given thread that is scheduled under the sporadic server
 17078 policy shall not be greater than *sched_ss_max_repl*.

17079 7. A replenishment operation consists of adding the corresponding *replenish_amount* to the
 17080 available execution capacity at the scheduled time. If, as a consequence of this operation,
 17081 the execution capacity would become larger than *sched_ss_initial_budget*, it shall be
 17082 rounded down to a value equal to *sched_ss_initial_budget*. Additionally, if the thread was
 17083 runnable or running, and had assigned priority equal to *sched_ss_low_priority*, then it
 17084 becomes the tail of the thread list for *sched_priority*.

17085 Execution time is defined in XBD [Section 3.117](#) (on page 49). |

17086 For this policy, changing the value of a CPU-time clock via *clock_settime()* shall have no effect on
 17087 its behavior.

17088 For this policy, valid priorities shall be within the range returned by the *sched_get_priority_min()*
 17089 and *sched_get_priority_max()* functions when SCHED_SPORADIC is provided as the parameter.
 17090 Conforming implementations shall provide a priority range of at least 32 distinct priorities for
 17091 this policy.

17092 If the scheduling policy of the target process is either SCHED_FIFO or SCHED_RR, the
 17093 *sched_ss_low_priority*, *sched_ss_repl_period*, and *sched_ss_init* budget members of the *param*
 17094 argument shall have no effect on the scheduling behavior. If the scheduling policy of this process
 17095 is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the effects of these members are
 17096 implementation-defined; this case includes the SCHED_OTHER policy.

17097 **SCHED_OTHER**

17098 Conforming implementations shall include one scheduling policy identified as SCHED_OTHER
 17099 (which may execute identically with either the FIFO or round robin scheduling policy). The
 17100 effect of scheduling threads with the SCHED_OTHER policy in a system in which other threads
 17101 are executing under SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC is implementation-
 17102 defined.

17103 This policy is defined to allow strictly conforming applications to be able to indicate in a
 17104 portable manner that they no longer need a realtime scheduling policy.

17105 For threads executing under this policy, the implementation shall use only priorities within the
 17106 range returned by the *sched_get_priority_max()* and *sched_get_priority_min()* functions when
 17107 SCHED_OTHER is provided as the parameter.

2.8.5 Clocks and Timers

The `<time.h>` header defines the types and manifest constants used by the timing facility.

Time Value Specification Structures

Many of the timing facility functions accept or return time value specifications. A time value structure `timespec` specifies a single time value and includes at least the following members:

Member Type	Member Name	Description
<code>time_t</code>	<code>tv_sec</code>	Seconds.
<code>long</code>	<code>tv_nsec</code>	Nanoseconds.

The `tv_nsec` member is only valid if greater than or equal to zero, and less than the number of nanoseconds in a second (1 000 million). The time interval described by this structure is $(tv_sec * 10^9 + tv_nsec)$ nanoseconds.

A time value structure `itimerspec` specifies an initial timer value and a repetition interval for use by the per-process timer functions. This structure includes at least the following members:

Member Type	Member Name	Description
<code>struct timespec</code>	<code>it_interval</code>	Timer period.
<code>struct timespec</code>	<code>it_value</code>	Timer expiration.

If the value described by `it_value` is non-zero, it indicates the time to or time of the next timer expiration (for relative and absolute timer values, respectively). If the value described by `it_value` is zero, the timer shall be disarmed.

If the value described by `it_interval` is non-zero, it specifies an interval which shall be used in reloading the timer when it expires; that is, a periodic timer is specified. If the value described by `it_interval` is zero, the timer is disarmed after its next expiration; that is, a one-shot timer is specified.

Timer Event Notification Control Block

Per-process timers may be created that notify the process of timer expirations by queuing a realtime extended signal. The `sigevent` structure, defined in the Base Definitions volume of POSIX.1-200x, `<signal.h>`, is used in creating such a timer. The `sigevent` structure contains the signal number and an application-specific data value which shall be used when notifying the calling process of timer expiration events.

Manifest Constants

The following constants are defined in the Base Definitions volume of POSIX.1-200x, `<time.h>`:

<code>CLOCK_REALTIME</code>	The identifier for the system-wide realtime clock.
<code>TIMER_ABSTIME</code>	Flag indicating time is absolute with respect to the clock associated with a timer.
<code>CLOCK_MONOTONIC</code>	The identifier for the system-wide monotonic clock, which is defined as a clock whose value cannot be set via <code>clock_settime()</code> and which cannot have backward clock jumps. The maximum possible clock jump is implementation-defined.

The maximum allowable resolution for `CLOCK_REALTIME` and `CLOCK_MONOTONIC` clocks and all time services based on these clocks is represented by `{_POSIX_CLOCKRES_MIN}` and shall be defined as 20 ms (1/50 of a second). Implementations may support smaller values of

17149 resolution for these clocks to provide finer granularity time bases. The actual resolution
 17150 supported by an implementation for a specific clock is obtained using the `clock_getres()`
 17151 function. If the actual resolution supported for a time service based on one of these clocks differs from the
 17152 resolution supported for that clock, the implementation shall document this difference.

17153 MON The minimum allowable maximum value for `CLOCK_REALTIME` and `CLOCK_MONOTONIC`
 17154 clocks and all absolute time services based on them is the same as that defined by the ISO C
 17155 standard for the `time_t` type. If the maximum value supported by a time service based on one of
 17156 these clocks differs from the maximum value supported by that clock, the implementation shall
 17157 document this difference.

17158 Execution Time Monitoring

17159 CPT If `_POSIX_CPUTIME` is defined, process CPU-time clocks shall be supported in addition to the
 17160 clocks described in [Manifest Constants](#) (on page 484).

17161 TCT If `_POSIX_THREAD_CPUTIME` is defined, thread CPU-time clocks shall be supported.

17162 CPT|TCT CPU-time clocks measure execution or CPU time, which is defined in XBD [Section 3.117](#) (on
 17163 page 49). The mechanism used to measure execution time is described in XBD [Section 4.10](#) (on
 17164 page 98).

17165 CPT If `_POSIX_CPUTIME` is defined, the following constant of the type `clockid_t` is defined in
 17166 `<time.h>`:

17167 `CLOCK_PROCESS_CPUTIME_ID`

17168 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
 17169 interpreted as the identifier of the CPU-time clock associated with the process making the
 17170 function call.

17171 TCT If `_POSIX_THREAD_CPUTIME` is defined, the following constant of the type `clockid_t` is
 17172 defined in `<time.h>`:

17173 `CLOCK_THREAD_CPUTIME_ID`

17174 When this value of the type `clockid_t` is used in a `clock()` or `timer*()` function call, it is
 17175 interpreted as the identifier of the CPU-time clock associated with the thread making the
 17176 function call.

17177 2.9 Threads

17178 This section defines functionality to support multiple flows of control, called “threads”, within a
 17179 process. For the definition of threads, see XBD [Section 3.396](#) (on page 88).

17180 The specific functional areas covered by threads and their scope include:

- 17181 • Thread management: the creation, control, and termination of multiple flows of control in
 17182 the same process under the assumption of a common shared address space
- 17183 • Synchronization primitives optimized for tightly coupled operation of multiple control
 17184 flows in a common, shared address space

17185 **2.9.1 Thread-Safety**

17186 All functions defined by this volume of POSIX.1-200x shall be thread-safe, except that the
17187 following functions⁶ need not be thread-safe.

17188	<i>asctime()</i>	<i>ftw()</i>	<i>getserbyport()</i>	<i>putc_unlocked()</i>
17189	<i>basename()</i>	<i>getc_unlocked()</i>	<i>getservent()</i>	<i>putchar_unlocked()</i>
17190	<i>catgets()</i>	<i>getchar_unlocked()</i>	<i>getutxent()</i>	<i>putenv()</i>
17191	<i>crypt()</i>	<i>getdate()</i>	<i>getutxid()</i>	<i>pututxline()</i>
17192	<i>ctime()</i>	<i>getenv()</i>	<i>getutxline()</i>	<i>rand()</i>
17193	<i>dbm_clearerr()</i>	<i>getgrent()</i>	<i>gmtime()</i>	<i>readdir()</i>
17194	<i>dbm_close()</i>	<i>getgrgid()</i>	<i>hcreate()</i>	<i>setenv()</i>
17195	<i>dbm_delete()</i>	<i>getgrnam()</i>	<i>hdestroy()</i>	<i>setgrent()</i>
17196	<i>dbm_error()</i>	<i>gethostent()</i>	<i>hsearch()</i>	<i>setkey()</i>
17197	<i>dbm_fetch()</i>	<i>getlogin()</i>	<i>inet_ntoa()</i>	<i>setpwent()</i>
17198	<i>dbm_firstkey()</i>	<i>getnetbyaddr()</i>	<i>l64a()</i>	<i>setutxent()</i>
17199	<i>dbm_nextkey()</i>	<i>getnetbyname()</i>	<i>lgamma()</i>	<i>strerror()</i>
17200	<i>dbm_open()</i>	<i>getnetent()</i>	<i>lgammaf()</i>	<i>strsignal()</i>
17201	<i>dbm_store()</i>	<i>getopt()</i>	<i>lgammal()</i>	<i>strtok()</i>
17202	<i>dirname()</i>	<i>getprotobyname()</i>	<i>localeconv()</i>	<i>system()</i>
17203	<i>dllerror()</i>	<i>getprotobynumber()</i>	<i>localtime()</i>	<i>ttyname()</i>
17204	<i>drand48()</i>	<i>getprotoent()</i>	<i>lrand48()</i>	<i>unsetenv()</i>
17205	<i>encrypt()</i>	<i>getpwent()</i>	<i>mrand48()</i>	<i>wcstombs()</i>
17206	<i>endgrent()</i>	<i>getpwnam()</i>	<i>nftw()</i>	<i>wctomb()</i>
17207	<i>endpwent()</i>	<i>getpwuid()</i>	<i>nl_langinfo()</i>	
17208	<i>endutxent()</i>	<i>getserbyname()</i>	<i>ptsname()</i>	

17209 The *ctermid()* and *tmpnam()* functions need not be thread-safe if passed a NULL argument. The
17210 *wctomb()* and *wcsrtombs()* functions need not be thread-safe if passed a NULL *ps* argument.

17211 Implementations shall provide internal synchronization as necessary in order to satisfy this
17212 requirement.

17213 **2.9.2 Thread IDs**

17214 Although implementations may have thread IDs that are unique in a system, applications
17215 should only assume that thread IDs are usable and unique within a single process. The effect of
17216 calling any of the functions defined in this volume of POSIX.1-200x and passing as an argument
17217 the thread ID of a thread from another process is unspecified. A conforming implementation is
17218 free to reuse a thread ID after the thread terminates if it was created with the *detachstate* attribute
17219 set to *PTHREAD_CREATE_DETACHED* or if *pthread_detach()* or *pthread_join()* has been called
17220 for that thread. If a thread is detached, its thread ID is invalid for use as an argument in a call to
17221 *pthread_detach()* or *pthread_join()*.

17222 ⁶ The functions in the table are not shaded to denote applicable options. Individual reference pages should be consulted.

17223 **2.9.3 Thread Mutexes**

17224 A thread that has blocked shall not prevent any unblocked thread that is eligible to use the same
 17225 processing resources from eventually making forward progress in its execution. Eligibility for
 17226 processing resources is determined by the scheduling policy.

17227 A thread shall become the owner of a mutex, *m*, when one of the following occurs:

- 17228 • It returns successfully from *pthread_mutex_lock()* with *m* as the *mutex* argument.
- 17229 • It returns successfully from *pthread_mutex_trylock()* with *m* as the *mutex* argument.
- 17230 • It returns successfully from *pthread_mutex_timedlock()* with *m* as the *mutex* argument.
- 17231 • It returns (successfully or not) from *pthread_cond_wait()* with *m* as the *mutex* argument
 17232 (except as explicitly indicated otherwise for certain errors).
- 17233 • It returns (successfully or not) from *pthread_cond_timedwait()* with *m* as the *mutex*
 17234 argument (except as explicitly indicated otherwise for certain errors).

17235 The thread shall remain the owner of *m* until one of the following occurs:

- 17236 • It executes *pthread_mutex_unlock()* with *m* as the *mutex* argument
- 17237 • It blocks in a call to *pthread_cond_wait()* with *m* as the *mutex* argument.
- 17238 • It blocks in a call to *pthread_cond_timedwait()* with *m* as the *mutex* argument.

17239 The implementation shall behave as if at all times there is at most one owner of any mutex.

17240 A thread that becomes the owner of a mutex is said to have “acquired” the mutex and the mutex
 17241 is said to have become “locked”; when a thread gives up ownership of a mutex it is said to have
 17242 “released” the mutex and the mutex is said to have become “unlocked”.

17243 A problem can occur if a process terminates while one of its threads holds a mutex lock.
 17244 Depending on the mutex type, it might be possible for another thread to unlock the mutex and
 17245 recover the state of the mutex. However, it is difficult to perform this recovery reliably.

17246 Robust mutexes provide a means to enable the implementation to notify other threads in the
 17247 event of a process terminating while one of its threads holds a mutex lock. The next thread that
 17248 acquires the mutex is notified about the termination by the return value [EOWNERDEAD] from
 17249 the locking function. The notified thread can then attempt to recover the state protected by the
 17250 mutex, and if successful mark the state protected by the mutex as consistent by a call to
 17251 *pthread_mutex_consistent()*. If the notified thread is unable to recover the state, it can declare the
 17252 state as not recoverable by a call to *pthread_mutex_unlock()* without a prior call to
 17253 *pthread_mutex_consistent()*.

17254 Whether or not the state protected by a mutex can be recovered is dependent solely on the
 17255 application using robust mutexes. The robust mutex support provided in the implementation
 17256 provides notification only that a mutex owner has terminated while holding a lock, or that the
 17257 state of the mutex is not recoverable.

2.9.4 Thread Scheduling

TPS The functionality described in this section shall be provided on implementations that support the Thread Execution Scheduling option (and the rest of this section is not further shaded for this option).

Thread Scheduling Attributes

In support of the scheduling function, threads have attributes which are accessed through the `pthread_attr_t` thread creation attributes object.

The *contentionscope* attribute defines the scheduling contention scope of the thread to be either `PTHREAD_SCOPE_PROCESS` or `PTHREAD_SCOPE_SYSTEM`.

The *inheritsched* attribute specifies whether a newly created thread is to inherit the scheduling attributes of the creating thread or to have its scheduling values set according to the other scheduling attributes in the `pthread_attr_t` object.

The *schedpolicy* attribute defines the scheduling policy for the thread. The *schedparam* attribute defines the scheduling parameters for the thread. The interaction of threads having different policies within a process is described as part of the definition of those policies.

If the Thread Execution Scheduling option is defined, and the *schedpolicy* attribute specifies one of the priority-based policies defined under this option, the *schedparam* attribute contains the scheduling priority of the thread. A conforming implementation ensures that the priority value in *schedparam* is in the range associated with the scheduling policy when the thread attributes object is used to create a thread, or when the scheduling attributes of a thread are dynamically modified. The meaning of the priority value in *schedparam* is the same as that of *priority*.

TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the *schedparam* attribute supports four new members that are used for the sporadic server scheduling policy. These members are *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The meaning of these attributes is the same as in the definitions that appear under [Section 2.8.4](#) (on page 479).

When a process is created, its single thread has a scheduling policy and associated attributes equal to the policy and attributes of the process. The default scheduling contention scope value is implementation-defined. The default values of other scheduling attributes are implementation-defined.

Thread Scheduling Contention Scope

The scheduling contention scope of a thread defines the set of threads with which the thread competes for use of the processing resources. The scheduling operation selects at most one thread to execute on each processor at any point in time and the thread's scheduling attributes (for example, *priority*), whether under process scheduling contention scope or system scheduling contention scope, are the parameters used to determine the scheduling decision.

The scheduling contention scope, in the context of scheduling a mixed scope environment, affects threads as follows:

- A thread created with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope contends for resources with all other threads in the same scheduling allocation domain relative to their system scheduling attributes. The system scheduling attributes of a thread created with `PTHREAD_SCOPE_SYSTEM` scheduling contention scope are the scheduling attributes with which the thread was created. The system scheduling attributes of a thread created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope are the implementation-defined mapping into system attribute space of the scheduling attributes

- 17303 with which the thread was created.
- 17304 • Threads created with `PTHREAD_SCOPE_PROCESS` scheduling contention scope contend
- 17305 directly with other threads within their process that were created with
- 17306 `PTHREAD_SCOPE_PROCESS` scheduling contention scope. The contention is resolved
- 17307 based on the threads' scheduling attributes and policies. It is unspecified how such threads
- 17308 are scheduled relative to threads in other processes or threads with
- 17309 `PTHREAD_SCOPE_SYSTEM` scheduling contention scope.
- 17310 • Conforming implementations shall support the `PTHREAD_SCOPE_PROCESS` scheduling
- 17311 contention scope, the `PTHREAD_SCOPE_SYSTEM` scheduling contention scope, or both.

17312 Scheduling Allocation Domain

17313 Implementations shall support scheduling allocation domains containing one or more

17314 processors. It should be noted that the presence of multiple processors does not automatically

17315 indicate a scheduling allocation domain size greater than one. Conforming implementations on

17316 multi-processors may map all or any subset of the CPUs to one or multiple scheduling allocation

17317 domains, and could define these scheduling allocation domains on a per-thread, per-process, or

17318 per-system basis, depending on the types of applications intended to be supported by the

17319 implementation. The scheduling allocation domain is independent of scheduling contention

17320 scope, as the scheduling contention scope merely defines the set of threads with which a thread

17321 contends for processor resources, while scheduling allocation domain defines the set of

17322 processors for which it contends. The semantics of how this contention is resolved among

17323 threads for processors is determined by the scheduling policies of the threads.

17324 The choice of scheduling allocation domain size and the level of application control over

17325 scheduling allocation domains is implementation-defined. Conforming implementations may

17326 change the size of scheduling allocation domains and the binding of threads to scheduling

17327 allocation domains at any time.

17328 For application threads with scheduling allocation domains of size equal to one, the scheduling

17329 rules defined for `SCHED_FIFO` and `SCHED_RR` shall be used; see [Scheduling Policies](#) (on page

17330 479). All threads with system scheduling contention scope, regardless of the processes in which

17331 they reside, compete for the processor according to their priorities. Threads with process

17332 scheduling contention scope compete only with other threads with process scheduling

17333 contention scope within their process.

17334 For application threads with scheduling allocation domains of size greater than one, the rules

17335 TSP defined for `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC` shall be used in an

17336 implementation-defined manner. Each thread with system scheduling contention scope

17337 competes for the processors in its scheduling allocation domain in an implementation-defined

17338 manner according to its priority. Threads with process scheduling contention scope are

17339 scheduled relative to other threads within the same scheduling contention scope in the process.

17340 TSP If `_POSIX_THREAD_SPORADIC_SERVER` is defined, the rules defined for `SCHED_SPORADIC`

17341 in [Scheduling Policies](#) (on page 479) shall be used in an implementation-defined manner for

17342 application threads whose scheduling allocation domain size is greater than one.

17343 **Scheduling Documentation**

17344 If `_POSIX_PRIORITY_SCHEDULING` is defined, then any scheduling policies beyond
 17345 TSP `SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`, and `SCHED_SPORADIC`, as well as the effects of
 17346 the scheduling policies indicated by these other values, and the attributes required in order to
 17347 support such a policy, are implementation-defined. Furthermore, the implementation shall
 17348 document the effect of all processor scheduling allocation domain values supported for these
 17349 policies.

17350 **2.9.5 Thread Cancellation**

17351 The thread cancellation mechanism allows a thread to terminate the execution of any other
 17352 thread in the process in a controlled manner. The target thread (that is, the one that is being
 17353 canceled) is allowed to hold cancellation requests pending in a number of ways and to perform
 17354 application-specific cleanup processing when the notice of cancellation is acted upon.

17355 Cancellation is controlled by the cancellation control functions. Each thread maintains its own
 17356 cancelability state. Cancellation may only occur at cancellation points or when the thread is
 17357 asynchronously cancelable.

17358 The thread cancellation mechanism described in this section depends upon programs having set
 17359 *deferred* cancelability state, which is specified as the default. Applications shall also carefully
 17360 follow static lexical scoping rules in their execution behavior. For example, use of `setjmp()`,
 17361 `return`, `goto`, and so on, to leave user-defined cancellation scopes without doing the necessary
 17362 scope pop operation results in undefined behavior.

17363 Use of asynchronous cancelability while holding resources which potentially need to be released
 17364 may result in resource loss. Similarly, cancellation scopes may only be safely manipulated
 17365 (pushed and popped) when the thread is in the *deferred* or *disabled* cancelability states.

17366 **2.9.5.1 Cancelability States**

17367 The cancelability state of a thread determines the action taken upon receipt of a cancellation
 17368 request. The thread may control cancellation in a number of ways.

17369 Each thread maintains its own cancelability state, which may be encoded in two bits:

- 17370 1. Cancelability-Enable: When cancelability is `PTHREAD_CANCEL_DISABLE` (as defined
 17371 in the Base Definitions volume of POSIX.1-200x, `<pthread.h>`), cancellation requests
 17372 against the target thread are held pending. By default, cancelability is set to
 17373 `PTHREAD_CANCEL_ENABLE` (as defined in `<pthread.h>`).
- 17374 2. Cancelability Type: When cancelability is enabled and the cancelability type is
 17375 `PTHREAD_CANCEL_ASYNCHRONOUS` (as defined in `<pthread.h>`), new or pending
 17376 cancellation requests may be acted upon at any time. When cancelability is enabled and
 17377 the cancelability type is `PTHREAD_CANCEL_DEFERRED` (as defined in `<pthread.h>`),
 17378 cancellation requests are held pending until a cancellation point (see below) is reached. If
 17379 cancelability is disabled, the setting of the cancelability type has no immediate effect as all
 17380 cancellation requests are held pending; however, once cancelability is enabled again the
 17381 new type is in effect. The cancelability type is `PTHREAD_CANCEL_DEFERRED` in all
 17382 newly created threads including the thread in which `main()` was first invoked.

17383 2.9.5.2 Cancellation Points

17384 Cancellation points shall occur when a thread is executing the following functions:

17385	<i>accept()</i>	<i>nanosleep()</i>	<i>select()</i>
17386	<i>aio_suspend()</i>	<i>open()</i>	<i>sem_timedwait()</i>
17387	<i>clock_nanosleep()</i>	<i>openat()</i>	<i>sem_wait()</i>
17388	<i>close()</i>	<i>pause()</i>	<i>send()</i>
17389	<i>connect()</i>	<i>poll()</i>	<i>sendmsg()</i>
17390	<i>creat()</i>	<i>pread()</i>	<i>sendto()</i>
17391	<i>fcntl()</i> †	<i>pselect()</i>	<i>sigsuspend()</i>
17392	<i>fdatasync()</i>	<i>pthread_cond_timedwait()</i>	<i>sigtimedwait()</i>
17393	<i>fsync()</i>	<i>pthread_cond_wait()</i>	<i>sigwait()</i>
17394	<i>getmsg()</i>	<i>pthread_join()</i>	<i>sigwaitinfo()</i>
17395	<i>getpmsg()</i>	<i>pthread_testcancel()</i>	<i>sleep()</i>
17396	<i>lockf()</i>	<i>putmsg()</i>	<i>system()</i>
17397	<i>mq_receive()</i>	<i>putpmsg()</i>	<i>tcdrain()</i>
17398	<i>mq_send()</i>	<i>pwrite()</i>	<i>wait()</i>
17399	<i>mq_timedreceive()</i>	<i>read()</i>	<i>waitid()</i>
17400	<i>mq_timedsend()</i>	<i>readv()</i>	<i>waitpid()</i>
17401	<i>msgrcv()</i>	<i>recv()</i>	<i>write()</i>
17402	<i>msgsnd()</i>	<i>recvfrom()</i>	<i>writew()</i>
17403	<i>msync()</i>	<i>recvmsg()</i>	

17404 † When the *cmd* argument is F_SETLKW.

17405 A cancellation point may also occur when a thread is executing the following functions:

17406	<i>access()</i>	<i>fprintf()</i>	<i>getprotobynumber()</i>
17407	<i>asctime()</i>	<i>fputc()</i>	<i>getprotoent()</i>
17408	<i>asctime_r()</i>	<i>fputs()</i>	<i>getpwent()</i>
17409	<i>catclose()</i>	<i>fputwc()</i>	<i>getpwnam()</i>
17410	<i>catgets()</i>	<i>fputws()</i>	<i>getpwnam_r()</i>
17411	<i>catopen()</i>	<i>fread()</i>	<i>getpwuid()</i>
17412	<i>chmod()</i>	<i>freopen()</i>	<i>getpwuid_r()</i>
17413	<i>chown()</i>	<i>fscanf()</i>	<i>gets()</i>
17414	<i>closedir()</i>	<i>fseek()</i>	<i>getserbyname()</i>
17415	<i>closelog()</i>	<i>fseeko()</i>	<i>getserbyport()</i>
17416	<i>ctermid()</i>	<i>fsetpos()</i>	<i>getservent()</i>
17417	<i>ctime()</i>	<i>fstat()</i>	<i>getutxent()</i>
17418	<i>ctime_r()</i>	<i>fstatat()</i>	<i>getutxid()</i>
17419	<i>dbm_close()</i>	<i>ftell()</i>	<i>getutxline()</i>
17420	<i>dbm_delete()</i>	<i>ftello()</i>	<i>getwc()</i>
17421	<i>dbm_fetch()</i>	<i>ftw()</i>	<i>getwchar()</i>
17422	<i>dbm_nextkey()</i>	<i>futimens()</i>	<i>glob()</i>
17423	<i>dbm_open()</i>	<i>fwprintf()</i>	<i>iconv_close()</i>
17424	<i>dbm_store()</i>	<i>fwrite()</i>	<i>iconv_open()</i>
17425	<i>dlclose()</i>	<i>fwscanf()</i>	<i>ioctl()</i>
17426	<i>dlopen()</i>	<i>getaddrinfo()</i>	<i>link()</i>
17427	<i>dprintf()</i>	<i>getc()</i>	<i>linkat()</i>
17428	<i>endgrent()</i>	<i>getc_unlocked()</i>	<i>lio_listio()</i>
17429	<i>endhostent()</i>	<i>getchar()</i>	<i>localtime()</i>
17430	<i>endnetent()</i>	<i>getchar_unlocked()</i>	<i>localtime_r()</i>
17431	<i>endprotoent()</i>	<i>getcwd()</i>	<i>lseek()</i>
17432	<i>endpwent()</i>	<i>getdate()</i>	<i>lstat()</i>
17433	<i>endservent()</i>	<i>getdelim()</i>	<i>mkdir()</i>
17434	<i>endutxent()</i>	<i>getgrent()</i>	<i>mkdirat()</i>
17435	<i>faccessat()</i>	<i>getgrgid()</i>	<i>mkdtemp()</i>
17436	<i>fchmod()</i>	<i>getgrgid_r()</i>	<i>mkfifo()</i>
17437	<i>fchmodat()</i>	<i>getgrnam()</i>	<i>mkfifoat()</i>
17438	<i>fchown()</i>	<i>getgrnam_r()</i>	<i>mknod()</i>
17439	<i>fchownat()</i>	<i>gethostent()</i>	<i>mknodat()</i>
17440	<i>fclose()</i>	<i>gethostid()</i>	<i>mkstemp()</i>
17441	<i>fcntl()</i> †	<i>gethostname()</i>	<i>mktime()</i>
17442	<i>fflush()</i>	<i>getline()</i>	<i>nftw()</i>
17443	<i>fgetc()</i>	<i>getlogin()</i>	<i>opendir()</i>
17444	<i>fgetpos()</i>	<i>getlogin_r()</i>	<i>openlog()</i>
17445	<i>fgets()</i>	<i>getnameinfo()</i>	<i>pathconf()</i>
17446	<i>fgetwc()</i>	<i>getnetbyaddr()</i>	<i>pclose()</i>
17447	<i>fgetws()</i>	<i>getnetbyname()</i>	<i>perror()</i>
17448	<i>fntmsg()</i>	<i>getnetent()</i>	<i>popen()</i>
17449	<i>fork()</i>	<i>getopt()</i> ††	<i>posix_fadvise()</i>
17450	<i>fpathconf()</i>	<i>getprotobyname()</i>	<i>posix_fallocate()</i>

17451	<i>posix_madvise()</i>	<i>putc_unlocked()</i>	<i>strerror()</i>
17452	<i>posix_openpt()</i>	<i>putchar()</i>	<i>strerror_r()</i>
17453	<i>posix_spawn()</i>	<i>putchar_unlocked()</i>	<i>strftime()</i>
17454	<i>posix_spawnnp()</i>	<i>puts()</i>	<i>symlink()</i>
17455	<i>posix_trace_clear()</i>	<i>pututxline()</i>	<i>symlinkat()</i>
17456	<i>posix_trace_close()</i>	<i>putwc()</i>	<i>sync()</i>
17457	<i>posix_trace_create()</i>	<i>putwchar()</i>	<i>syslog()</i>
17458	<i>posix_trace_create_withlog()</i>	<i>readdir()</i>	<i>tmpfile()</i>
17459	<i>posix_trace_eventtypelist_getnext_id()</i>	<i>readdir_r()</i>	<i>tmpnam()</i>
17460	<i>posix_trace_eventtypelist_rewind()</i>	<i>readlink()</i>	<i>ttyname()</i>
17461	<i>posix_trace_flush()</i>	<i>readlinkat()</i>	<i>ttyname_r()</i>
17462	<i>posix_trace_get_attr()</i>	<i>remove()</i>	<i>tzset()</i>
17463	<i>posix_trace_get_filter()</i>	<i>rename()</i>	<i>ungetc()</i>
17464	<i>posix_trace_get_status()</i>	<i>renameat()</i>	<i>ungetwc()</i>
17465	<i>posix_trace_getnext_event()</i>	<i>rewind()</i>	<i>unlink()</i>
17466	<i>posix_trace_open()</i>	<i>rewinddir()</i>	<i>unlinkat()</i>
17467	<i>posix_trace_rewind()</i>	<i>scandir()</i>	<i>utime()</i>
17468	<i>posix_trace_set_filter()</i>	<i>scanf()</i>	<i>utimensat()</i>
17469	<i>posix_trace_shutdown()</i>	<i>seekdir()</i>	<i>utimes()</i>
17470	<i>posix_trace_timedgetnext_event()</i>	<i>semop()</i>	<i>vfprintf()</i>
17471	<i>posix_typed_mem_open()</i>	<i>setgrent()</i>	<i>vfwprintf()</i>
17472	<i>printf()</i>	<i>sethostent()</i>	<i>vprintf()</i>
17473	<i>psiginfo()</i>	<i>setnetent()</i>	<i>vwprintf()</i>
17474	<i>psignal()</i>	<i>setprotoent()</i>	<i>wcsftime()</i>
17475	<i>pthread_rwlock_rdlock()</i>	<i>setpwent()</i>	<i>wordexp()</i>
17476	<i>pthread_rwlock_timedrdlock()</i>	<i>setservent()</i>	<i>wprintf()</i>
17477	<i>pthread_rwlock_timedwrlock()</i>	<i>setutxent()</i>	<i>wscanf()</i>
17478	<i>pthread_rwlock_wrlock()</i>	<i>sigpause()</i>	
17479	<i>putc()</i>	<i>stat()</i>	

17480 An implementation shall not introduce cancellation points into any other functions specified in
17481 this volume of POSIX.1-200x.

17482 The side effects of acting upon a cancellation request while suspended during a call of a function
17483 are the same as the side effects that may be seen in a single-threaded program when a call to a
17484 function is interrupted by a signal and the given function returns [EINTR]. Any such side
17485 effects occur before any cancellation cleanup handlers are called.

17486 Whenever a thread has cancelability enabled and a cancellation request has been made with that
17487 thread as the target, and the thread then calls any function that is a cancellation point (such as
17488 *pthread_testcancel()* or *read()*), the cancellation request shall be acted upon before the function
17489 returns. If a thread has cancelability enabled and a cancellation request is made with the thread
17490 as a target while the thread is suspended at a cancellation point, the thread shall be awakened
17491 and the cancellation request shall be acted upon. It is unspecified whether the cancellation
17492 request is acted upon or whether the cancellation request remains pending and the thread
17493 resumes normal execution if:

- 17494 • The thread is suspended at a cancellation point and the event for which it is waiting occurs
 - 17495 • A specified timeout expired
- 17496 before the cancellation request is acted upon.

17497 † For any value of the *cmd* argument.

17498 †† If *opterr* is non-zero.

17499 2.9.5.3 *Thread Cancellation Cleanup Handlers*

17500 Each thread maintains a list of cancellation cleanup handlers. The programmer uses the
 17501 *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions to place routines on and remove
 17502 routines from this list.

17503 When a cancellation request is acted upon, or when a thread calls *pthread_exit()*, the thread first
 17504 disables cancellation by setting its cancelability state to `PTHREAD_CANCEL_DISABLE` and its
 17505 cancelability type to `PTHREAD_CANCEL_DEFERRED`. The cancelability state shall remain set
 17506 to `PTHREAD_CANCEL_DISABLE` until the thread has terminated. The behavior is undefined if
 17507 a cancellation cleanup handler or thread-specific data destructor routine changes the
 17508 cancelability state to `PTHREAD_CANCEL_ENABLE`.

17509 The routines in the thread's list of cancellation cleanup handlers are invoked one by one in LIFO
 17510 sequence; that is, the last routine pushed onto the list (Last In) is the first to be invoked (First
 17511 Out). When the cancellation cleanup handler for a scope is invoked, the storage for that scope
 17512 remains valid. If the last cancellation cleanup handler returns, thread-specific data destructors (if
 17513 any) associated with thread-specific data keys for which the thread has non-NULL values will
 17514 be run, in unspecified order, as described for *pthread_key_create()*.

17515 After all cancellation cleanup handlers and thread-specific data destructors have returned,
 17516 thread execution is terminated. If the thread has terminated because of a call to *pthread_exit()*,
 17517 the *value_ptr* argument is made available to any threads joining with the target. If the thread has
 17518 terminated by acting on a cancellation request, a status of `PTHREAD_CANCELED` is made
 17519 available to any threads joining with the target. The symbolic constant `PTHREAD_CANCELED`
 17520 expands to a constant expression of type `(void *)` whose value matches no pointer to an object in
 17521 memory nor the value `NULL`.

17522 A side effect of acting upon a cancellation request while in a condition variable wait is that the
 17523 mutex is re-acquired before calling the first cancellation cleanup handler. In addition, the thread
 17524 is no longer considered to be waiting for the condition and the thread shall not have consumed
 17525 any pending condition signals on the condition.

17526 A cancellation cleanup handler cannot exit via *longjmp()* or *siglongjmp()*.

17527 2.9.5.4 *Async-Cancel Safety*

17528 The *pthread_cancel()*, *pthread_setcancelstate()*, and *pthread_setcanceltype()* functions are defined to
 17529 be async-cancel safe.

17530 No other functions in this volume of POSIX.1-200x are required to be async-cancel-safe.

17531 2.9.6 **Thread Read-Write Locks**

17532 Multiple readers, single writer (read-write) locks allow many threads to have simultaneous
 17533 read-only access to data while allowing only one thread to have exclusive write access at any
 17534 given time. They are typically used to protect data that is read more frequently than it is
 17535 changed.

17536 One or more readers acquire read access to the resource by performing a read lock operation on
 17537 the associated read-write lock. A writer acquires exclusive write access by performing a write
 17538 lock operation. Basically, all readers exclude any writers and a writer excludes all readers and
 17539 any other writers.

17540 A thread that has blocked on a read-write lock (for example, has not yet returned from a
 17541 *pthread_rwlock_rdlock()* or *pthread_rwlock_wrlock()* call) shall not prevent any unblocked thread
 17542 that is eligible to use the same processing resources from eventually making forward progress in
 17543 its execution. Eligibility for processing resources shall be determined by the scheduling policy.

17544 Read-write locks can be used to synchronize threads in the current process and other processes if
 17545 they are allocated in memory that is writable and shared among the cooperating processes and
 17546 have been initialized for this behavior.

17547 2.9.7 Thread Interactions with Regular File Operations

17548 All of the following functions shall be atomic with respect to each other in the effects specified in
 17549 POSIX.1-200x when they operate on regular files or symbolic links:

17550	<i>chmod()</i>	<i>fchownat()</i>	<i>lseek()</i>	<i>readv()</i>	<i>unlink()</i>
17551	<i>chown()</i>	<i>fcntl()</i>	<i>lstat()</i>	<i>pwrite()</i>	<i>unlinkat()</i>
17552	<i>close()</i>	<i>fstat()</i>	<i>open()</i>	<i>rename()</i>	<i>utime()</i>
17553	<i>creat()</i>	<i>fstatat()</i>	<i>openat()</i>	<i>renameat()</i>	<i>utimensat()</i>
17554	<i>dup2()</i>	<i>ftruncate()</i>	<i>pread()</i>	<i>stat()</i>	<i>utimes()</i>
17555	<i>fchmod()</i>	<i>lchown()</i>	<i>read()</i>	<i>symlink()</i>	<i>write()</i>
17556	<i>fchmodat()</i>	<i>link()</i>	<i>readlink()</i>	<i>symlinkat()</i>	<i>writev()</i>
17557	<i>fchown()</i>	<i>linkat()</i>	<i>readlinkat()</i>	<i>truncate()</i>	

17558 If two threads each call one of these functions, each call shall either see all of the specified effects
 17559 of the other call, or none of them.

17560 2.9.8 Use of Application-Managed Thread Stacks

17561 An “application-managed thread stack” is a region of memory allocated by the application—for
 17562 example, memory returned by the *malloc()* or *mmmap()* functions—and designated as a stack
 17563 through the act of passing the address and size of the stack, respectively, as the *stackaddr* and
 17564 *stacksize* arguments to *pthread_attr_setstack()*. Application-managed stacks allow the application
 17565 to precisely control the placement and size of a stack.

17566 The application grants to the implementation permanent ownership of and control over the
 17567 application-managed stack when the attributes object in which the *stack* or *stackaddr* attribute has
 17568 been set is used, either by presenting that attribute’s object as the *attr* argument in a call to
 17569 *pthread_create()* that completes successfully, or by storing a pointer to the attributes object in the
 17570 *sigev_notify_attributes* member of a **struct sigevent** and passing that **struct sigevent** to a function
 17571 accepting such argument that completes successfully. The application may thereafter utilize the
 17572 memory within the stack only within the normal context of stack usage within or properly
 17573 synchronized with a thread that has been scheduled by the implementation with stack pointer
 17574 value(s) that are within the range of that stack. In particular, the region of memory cannot be
 17575 freed, nor can it be later specified as the stack for another thread.

17576 When specifying an attributes object with an application-managed stack through the
 17577 *sigev_notify_attributes* member of a **struct sigevent**, the results are undefined if the requested
 17578 signal is generated multiple times (as for a repeating timer).

17579 Until an attributes object in which the *stack* or *stackaddr* attribute has been set is used, the
 17580 application retains ownership of and control over the memory allocated to the stack. It may free
 17581 or reuse the memory as long as it either deletes the attributes object, or before using the
 17582 attributes object replaces the stack by making an additional call to *pthread_attr_setstack()*, that
 17583 was used originally to designate the stack. There is no mechanism to retract the reference to an
 17584 application-managed stack by an existing attributes object.

17585 Once an attributes object with an application-managed stack has been used, that attributes object
 17586 cannot be used again by a subsequent call to *pthread_create()* or any function accepting a **struct**
 17587 **sigevent** with *sigev_notify_attributes* containing a pointer to the attributes object, without
 17588 designating an unused application-managed stack by making an additional call to
 17589 *pthread_attr_setstack()*.

17590 2.10 Sockets

17591 A socket is an endpoint for communication using the facilities described in this section. A socket
 17592 is created with a specific socket type, described in [Section 2.10.6](#) (on page 497), and is associated
 17593 with a specific protocol, detailed in [Section 2.10.3](#). A socket is accessed via a file descriptor
 17594 obtained when the socket is created.

17595 2.10.1 Address Families

17596 All network protocols are associated with a specific address family. An address family provides
 17597 basic services to the protocol implementation to allow it to function within a specific network
 17598 environment. These services may include packet fragmentation and reassembly, routing,
 17599 addressing, and basic transport. An address family is normally comprised of a number of
 17600 protocols, one per socket type. Each protocol is characterized by an abstract socket type. It is not
 17601 required that an address family support all socket types. An address family may contain
 17602 multiple protocols supporting the same socket abstraction.

17603 [Section 2.10.17](#) (on page 503), [Section 2.10.19](#) (on page 504), and [Section 2.10.20](#) (on page 504),
 17604 respectively, describe the use of sockets for local UNIX connections, for Internet protocols based
 17605 on IPv4, and for Internet protocols based on IPv6.

17606 2.10.2 Addressing

17607 An address family defines the format of a socket address. All network addresses are described
 17608 using a general structure, called a **sockaddr**, as defined in the Base Definitions volume of
 17609 POSIX.1-200x, `<sys/socket.h>`. However, each address family imposes finer and more specific
 17610 structure, generally defining a structure with fields specific to the address family. The field
 17611 *sa_family* in the **sockaddr** structure contains the address family identifier, specifying the format
 17612 of the *sa_data* area. The size of the *sa_data* area is unspecified.

17613 2.10.3 Protocols

17614 A protocol supports one of the socket abstractions detailed in [Section 2.10.6](#) (on page 497).
 17615 Selecting a protocol involves specifying the address family, socket type, and protocol number to
 17616 the *socket()* function. Certain semantics of the basic socket abstractions are protocol-specific. All
 17617 protocols are expected to support the basic model for their particular socket type, but may, in
 17618 addition, provide non-standard facilities or extensions to a mechanism.

17619 2.10.4 Routing

17620 Sockets provides packet routing facilities. A routing information database is maintained, which
 17621 is used in selecting the appropriate network interface when transmitting packets.

17622 2.10.5 Interfaces

17623 Each network interface in a system corresponds to a path through which messages can be sent
 17624 and received. A network interface usually has a hardware device associated with it, though
 17625 certain interfaces such as the loopback interface, do not.

2.10.6 Socket Types

17626

17627

17628

17629

17630

RS

A socket is created with a specific type, which defines the communication semantics and which allows the selection of an appropriate communication protocol. Four types are defined: `SOCK_RAW`, `SOCK_STREAM`, `SOCK_SEQPACKET`, and `SOCK_DGRAM`. Implementations may specify additional socket types.

17631

17632

17633

17634

17635

17636

17637

17638

17639

17640

The `SOCK_STREAM` socket type provides reliable, sequenced, full-duplex octet streams between the socket and a peer to which the socket is connected. A socket of type `SOCK_STREAM` must be in a connected state before any data may be sent or received. Record boundaries are not maintained; data sent on a stream socket using output operations of one size may be received using input operations of smaller or larger sizes without loss of data. Data may be buffered; successful return from an output function does not imply that the data has been delivered to the peer or even transmitted from the local system. If data cannot be successfully transmitted within a given time then the connection is considered broken, and subsequent operations shall fail. A `SIGPIPE` signal is raised if a thread sends on a broken stream (one that is no longer connected). Support for an out-of-band data transmission facility is protocol-specific.

17641

17642

17643

17644

17645

17646

17647

The `SOCK_SEQPACKET` socket type is similar to the `SOCK_STREAM` type, and is also connection-oriented. The only difference between these types is that record boundaries are maintained using the `SOCK_SEQPACKET` type. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers parts of more than one record. Record boundaries are visible to the receiver via the `MSG_EOR` flag in the received message flags returned by the `recvmsg()` function. It is protocol-specific whether a maximum record size is imposed.

17648

17649

17650

17651

17652

17653

17654

17655

17656

17657

17658

17659

17660

The `SOCK_DGRAM` socket type supports connectionless data transfer which is not necessarily acknowledged or reliable. Datagrams may be sent to the address specified (possibly multicast or broadcast) in each output operation, and incoming datagrams may be received from multiple sources. The source address of each datagram is available when receiving the datagram. An application may also pre-specify a peer address, in which case calls to output functions that do not specify a peer address shall send to the pre-specified peer. If a peer has been specified, only datagrams from that peer shall be received. A datagram must be sent in a single output operation, and must be received in a single input operation. The maximum size of a datagram is protocol-specific; with some protocols, the limit is implementation-defined. Output datagrams may be buffered within the system; thus, a successful return from an output function does not guarantee that a datagram is actually sent or received. However, implementations should attempt to detect any errors possible before the return of an output function, reporting any error by an unsuccessful return value.

17661

17662

17663

17664

17665

RS

The `SOCK_RAW` socket type is similar to the `SOCK_DGRAM` type. It differs in that it is normally used with communication providers that underlie those used for the other socket types. For this reason, the creation of a socket with type `SOCK_RAW` shall require appropriate privilege. The format of datagrams sent and received with this socket type generally include specific protocol headers, and the formats are protocol-specific and implementation-defined.

17666 2.10.7 Socket I/O Mode

17667 The I/O mode of a socket is described by the `O_NONBLOCK` file status flag which pertains to
 17668 the open file description for the socket. This flag is initially off when a socket is created, but may
 17669 be set and cleared by the use of the `F_SETFL` command of the `fcntl()` function.

17670 When the `O_NONBLOCK` flag is set, functions that would normally block until they are
 17671 complete shall either return immediately with an error, or shall complete asynchronously to the
 17672 execution of the calling process. Data transfer operations (the `read()`, `write()`, `send()`, and `recv()`
 17673 functions) shall complete immediately, transfer only as much as is available, and then return
 17674 without blocking, or return an error indicating that no transfer could be made without blocking.
 17675 The `connect()` function initiates a connection and shall return without blocking when
 17676 `O_NONBLOCK` is set; it shall return the error `[EINPROGRESS]` to indicate that the connection
 17677 was initiated successfully, but that it has not yet completed.

17678 2.10.8 Socket Owner

17679 The owner of a socket is unset when a socket is created. The owner may be set to a process ID or
 17680 process group ID using the `F_SETOWN` command of the `fcntl()` function.

17681 2.10.9 Socket Queue Limits

17682 The transmit and receive queue sizes for a socket are set when the socket is created. The default
 17683 sizes used are both protocol-specific and implementation-defined. The sizes may be changed
 17684 using the `setsockopt()` function.

17685 2.10.10 Pending Error

17686 Errors may occur asynchronously, and be reported to the socket in response to input from the
 17687 network protocol. The socket stores the pending error to be reported to a user of the socket at the
 17688 next opportunity. The error is returned in response to a subsequent `send()`, `recv()`, or `getsockopt()`
 17689 operation on the socket, and the pending error is then cleared.

17690 2.10.11 Socket Receive Queue

17691 A socket has a receive queue that buffers data when it is received by the system until it is
 17692 removed by a receive call. Depending on the type of the socket and the communication provider,
 17693 the receive queue may also contain ancillary data such as the addressing and other protocol data
 17694 associated with the normal data in the queue, and may contain out-of-band or expedited data.
 17695 The limit on the queue size includes any normal, out-of-band data, datagram source addresses,
 17696 and ancillary data in the queue. The description in this section applies to all sockets, even
 17697 though some elements cannot be present in some instances.

17698 The contents of a receive buffer are logically structured as a series of data segments with
 17699 associated ancillary data and other information. A data segment may contain normal data or
 17700 out-of-band data, but never both. A data segment may complete a record if the protocol
 17701 supports records (always true for types `SOCK_SEQPACKET` and `SOCK_DGRAM`). A record
 17702 may be stored as more than one segment; the complete record might never be present in the
 17703 receive buffer at one time, as a portion might already have been returned to the application, and
 17704 another portion might not yet have been received from the communications provider. A data
 17705 segment may contain ancillary protocol data, which is logically associated with the segment.
 17706 Ancillary data is received as if it were queued along with the first normal data octet in the
 17707 segment (if any). A segment may contain ancillary data only, with no normal or out-of-band
 17708 data. For the purposes of this section, a datagram is considered to be a data segment that
 17709 terminates a record, and that includes a source address as a special type of ancillary data. Data

17710 segments are placed into the queue as data is delivered to the socket by the protocol. Normal
 17711 data segments are placed at the end of the queue as they are delivered. If a new segment
 17712 contains the same type of data as the preceding segment and includes no ancillary data, and if
 17713 the preceding segment does not terminate a record, the segments are logically merged into a
 17714 single segment.

17715 The receive queue is logically terminated if an end-of-file indication has been received or a
 17716 connection has been terminated. A segment shall be considered to be terminated if another
 17717 segment follows it in the queue, if the segment completes a record, or if an end-of-file or other
 17718 connection termination has been reported. The last segment in the receive queue shall also be
 17719 considered to be terminated while the socket has a pending error to be reported.

17720 A receive operation shall never return data or ancillary data from more than one segment.

17721 2.10.12 Socket Out-of-Band Data State

17722 The handling of received out-of-band data is protocol-specific. Out-of-band data may be placed
 17723 in the socket receive queue, either at the end of the queue or before all normal data in the queue.
 17724 In this case, out-of-band data is returned to an application program by a normal receive call.
 17725 Out-of-band data may also be queued separately rather than being placed in the socket receive
 17726 queue, in which case it shall be returned only in response to a receive call that requests out-of-
 17727 band data. It is protocol-specific whether an out-of-band data mark is placed in the receive
 17728 queue to demarcate data preceding the out-of-band data and following the out-of-band data. An
 17729 out-of-band data mark is logically an empty data segment that cannot be merged with other
 17730 segments in the queue. An out-of-band data mark is never returned in response to an input
 17731 operation. The *socketmark()* function can be used to test whether an out-of-band data mark is the
 17732 first element in the queue. If an out-of-band data mark is the first element in the queue when an
 17733 input function is called without the MSG_PEEK option, the mark is removed from the queue
 17734 and the following data (if any) is processed as if the mark had not been present.

17735 2.10.13 Connection Indication Queue

17736 Sockets that are used to accept incoming connections maintain a queue of outstanding
 17737 connection indications. This queue is a list of connections that are awaiting acceptance by the
 17738 application; see *listen()*.

17739 2.10.14 Signals

17740 One category of event at the socket interface is the generation of signals. These signals report
 17741 protocol events or process errors relating to the state of the socket. The generation or delivery of
 17742 a signal does not change the state of the socket, although the generation of the signal may have
 17743 been caused by a state change.

17744 The SIGPIPE signal shall be sent to a thread that attempts to send data on a socket that is no
 17745 longer able to send. In addition, the send operation fails with the error [EPIPE].

17746 If a socket has an owner, the SIGURG signal is sent to the owner of the socket when it is notified
 17747 of expedited or out-of-band data. The socket state at this time is protocol-dependent, and the
 17748 status of the socket is specified in [Section 2.10.17](#) (on page 503), [Section 2.10.19](#) (on page 504),
 17749 and [Section 2.10.20](#) (on page 504). Depending on the protocol, the expedited data may or may
 17750 not have arrived at the time of signal generation.

17751 **2.10.15 Asynchronous Errors**

17752 If any of the following conditions occur asynchronously for a socket, the corresponding value
17753 listed below shall become the pending error for the socket:

17754 [ECONNABORTED]

17755 The connection was aborted locally.

17756 [ECONNREFUSED]

17757 For a connection-mode socket attempting a non-blocking connection, the attempt to connect
17758 was forcefully rejected. For a connectionless-mode socket, an attempt to deliver a datagram
17759 was forcefully rejected.

17760 [ECONNRESET]

17761 The peer has aborted the connection.

17762 [EHOSTDOWN]

17763 The destination host has been determined to be down or disconnected.

17764 [EHOSTUNREACH]

17765 The destination host is not reachable.

17766 [EMSGSIZE]

17767 For a connectionless-mode socket, the size of a previously sent datagram prevented
17768 delivery.

17769 [ENETDOWN]

17770 The local network connection is not operational.

17771 [ENETRESET]

17772 The connection was aborted by the network.

17773 [ENETUNREACH]

17774 The destination network is not reachable.

17775 **2.10.16 Use of Options**

17776 There are a number of socket options which either specialize the behavior of a socket or provide
17777 useful information. These options may be set at different protocol levels and are always present
17778 at the uppermost "socket" level.

17779 Socket options are manipulated by two functions, *getsockopt()* and *setsockopt()*. These functions
17780 allow an application program to customize the behavior and characteristics of a socket to
17781 provide the desired effect.

17782 All of the options have default values. The type and meaning of these values is defined by the
17783 protocol level to which they apply. Instead of using the default values, an application program
17784 may choose to customize one or more of the options. However, in the bulk of cases, the default
17785 values are sufficient for the application.

17786 Some of the options are used to enable or disable certain behavior within the protocol modules
17787 (for example, turn on debugging) while others may be used to set protocol-specific information
17788 (for example, IP time-to-live on all the application's outgoing packets). As each of the options is
17789 introduced, its effect on the underlying protocol modules is described.

17790 [Table 2-1](#) (on page 501) shows the value for the socket level.

17791

Table 2-1 Value of Level for Socket Options

17792

17793

Name	Description
SOL_SOCKET	Options are intended for the sockets level.

17794

17795

17796

17797

17798

Table 2-2 lists those options present at the socket level; that is, when the *level* parameter of the *getsockopt()* or *setsockopt()* function is SOL_SOCKET, the types of the option value parameters associated with each option, and a brief synopsis of the meaning of the option value parameter. Unless otherwise noted, each may be examined with *getsockopt()* and set with *setsockopt()* on all types of socket.

17799

Table 2-2 Socket-Level Options

17800

17801

17802

17803

17804

17805

17806

17807

17808

17809

17810

17811

17812

17813

17814

17815

17816

17817

17818

17819

17820

17821

17822

17823

17824

17825

17826

17827

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	int	Non-zero requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DEBUG	int	Non-zero requests debugging in underlying protocol modules.
SO_DONTROUTE	int	Non-zero requests bypass of normal routing; route based on destination address only.
SO_ERROR	int	Requests and clears pending error information on the socket (<i>getsockopt()</i> only).
SO_KEEPALIVE	int	Non-zero requests periodic transmission of keepalive messages (protocol-specific).
SO_LINGER	struct linger	Specify actions to be taken for queued, unsent data on <i>close()</i> : linger on/off and linger time in seconds.
SO_OOINLINE	int	Non-zero requests that out-of-band data be placed into normal data input queue as received.
SO_RCVBUF	int	Size of receive buffer (in bytes).
SO_RCVLOWAT	int	Minimum amount of data to return to application for input operations (in bytes).
SO_RCVTIMEO	struct timeval	Timeout value for a socket receive operation.
SO_REUSEADDR	int	Non-zero requests reuse of local addresses in <i>bind()</i> (protocol-specific).
SO_SNDBUF	int	Size of send buffer (in bytes).
SO_SNDLOWAT	int	Minimum amount of data to send for output operations (in bytes).
SO_SNDTIMEO	struct timeval	Timeout value for a socket send operation.
SO_TYPE	int	Identify socket type (<i>getsockopt()</i> only).

17828

17829

17830

The SO_BROADCAST option requests permission to send broadcast datagrams on the socket. Support for SO_BROADCAST is protocol-specific. The default for SO_BROADCAST is that the ability to send broadcast datagrams on a socket is disabled.

17831

17832

17833

17834

The SO_DEBUG option enables debugging in the underlying protocol modules. This can be useful for tracing the behavior of the underlying protocol modules during normal system operation. The semantics of the debug reports are implementation-defined. The default value for SO_DEBUG is for debugging to be turned off.

17835

The SO_DONTROUTE option requests that outgoing messages bypass the standard routing

17836 facilities. The destination must be on a directly-connected network, and messages are directed to
 17837 the appropriate network interface according to the destination address. It is protocol-specific
 17838 whether this option has any effect and how the outgoing network interface is chosen. Support
 17839 for this option with each protocol is implementation-defined.

17840 The `SO_ERROR` option is used only on `getsockopt()`. When this option is specified, `getsockopt()`
 17841 shall return any pending error on the socket and clear the error status. It shall return a value of 0
 17842 if there is no pending error. `SO_ERROR` may be used to check for asynchronous errors on
 17843 connected connectionless-mode sockets or for other types of asynchronous errors. `SO_ERROR`
 17844 has no default value.

17845 The `SO_KEEPALIVE` option enables the periodic transmission of messages on a connected
 17846 socket. The behavior of this option is protocol-specific. The default value for `SO_KEEPALIVE` is
 17847 zero, specifying that this capability is turned off.

17848 The `SO_LINGER` option controls the action of the interface when unsent messages are queued
 17849 on a socket and a `close()` is performed. The details of this option are protocol-specific. The
 17850 default value for `SO_LINGER` is zero, or off, for the `l_onoff` element of the option value and zero
 17851 seconds for the linger time specified by the `l_linger` element.

17852 The `SO_OOBINLINE` option is valid only on protocols that support out-of-band data. The
 17853 `SO_OOBINLINE` option requests that out-of-band data be placed in the normal data input
 17854 queue as received; it is then accessible using the `read()` or `recv()` functions without the
 17855 `MSG_OOB` flag set. The default for `SO_OOBINLINE` is off; that is, for out-of-band data not to be
 17856 placed in the normal data input queue.

17857 The `SO_RCVBUF` option requests that the buffer space allocated for receive operations on this
 17858 socket be set to the value, in bytes, of the option value. Applications may wish to increase buffer
 17859 size for high volume connections, or may decrease buffer size to limit the possible backlog of
 17860 incoming data. The default value for the `SO_RCVBUF` option value is implementation-defined,
 17861 and may vary by protocol.

17862 The `SO_RCVLOWAT` option sets the minimum number of bytes to process for socket input
 17863 operations. In general, receive calls block until any (non-zero) amount of data is received, then
 17864 return the smaller of the amount available or the amount requested. The default value for
 17865 `SO_RCVLOWAT` is 1, and does not affect the general case. If `SO_RCVLOWAT` is set to a larger
 17866 value, blocking receive calls normally wait until they have received the smaller of the low water
 17867 mark value or the requested amount. Receive calls may still return less than the low water mark
 17868 if an error occurs, a signal is caught, or the type of data next in the receive queue is different
 17869 from that returned (for example, out-of-band data). As mentioned previously, the default value
 17870 for `SO_RCVLOWAT` is 1 byte. It is implementation-defined whether the `SO_RCVLOWAT` option
 17871 can be set.

17872 The `SO_RCVTIMEO` option is an option to set a timeout value for input operations. It accepts a
 17873 `timeval` structure with the number of seconds and microseconds specifying the limit on how
 17874 long to wait for an input operation to complete. If a receive operation has blocked for this much
 17875 time without receiving additional data, it shall return with a partial count or `errno` shall be set to
 17876 `[EWOULDBLOCK]` if no data were received. The default for this option is the value zero, which
 17877 indicates that a receive operation will not time out. It is implementation-defined whether the
 17878 `SO_RCVTIMEO` option can be set.

17879 The `SO_REUSEADDR` option indicates that the rules used in validating addresses supplied in a
 17880 `bind()` should allow reuse of local addresses. Operation of this option is protocol-specific. The
 17881 default value for `SO_REUSEADDR` is off; that is, reuse of local addresses is not permitted.

17882 The `SO_SNDBUF` option requests that the buffer space allocated for send operations on this
 17883 socket be set to the value, in bytes, of the option value. The default value for the `SO_SNDBUF`

17884 option value is implementation-defined, and may vary by protocol.

17885 The SO_SNDBLOWAT option sets the minimum number of bytes to process for socket output
 17886 operations. Most output operations process all of the data supplied by the call, delivering data to
 17887 the protocol for transmission and blocking as necessary for flow control. Non-blocking output
 17888 operations process as much data as permitted subject to flow control without blocking, but
 17889 process no data if flow control does not allow the smaller of the send low water mark value or
 17890 the entire request to be processed. A *select()* operation testing the ability to write to a socket shall
 17891 return true only if the send low water mark could be processed. The default value for
 17892 SO_SNDBLOWAT is implementation-defined and protocol-specific. It is implementation-defined
 17893 whether the SO_SNDBLOWAT option can be set.

17894 The SO_SNDTIMEO option is an option to set a timeout value for the amount of time that an
 17895 output function shall block because flow control prevents data from being sent. As noted in
 17896 Table 2-2 (on page 501), the option value is a **timeval** structure with the number of seconds and
 17897 microseconds specifying the limit on how long to wait for an output operation to complete. If a
 17898 send operation has blocked for this much time, it shall return with a partial count or *errno* set to
 17899 [EWOULDBLOCK] if no data were sent. The default for this option is the value zero, which
 17900 indicates that a send operation will not time out. It is implementation-defined whether the
 17901 SO_SNDTIMEO option can be set.

17902 The SO_TYPE option is used only on *getsockopt()*. When this option is specified, *getsockopt()*
 17903 shall return the type of the socket (for example, SOCK_STREAM). This option is useful to
 17904 servers that inherit sockets on start-up. SO_TYPE has no default value.

17905 2.10.17 Use of Sockets for Local UNIX Connections

17906 Support for UNIX domain sockets is mandatory.

17907 UNIX domain sockets provide process-to-process communication in a single system.

17908 2.10.17.1 Headers

17909 The symbolic constant AF_UNIX defined in the `<sys/socket.h>` header is used to identify the
 17910 UNIX domain address family. The `<sys/un.h>` header contains other definitions used in
 17911 connection with UNIX domain sockets. See XBD Chapter 13 (on page 205).

17912 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to
 17913 accommodate a `sockaddr_un` structure (see the `<sys/un.h>` header defined in XBD Chapter 13,
 17914 on page 205) and shall be aligned at an appropriate boundary so that pointers to it can be cast as
 17915 pointers to `sockaddr_un` structures and used to access the fields of those structures without
 17916 alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_un` structure, the
 17917 `ss_family` field maps onto the `sun_family` field.

17918 2.10.18 Use of Sockets over Internet Protocols

17919 When a socket is created in the Internet family with a protocol value of zero, the implementation
 17920 shall use the protocol listed below for the type of socket created.

17921 SOCK_STREAM IPPROTO_TCP.

17922 SOCK_DGRAM IPPROTO_UDP.

17923 RS SOCK_RAW IPPROTO_RAW.

17924 SOCK_SEQPACKET Unspecified.

17925 RS A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The default
 17926 protocol for type SOCK_RAW shall be identified in the IP header with the value

17927 IPPROTO_RAW. Applications should not use the default protocol when creating a socket with
 17928 type SOCK_RAW, but should identify a specific protocol by value. The ICMP control protocol is
 17929 accessible from a raw socket by specifying a value of IPPROTO_ICMP for protocol.

17930 2.10.19 Use of Sockets over Internet Protocols Based on IPv4

17931 Support for sockets over Internet protocols based on IPv4 is mandatory.

17932 2.10.19.1 Headers

17933 The symbolic constant AF_INET defined in the `<sys/socket.h>` header is used to identify the
 17934 IPv4 Internet address family. The `<netinet/in.h>` header contains other definitions used in
 17935 connection with IPv4 Internet sockets. See XBD Chapter 13 (on page 205).

17936 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to
 17937 accommodate a `sockaddr_in` structure (see the `<netinet/in.h>` header defined in XBD Chapter
 17938 13, on page 205) and shall be aligned at an appropriate boundary so that pointers to it can be
 17939 cast as pointers to `sockaddr_in` structures and used to access the fields of those structures
 17940 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in`
 17941 structure, the `ss_family` field maps onto the `sin_family` field.

17942 2.10.20 Use of Sockets over Internet Protocols Based on IPv6

17943 IP6 This section describes extensions to support sockets over Internet protocols based on IPv6. The
 17944 functionality described in this section shall be provided on implementations that support the
 17945 IPV6 option (and the rest of this section is not further shaded for this option).

17946 To enable smooth transition from IPv4 to IPv6, the features defined in this section may, in certain
 17947 circumstances, also be used in connection with IPv4; see Section 2.10.20.2 (on page 505).

17948 2.10.20.1 Addressing

17949 IPv6 overcomes the addressing limitations of earlier versions by using 128-bit addresses instead
 17950 of 32-bit addresses. The IPv6 address architecture is described in RFC 2373.

17951 There are three kinds of IPv6 address:

17952 Unicast

17953 Identifies a single interface.

17954 A unicast address can be global, link-local (designed for use on a single link), or site-local
 17955 (designed for systems not connected to the Internet). Link-local and site-local addresses
 17956 need not be globally unique.

17957 Anycast

17958 Identifies a set of interfaces such that a packet sent to the address can be delivered to any
 17959 member of the set.

17960 An anycast address is similar to a unicast address; the nodes to which an anycast address is
 17961 assigned must be explicitly configured to know that it is an anycast address.

17962 Multicast

17963 Identifies a set of interfaces such that a packet sent to the address should be delivered to
 17964 every member of the set.

17965 An application can send multicast datagrams by simply specifying an IPv6 multicast
 17966 address in the `address` argument of `sendto()`. To receive multicast datagrams, an application
 17967 must join the multicast group (using `setsockopt()` with `IPV6_JOIN_GROUP`) and must bind
 17968 to the socket the UDP port on which datagrams will be received. Some applications should

17969 also bind the multicast group address to the socket, to prevent other datagrams destined to
17970 that port from being delivered to the socket.

17971 A multicast address can be global, node-local, link-local, site-local, or organization-local.

17972 The following special IPv6 addresses are defined:

17973 Unspecified

17974 An address that is not assigned to any interface and is used to indicate the absence of an
17975 address.

17976 Loopback

17977 A unicast address that is not assigned to any interface and can be used by a node to send
17978 packets to itself.

17979 Two sets of IPv6 addresses are defined to correspond to IPv4 addresses:

17980 IPv4-compatible addresses

17981 These are assigned to nodes that support IPv6 and can be used when traffic is “tunneled”
17982 through IPv4.

17983 IPv4-mapped addresses

17984 These are used to represent IPv4 addresses in IPv6 address format; see [Section 2.10.20.2](#).

17985 Note that the unspecified address and the loopback address must not be treated as
17986 IPv4-compatible addresses.

17987 2.10.20.2 Compatibility with IPv4

17988 The API provides the ability for IPv6 applications to interoperate with applications using IPv4,
17989 by using IPv4-mapped IPv6 addresses. These addresses can be generated automatically by the
17990 `getaddrinfo()` function when the specified host has only IPv4 addresses.

17991 Applications can use `AF_INET6` sockets to open TCP connections to IPv4 nodes, or send UDP
17992 packets to IPv4 nodes, by simply encoding the destination’s IPv4 address as an IPv4-mapped
17993 IPv6 address, and passing that address, within a `sockaddr_in6` structure, in the `connect()`,
17994 `sendto()`, or `sendmsg()` function. When applications use `AF_INET6` sockets to accept TCP
17995 connections from IPv4 nodes, or receive UDP packets from IPv4 nodes, the system shall return
17996 the peer’s address to the application in the `accept()`, `recvfrom()`, `recvmsg()`, or `getpeername()`
17997 function using a `sockaddr_in6` structure encoded this way. If a node has an IPv4 address, then
17998 the implementation shall allow applications to communicate using that address via an
17999 `AF_INET6` socket. In such a case, the address will be represented at the API by the
18000 corresponding IPv4-mapped IPv6 address. Also, the implementation may allow an `AF_INET6`
18001 socket bound to `in6addr_any` to receive inbound connections and packets destined to one of the
18002 node’s IPv4 addresses.

18003 An application can use `AF_INET6` sockets to bind to a node’s IPv4 address by specifying the
18004 address as an IPv4-mapped IPv6 address in a `sockaddr_in6` structure in the `bind()` function. For
18005 an `AF_INET6` socket bound to a node’s IPv4 address, the system shall return the address in the
18006 `getsockname()` function as an IPv4-mapped IPv6 address in a `sockaddr_in6` structure.

18007 2.10.20.3 Interface Identification

18008 Each local interface is assigned a unique positive integer as a numeric index. Indexes start at 1;
18009 zero is not used. There may be gaps so that there is no current interface for a particular positive
18010 index. Each interface also has a unique implementation-defined name.

18011 2.10.20.4 Options

18012 The following options apply at the IPPROTO_IPV6 level:

18013 IPV6_JOIN_GROUP

18014 When set via *setsockopt()*, it joins the application to a multicast group on an interface
 18015 (identified by its index) and addressed by a given multicast address, enabling packets sent
 18016 to that address to be read via the socket. If the interface index is specified as zero, the
 18017 system selects the interface (for example, by looking up the address in a routing table and
 18018 using the resulting interface).

18019 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18020 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18021 IPV6_LEAVE_GROUP

18022 When set via *setsockopt()*, it removes the application from the multicast group on an
 18023 interface (identified by its index) and addressed by a given multicast address.

18024 An attempt to read this option using *getsockopt()* shall result in an [EOPNOTSUPP] error.

18025 The parameter type of this option is a pointer to an **ipv6_mreq** structure.

18026 IPV6_MULTICAST_HOPS

18027 The value of this option is the hop limit for outgoing multicast IPv6 packets sent via the
 18028 socket. Its possible values are the same as those of IPV6_UNICAST_HOPS. If the
 18029 IPV6_MULTICAST_HOPS option is not set, a value of 1 is assumed. This option can be set
 18030 via *setsockopt()* and read via *getsockopt()*.

18031 The parameter type of this option is a pointer to an **int**. (Default value: 1)

18032 IPV6_MULTICAST_IF

18033 The index of the interface to be used for outgoing multicast packets. It can be set via
 18034 *setsockopt()* and read via *getsockopt()*. If the interface index is specified as zero, the system
 18035 selects the interface (for example, by looking up the address in a routing table and using the
 18036 resulting interface).

18037 The parameter type of this option is a pointer to an **unsigned int**. (Default value: 0)

18038 IPV6_MULTICAST_LOOP

18039 This option controls whether outgoing multicast packets should be delivered back to the
 18040 local application when the sending interface is itself a member of the destination multicast
 18041 group. If it is set to 1 they are delivered. If it is set to 0 they are not. Other values result in an
 18042 [EINVAL] error. This option can be set via *setsockopt()* and read via *getsockopt()*.

18043 The parameter type of this option is a pointer to an **unsigned int** which is used as a Boolean
 18044 value. (Default value: 1)

18045 IPV6_UNICAST_HOPS

18046 The value of this option is the hop limit for outgoing unicast IPv6 packets sent via the
 18047 socket. If the option is not set, or is set to -1, the system selects a default value. Attempts to
 18048 set a value less than -1 or greater than 255 shall result in an [EINVAL] error. This option can
 18049 be set via *setsockopt()* and read via *getsockopt()*.

18050 The parameter type of this option is a pointer to an **int**. (Default value: Unspecified)

18051 IPV6_V6ONLY

18052 This socket option restricts AF_INET6 sockets to IPv6 communications only. AF_INET6
 18053 sockets may be used for both IPv4 and IPv6 communications. Some applications may want
 18054 to restrict their use of an AF_INET6 socket to IPv6 communications only. For these
 18055 applications, the IPV6_V6ONLY socket option is defined. When this option is turned on, the

18056 socket can be used to send and receive IPv6 packets only. This is an IPPROTO_IPV6-level
18057 option.

18058 The parameter type of this option is a pointer to an **int** which is used as a Boolean value.
18059 (Default value: 0)

18060 An [EOPNOTSUPP] error shall result if IPV6_JOIN_GROUP or IPV6_LEAVE_GROUP is used
18061 with *getsockopt()*.

18062 2.10.20.5 Headers

18063 The symbolic constant AF_INET6 is defined in the `<sys/socket.h>` header to identify the IPv6
18064 Internet address family. See XBD Chapter 13 (on page 205).

18065 The `sockaddr_storage` structure defined in `<sys/socket.h>` shall be large enough to
18066 accommodate a `sockaddr_in6` structure (see the `<netinet/in.h>` header defined in XBD Chapter
18067 13, on page 205) and shall be aligned at an appropriate boundary so that pointers to it can be
18068 cast as pointers to `sockaddr_in6` structures and used to access the fields of those structures
18069 without alignment problems. When a `sockaddr_storage` structure is cast as a `sockaddr_in6`
18070 structure, the `ss_family` field maps onto the `sin6_family` field.

18071 The `<netinet/in.h>`, `<arpa/inet.h>`, and `<netdb.h>` headers contain other definitions used in
18072 connection with IPv6 Internet sockets; see XBD Chapter 13 (on page 205).

18073 2.11 Tracing

18074 OB TRC This section describes extensions to support tracing of user applications. The functionality
18075 described in this section is dependent on support of the Trace option (and the rest of this section
18076 is not further shaded for this option).

18077 The tracing facilities defined in POSIX.1-200x allow a process to select a set of trace event types,
18078 to activate a trace stream of the selected trace events as they occur in the flow of execution, and
18079 to retrieve the recorded trace events.

18080 The tracing operation relies on three logically different components: the traced process, the
18081 controller process, and the analyzer process. During the execution of the traced process, when a
18082 trace point is reached, a trace event is recorded into the trace streams created for that process in
18083 which the associated trace event type identifier is not being filtered out. The controller process
18084 controls the operation of recording the trace events into the trace stream. It shall be able to:

- 18085 • Initialize the attributes of a trace stream
- 18086 • Create the trace stream (for a specified traced process) using those attributes
- 18087 • Start and stop tracing for the trace stream
- 18088 • Filter the type of trace events to be recorded, if the Trace Event Filter option is supported
- 18089 • Shut a trace stream down

18090 These operations can be done for an active trace stream. The analyzer process retrieves the
18091 traced events either at runtime, when the trace stream has not yet been shut down, but is still
18092 recording trace events; or after opening a trace log that had been previously recorded and shut
18093 down. These three logically different operations can be performed by the same process, or can
18094 be distributed into different processes.

18095 A trace stream identifier can be created by a call to *posix_trace_create()*,
18096 *posix_trace_create_withlog()*, or *posix_trace_open()*. The *posix_trace_create()* and
18097 *posix_trace_create_withlog()* functions should be used by a controller process. The

18098 *posix_trace_open()* should be used by an analyzer process.

18099 The tracing functions can serve different purposes. One purpose is debugging the possibly pre-
 18100 instrumented code, while another is post-mortem fault analysis. These two potential uses differ
 18101 in that the first requires pre-filtering capabilities to avoid overwhelming the trace stream and
 18102 permits focusing on expected information; while the second needs comprehensive trace
 18103 capabilities in order to be able to record all types of information.

18104 The events to be traced belong to two classes:

- 18105 1. User trace events (generated by the application instrumentation)
- 18106 2. System trace events (generated by the operating system)

18107 The trace interface defines several system trace event types associated with control of and
 18108 operation of the trace stream. This small set of system trace events includes the minimum
 18109 required to interpret correctly the trace event information present in the stream. Other desirable
 18110 system trace events for some particular application profile may be implemented and are
 18111 encouraged; for example, process and thread scheduling, signal occurrence, and so on.

18112 Each traced process shall have a mapping of the trace event names to trace event type identifiers
 18113 that have been defined for that process. Each active trace stream shall have a mapping that
 18114 incorporates all the trace event type identifiers predefined by the trace system plus all the
 18115 mappings of trace event names to trace event type identifiers of the processes that are being
 18116 traced into that trace stream. These mappings are defined from the instrumented application by
 18117 calling the *posix_trace_eventid_open()* function and from the controller process by calling the
 18118 *posix_trace_trid_eventid_open()* function. For a pre-recorded trace stream, the list of trace event
 18119 types is obtained from the pre-recorded trace log.

18120 The last data modification and file status change timestamps of a file associated with an active |
 18121 trace stream shall be marked for update every time any of the tracing operations modifies that |
 18122 file. |

18123 The last data access timestamp of a file associated with a trace stream shall be marked for |
 18124 update every time any of the tracing operations causes data to be read from that file. |

18125 Results are undefined if the application performs any operation on a file descriptor associated
 18126 with an active or pre-recorded trace stream until *posix_trace_shutdown()* or *posix_trace_close()* is
 18127 called for that trace stream. Results are also undefined if the analyzer process and the traced
 18128 process do not share the same programming environment (see *c99*, Programming Environments
 18129 in the Shell and Utilities volume of POSIX.1-200x.

18130 The main purpose of this option is to define a complete set of functions and concepts that allow
 18131 a conforming application to be traced from creation to termination, whatever its realtime
 18132 constraints and properties.

18133 **2.11.1 Tracing Data Definitions**18134 **2.11.1.1 Structures**

18135 The **<trace.h>** header shall define the *posix_trace_status_info* and *posix_trace_event_info* structures
 18136 described below. Implementations may add extensions to these structures.

18137 **posix_trace_status_info Structure**

18138 To facilitate control of a trace stream, information about the current state of an active trace
 18139 stream can be obtained dynamically. This structure is returned by a call to the
 18140 *posix_trace_get_status()* function.

18141 The **posix_trace_status_info** structure defined in **<trace.h>** shall contain at least the following
 18142 members:

Member Type	Member Name	Description
int	<i>posix_stream_status</i>	The operating mode of the trace stream.
int	<i>posix_stream_full_status</i>	The full status of the trace stream.
int	<i>posix_stream_overrun_status</i>	Indicates whether trace events were lost in the trace stream.

18148 If the Trace Log option is supported in addition to the Trace option, the **posix_trace_status_info**
 18149 structure defined in **<trace.h>** shall contain at least the following additional members:

Member Type	Member Name	Description
int	<i>posix_stream_flush_status</i>	Indicates whether a flush is in progress.
int	<i>posix_stream_flush_error</i>	Indicates whether any error occurred during the last flush operation.
int	<i>posix_log_overrun_status</i>	Indicates whether trace events were lost in the trace log.
int	<i>posix_log_full_status</i>	The full status of the trace log.

18157 The *posix_stream_status* member indicates the operating mode of the trace stream and shall have
 18158 one of the following values defined by manifest constants in the **<trace.h>** header:

18159 **POSIX_TRACE_RUNNING**

18160 Tracing is in progress; that is, the trace stream is accepting trace events.

18161 **POSIX_TRACE_SUSPENDED**

18162 The trace stream is not accepting trace events. The tracing operation has not yet started or
 18163 has stopped, either following a *posix_trace_stop()* function call or because the trace resources
 18164 are exhausted.

18165 The *posix_stream_full_status* member indicates the full status of the trace stream, and it shall have
 18166 one of the following values defined by manifest constants in the **<trace.h>** header:

18167 **POSIX_TRACE_FULL**

18168 The space in the trace stream for trace events is exhausted.

18169 **POSIX_TRACE_NOT_FULL**

18170 There is still space available in the trace stream.

18171 The combination of the *posix_stream_status* and *posix_stream_full_status* members also indicates
 18172 the actual status of the stream. The status shall be interpreted as follows:

- 18173 POSIX_TRACE_RUNNING and POSIX_TRACE_NOT_FULL
 18174 This status combination indicates that tracing is in progress, and there is space available for
 18175 recording more trace events.
- 18176 POSIX_TRACE_RUNNING and POSIX_TRACE_FULL
 18177 This status combination indicates that tracing is in progress and that the trace stream is full
 18178 of trace events. This status combination cannot occur unless the *stream-full-policy* is set to
 18179 POSIX_TRACE_LOOP. The trace stream contains trace events recorded during a moving
 18180 time window of prior trace events, and some older trace events may have been overwritten
 18181 and thus lost.
- 18182 POSIX_TRACE_SUSPENDED and POSIX_TRACE_NOT_FULL
 18183 This status combination indicates that tracing has not yet been started, has been stopped by
 18184 the *posix_trace_stop()* function, or has been cleared by the *posix_trace_clear()* function.
- 18185 POSIX_TRACE_SUSPENDED and POSIX_TRACE_FULL
 18186 This status combination indicates that tracing has been stopped by the implementation
 18187 because the *stream-full-policy* attribute was POSIX_TRACE_UNTIL_FULL and trace
 18188 resources were exhausted, or that the trace stream was stopped by the function
 18189 *posix_trace_stop()* at a time when trace resources were exhausted.
- 18190 The *posix_stream_overrun_status* member indicates whether trace events were lost in the trace
 18191 stream, and shall have one of the following values defined by manifest constants in the
 18192 **<trace.h>** header:
- 18193 POSIX_TRACE_OVERRUN
 18194 At least one trace event was lost and thus was not recorded in the trace stream.
- 18195 POSIX_TRACE_NO_OVERRUN
 18196 No trace events were lost.
- 18197 When the corresponding trace stream is created, the *posix_stream_overrun_status* member shall be
 18198 set to POSIX_TRACE_NO_OVERRUN.
- 18199 Whenever an overrun occurs, the *posix_stream_overrun_status* member shall be set to
 18200 POSIX_TRACE_OVERRUN.
- 18201 An overrun occurs when:
- 18202 • The policy is POSIX_TRACE_LOOP and a recorded trace event is overwritten.
 - 18203 • The policy is POSIX_TRACE_UNTIL_FULL and the trace stream is full when a trace event
 18204 is generated.
 - 18205 • If the Trace Log option is supported, the policy is POSIX_TRACE_FLUSH and at least one
 18206 trace event is lost while flushing the trace stream to the trace log.
- 18207 The *posix_stream_overrun_status* member is reset to zero after its value is read.
- 18208 If the Trace Log option is supported in addition to the Trace option, the *posix_stream_flush_status*,
 18209 *posix_stream_flush_error*, *posix_log_overrun_status*, and *posix_log_full_status* members are defined
 18210 as follows; otherwise, they are undefined.
- 18211 The *posix_stream_flush_status* member indicates whether a flush operation is being performed
 18212 and shall have one of the following values defined by manifest constants in the header
 18213 **<trace.h>**:
- 18214 POSIX_TRACE_FLUSHING
 18215 The trace stream is currently being flushed to the trace log.

18216 POSIX_TRACE_NOT_FLUSHING
18217 No flush operation is in progress.

18218 The *posix_stream_flush_status* member shall be set to POSIX_TRACE_FLUSHING if a flush
18219 operation is in progress either due to a call to the *posix_trace_flush()* function (explicit or caused
18220 by a trace stream shutdown operation) or because the trace stream has become full with the
18221 *stream-full-policy* attribute set to POSIX_TRACE_FLUSH. The *posix_stream_flush_status* member
18222 shall be set to POSIX_TRACE_NOT_FLUSHING if no flush operation is in progress.

18223 The *posix_stream_flush_error* member shall be set to zero if no error occurred during flushing. If
18224 an error occurred during a previous flushing operation, the *posix_stream_flush_error* member
18225 shall be set to the value of the first error that occurred. If more than one error occurs while
18226 flushing, error values after the first shall be discarded. The *posix_stream_flush_error* member is
18227 reset to zero after its value is read.

18228 The *posix_log_outrun_status* member indicates whether trace events were lost in the trace log,
18229 and shall have one of the following values defined by manifest constants in the **<trace.h>**
18230 header:

18231 POSIX_TRACE_OVERRUN
18232 At least one trace event was lost.

18233 POSIX_TRACE_NO_OVERRUN
18234 No trace events were lost.

18235 When the corresponding trace stream is created, the *posix_log_outrun_status* member shall be set
18236 to POSIX_TRACE_NO_OVERRUN. Whenever an overrun occurs, this status shall be set to
18237 POSIX_TRACE_OVERRUN. The *posix_log_outrun_status* member is reset to zero after its value
18238 is read.

18239 The *posix_log_full_status* member indicates the full status of the trace log, and it shall have one of
18240 the following values defined by manifest constants in the **<trace.h>** header:

18241 POSIX_TRACE_FULL
18242 The space in the trace log is exhausted.

18243 POSIX_TRACE_NOT_FULL
18244 There is still space available in the trace log.

18245 The *posix_log_full_status* member is only meaningful if the *log-full-policy* attribute is either
18246 POSIX_TRACE_UNTIL_FULL or POSIX_TRACE_LOOP.

18247 For an active trace stream without log, that is created by the *posix_trace_create()* function, the
18248 *posix_log_outrun_status* member shall be set to POSIX_TRACE_NO_OVERRUN and the
18249 *posix_log_full_status* member shall be set to POSIX_TRACE_NOT_FULL.

18250 **posix_trace_event_info** Structure

18251 The trace event structure **posix_trace_event_info** contains the information for one recorded
18252 trace event. This structure is returned by the set of functions *posix_trace_getnext_event()*,
18253 *posix_trace_timedgetnext_event()*, and *posix_trace_trygetnext_event()*.

18254 The **posix_trace_event_info** structure defined in **<trace.h>** shall contain at least the following
18255 members:

Member Type	Member Name	Description
trace_event_id_t	<i>posix_event_id</i>	Trace event type identification.
pid_t	<i>posix_pid</i>	Process ID of the process that generated the trace event.
void *	<i>posix_prog_address</i>	Address at which the trace point was invoked.
int	<i>posix_truncation_status</i>	Status about the truncation of the data associated with this trace event.
struct timespec	<i>posix_timestamp</i>	Time at which the trace event was generated.

In addition, the **posix_trace_event_info** structure defined in `<trace.h>` shall contain the following additional member:

Member Type	Member Name	Description
pthread_t	<i>posix_thread_id</i>	Thread ID of the thread that generated the trace event.

The *posix_event_id* member represents the identification of the trace event type and its value is not directly defined by the user. This identification is returned by a call to one of the following functions: *posix_trace_trid_eventid_open()*, *posix_trace_eventtypelist_getnext_id()*, or *posix_trace_eventid_open()*. The name of the trace event type can be obtained by calling *posix_trace_eventid_get_name()*.

The *posix_pid* is the process identifier of the traced process which generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any process, the *posix_pid* member shall be set to zero.

For a user trace event, the *posix_prog_address* member is the process mapped address of the point at which the associated call to the *posix_trace_event()* function was made. For a system trace event, if the trace event is caused by a system service explicitly called by the application, the *posix_prog_address* member shall be the address of the process at the point where the call to that system service was made.

The *posix_truncation_status* member defines whether the data associated with a trace event has been truncated at the time the trace event was generated, or at the time the trace event was read from the trace stream, or (if the Trace Log option is supported) from the trace log (see the *event* argument from the *posix_trace_getnext_event()* function). The *posix_truncation_status* member shall have one of the following values defined by manifest constants in the `<trace.h>` header:

POSIX_TRACE_NOT_TRUNCATED

All the traced data is available.

POSIX_TRACE_TRUNCATED_RECORD

Data was truncated at the time the trace event was generated.

POSIX_TRACE_TRUNCATED_READ

Data was truncated at the time the trace event was read from a trace stream or a trace log because the reader's buffer was too small. This truncation status overrides the POSIX_TRACE_TRUNCATED_RECORD status.

The *posix_timestamp* member shall be the time at which the trace event was generated. The clock used is implementation-defined, but the resolution of this clock can be retrieved by a call to the *posix_trace_attr_getclockres()* function.

The *posix_thread_id* member is the identifier of the thread that generated the trace event. If the *posix_event_id* member is one of the implementation-defined system trace events and that trace event is not associated with any thread, the *posix_thread_id* member shall be set to zero.

18302 2.11.1.2 Trace Stream Attributes

18303 Trace streams have attributes that compose the **posix_trace_attr_t** trace stream attributes object.
 18304 This object shall contain at least the following attributes:

- 18305 • The *generation-version* attribute identifies the origin and version of the trace system.
- 18306 • The *trace-name* attribute is a character string defined by the trace controller, and that
 18307 identifies the trace stream.
- 18308 • The *creation-time* attribute represents the time of the creation of the trace stream.
- 18309 • The *clock-resolution* attribute defines the clock resolution of the clock used to generate
 18310 timestamps.
- 18311 • The *stream-min-size* attribute defines the minimum size in bytes of the trace stream strictly
 18312 reserved for the trace events.
- 18313 • The *stream-full-policy* attribute defines the policy followed when the trace stream is full; its
 18314 value is `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or `POSIX_TRACE_FLUSH`.
- 18315 • The *max-data-size* attribute defines the maximum record size in bytes of a trace event.

18316 In addition, if the Trace option and the Trace Inherit option are both supported, the
 18317 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
 18318 attributes:

- 18319 • The *inheritance* attribute specifies whether a newly created trace stream will inherit tracing
 18320 in its parent's process trace stream. It is either `POSIX_TRACE_INHERITED` or
 18321 `POSIX_TRACE_CLOSE_FOR_CHILD`.

18322 In addition, if the Trace option and the Trace Log option are both supported, the
 18323 **posix_trace_attr_t** trace stream creation attributes object shall contain at least the following
 18324 attribute:

- 18325 • If the file type corresponding to the trace log supports the `POSIX_TRACE_LOOP` or the
 18326 `POSIX_TRACE_UNTIL_FULL` policies, the *log-max-size* attribute defines the maximum
 18327 size in bytes of the trace log associated with an active trace stream. Other stream data—for
 18328 example, trace attribute values—shall not be included in this size.
- 18329 • The *log-full-policy* attribute defines the policy of a trace log associated with an active trace
 18330 stream to be `POSIX_TRACE_LOOP`, `POSIX_TRACE_UNTIL_FULL`, or
 18331 `POSIX_TRACE_APPEND`.

18332 2.11.2 Trace Event Type Definitions

18333 2.11.2.1 System Trace Event Type Definitions

18334 The following system trace event types, defined in the **<trace.h>** header, track the invocation of
 18335 the trace operations:

- 18336 • `POSIX_TRACE_START` shall be associated with a trace start operation.
- 18337 • `POSIX_TRACE_STOP` shall be associated with a trace stop operation.
- 18338 • If the Trace Event Filter option is supported, `POSIX_TRACE_FILTER` shall be associated
 18339 with a trace event type filter change operation.

18340 The following system trace event types, defined in the **<trace.h>** header, report operational trace
 18341 events:

- 18342 • POSIX_TRACE_OVERFLOW shall mark the beginning of a trace overflow condition.
- 18343 • POSIX_TRACE_RESUME shall mark the end of a trace overflow condition.
- 18344 • If the Trace Log option is supported, POSIX_TRACE_FLUSH_START shall mark the
- 18345 beginning of a flush operation.
- 18346 • If the Trace Log option is supported, POSIX_TRACE_FLUSH_STOP shall mark the end of
- 18347 a flush operation.
- 18348 • If an implementation-defined trace error condition is reported, it shall be marked
- 18349 POSIX_TRACE_ERROR.

18350 The interpretation of a trace stream or a trace log by a trace analyzer process relies on the
 18351 information recorded for each trace event, and also on system trace events that indicate the
 18352 invocation of trace control operations and trace system operational trace events.

18353 The POSIX_TRACE_START and POSIX_TRACE_STOP trace events specify the time windows
 18354 during which the trace stream is running.

- 18355 • The POSIX_TRACE_STOP trace event with an associated data that is equal to zero
- 18356 indicates a call of the function *posix_trace_stop()*.
- 18357 • The POSIX_TRACE_STOP trace event with an associated data that is different from zero
- 18358 indicates an automatic stop of the trace stream (see the definition of the
- 18359 *posix_trace_attr_getstreamfullpolicy()* function in *posix_trace_attr_getinherited()*).

18360 The POSIX_TRACE_FILTER trace event indicates that a trace event type filter value changed
 18361 while the trace stream was running.

18362 The POSIX_TRACE_ERROR serves to inform the analyzer process that an implementation-
 18363 defined internal error of the trace system occurred.

18364 The POSIX_TRACE_OVERFLOW trace event shall be reported with a timestamp equal to the
 18365 timestamp of the first trace event overwritten. This is an indication that some generated trace
 18366 events have been lost.

18367 The POSIX_TRACE_RESUME trace event shall be reported with a timestamp equal to the
 18368 timestamp of the first valid trace event reported after the overflow condition ends and shall be
 18369 reported before this first valid trace event. This is an indication that the trace system is reliably
 18370 recording trace events after an overflow condition.

18371 Each of these trace event types shall be defined by a constant trace event name and a
 18372 **trace_event_id_t** constant; trace event data is associated with some of these trace events.

18373 If the Trace option is supported and the Trace Event Filter option and the Trace Log option are
 18374 not supported, the following predefined system trace events in [Table 2-3](#) (on page 515) shall be
 18375 defined:

18376

Table 2-3 Trace Option: System Trace Events

18377

18378

18379

18380

18381

18382

18383

18384

18385

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

18386

18387

18388

If the Trace option and the Trace Event Filter option are both supported, and if the Trace Log option is not supported, the following predefined system trace events in [Table 2-4](#) shall be defined:

18389

Table 2-4 Trace and Trace Event Filter Options: System Trace Events

18390

18391

18392

18393

18394

18395

18396

18397

18398

18399

18400

18401

18402

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error
		int
posix_trace_start	POSIX_TRACE_START	event_filter
		trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto
		int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter
		new_event_filter
		trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.

18403

18404

18405

If the Trace option and the Trace Log option are both supported, and if the Trace Event Filter option is not supported, the following predefined system trace events in [Table 2-5](#) (on page 516) shall be defined:

Table 2-5 Trace and Trace Log Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	None.
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

If the Trace option, the Trace Event Filter option, and the Trace Log option are all supported, the following predefined system trace events in Table 2-6 shall be defined:

Table 2-6 Trace, Trace Log, and Trace Event Filter Options: System Trace Events

Event Name	Constant	Associated Data
		Data Type
posix_trace_error	POSIX_TRACE_ERROR	error int
posix_trace_start	POSIX_TRACE_START	event_filter trace_event_set_t
posix_trace_stop	POSIX_TRACE_STOP	auto int
posix_trace_filter	POSIX_TRACE_FILTER	old_event_filter new_event_filter trace_event_set_t
posix_trace_overflow	POSIX_TRACE_OVERFLOW	None.
posix_trace_resume	POSIX_TRACE_RESUME	None.
posix_trace_flush_start	POSIX_TRACE_FLUSH_START	None.
posix_trace_flush_stop	POSIX_TRACE_FLUSH_STOP	None.

2.11.2.2 User Trace Event Type Definitions

The user trace event `POSIX_TRACE_UNNAMED_USEREVENT` is defined in the `<trace.h>` header. If the limit of per-process user trace event names represented by `{TRACE_USER_EVENT_MAX}` has already been reached, this predefined user event shall be returned when the application tries to register more events than allowed. The data associated with this trace event is application-defined.

The following predefined user trace event in Table 2-7 (on page 517) shall be defined:

18443 **Table 2-7** Trace Option: User Trace Event

Event Name	Constant
posix_trace_unnamed_userevent	POSIX_TRACE_UNNAMED_USEREVENT

18446 **2.11.3 Trace Functions**

18447 The trace interface is built and structured to improve portability through use of trace data of
 18448 opaque type. The object-oriented approach for the manipulation of trace attributes and trace
 18449 event type identifiers requires definition of many constructor and selector functions which
 18450 operate on these opaque types. Also, the trace interface must support several different tracing
 18451 roles. To facilitate reading the trace interface, the trace functions are grouped into small
 18452 functional sets supporting the three different roles:

- 18453 • A trace controller process requires functions to set up and customize all the resources
 18454 needed to run a trace stream, including:
 - 18455 — Attribute initialization and destruction (*posix_trace_attr_init()*)
 - 18456 — Identification information manipulation (*posix_trace_attr_getgenversion()*)
 - 18457 — Trace system behavior modification (*posix_trace_attr_getinherited()*)
 - 18458 — Trace stream and trace log size set (*posix_trace_attr_getmaxusereventsize()*)
 - 18459 — Trace stream creation, flush, and shutdown (*posix_trace_create()*)
 - 18460 — Trace stream and trace log clear (*posix_trace_clear()*)
 - 18461 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
 - 18462 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
 - 18463 — Trace event type set manipulation (*posix_trace_eventset_empty()*)
 - 18464 — Trace event type filter set (*posix_trace_set_filter()*)
 - 18465 — Trace stream start and stop (*posix_trace_start()*)
 - 18466 — Trace stream information and status read (*posix_trace_get_attr()*)
- 18467 • A traced process requires functions to instrument trace points:
 - 18468 — Trace event type identifiers definition and trace points insertion (*posix_trace_event()*)
- 18469 • A trace analyzer process requires functions to retrieve information from a trace stream and
 18470 trace log:
 - 18471 — Identification information read (*posix_trace_attr_getgenversion()*)
 - 18472 — Trace system behavior information read (*posix_trace_attr_getinherited()*)
 - 18473 — Trace stream and trace log size get (*posix_trace_attr_getmaxusereventsize()*)
 - 18474 — Trace event type identifier manipulation (*posix_trace_trid_eventid_open()*)
 - 18475 — Trace event type identifier list exploration (*posix_trace_eventtypelist_getnext_id()*)
 - 18476 — Trace log open, rewind, and close (*posix_trace_open()*)
 - 18477 — Trace stream information and status read (*posix_trace_get_attr()*)

18478 — Trace event read (*posix_trace_getnext_event()*)

18479 2.12 Data Types

18480 2.12.1 Defined Types

18481 All of the data types used by various functions are defined by the implementation. The
 18482 following table describes some of these types. Other types referenced in the description of a
 18483 function, not mentioned here, can be found in the appropriate header for that function.

18484	Defined Type	Description
18485	cc_t	Type used for terminal special characters.
18486	clock_t	Integer or real-floating type used for processor times, as defined in 18487 the ISO C standard.
18488	clockid_t	Used for clock ID type in some timer functions.
18489	dev_t	Arithmetic type used for device numbers.
18490	DIR	Type representing a directory stream.
18491	div_t	Structure type returned by the <i>div()</i> function.
18492	FILE	Structure containing information about a file.
18493	glob_t	Structure type used in pathname pattern matching.
18494	fpos_t	Type containing all information needed to specify uniquely every 18495 position within a file.
18496	gid_t	Integer type used for group IDs.
18497	iconv_t	Type used for conversion descriptors.
18498	id_t	Integer type used as a general identifier; can be used to contain 18499 at least the largest of a pid_t , uid_t , or gid_t .
18500	ino_t	Unsigned integer type used for file serial numbers.
18501	key_t	Arithmetic type used for XSI interprocess communication.
18502	ldiv_t	Structure type returned by the <i>ldiv()</i> function.
18503	mode_t	Integer type used for file attributes.
18504	mqd_t	Used for message queue descriptors.
18505	nfds_t	Integer type used for the number of file descriptors.
18506	nlink_t	Integer type used for link counts.
18507	off_t	Signed integer type used for file sizes.
18508	pid_t	Signed integer type used for process and process group IDs.
18509	pthread_attr_t	Used to identify a thread attribute object.
18510	pthread_cond_t	Used for condition variables.
18511	pthread_condattr_t	Used to identify a condition attribute object.
18512	pthread_key_t	Used for thread-specific data keys.
18513	pthread_mutex_t	Used for mutexes.
18514	pthread_mutexattr_t	Used to identify a mutex attribute object.
18515	pthread_once_t	Used for dynamic package initialization.
18516	pthread_rwlock_t	Used for read-write locks.
18517	pthread_rwlockattr_t	Used for read-write lock attributes.
18518	pthread_t	Used to identify a thread.
18519	ptrdiff_t	Signed integer type of the result of subtracting two pointers.
18520	regex_t	Structure type used in regular expression matching.
18521	regmatch_t	Structure type used in regular expression matching.
18522	rlim_t	Unsigned integer type used for limit values, to which objects of

Defined Type	Description
18523	
18524	type int and off_t can be cast without loss of value.
18525	sem_t Type used in performing semaphore operations.
18526	sig_atomic_t Integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.
18527	
18528	sigset_t Integer or structure type of an object used to represent sets of signals.
18529	
18530	size_t Unsigned integer type used for size of objects.
18531	speed_t Type used for terminal baud rates.
18532	ssize_t Signed integer type used for a count of bytes or an error indication.
18533	
18534	suseconds_t Signed integer type used for time in microseconds.
18535	tflag_t Type used for terminal modes.
18536	time_t Integer or real-floating type used for time in seconds, as defined in the ISO C standard.
18537	
18538	timer_t Used for timer ID returned by the <i>timer_create()</i> function.
18539	uid_t Integer type used for user IDs.
18540	va_list Type used for traversing variable argument lists.
18541	wchar_t Integer type whose range of values can represent distinct codes for all members of the largest extended character set specified by the supported locales.
18542	
18543	
18544	wctype_t Scalar type which represents a character class descriptor.
18545	wint_t Integer type capable of storing any valid value of wchar_t or WEOF.
18546	
18547	wordexp_t Structure type used in word expansion.

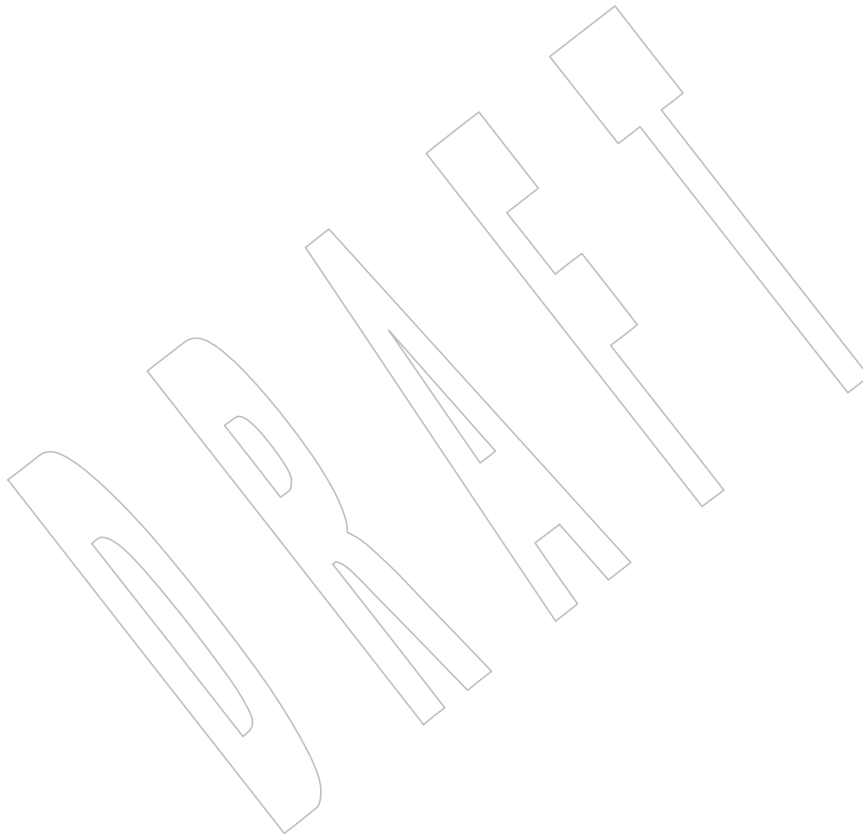
2.12.2 The char Type

The type **char** is defined as a single byte; see XBD [Chapter 3](#) (on page 33) (Byte and Character).

2.12.3 Pointer Types

All function pointer types shall have the same representation as the type pointer to **void**. Conversion of a function pointer to **void *** shall not alter the representation. A **void *** value resulting from such a conversion can be converted back to the original function pointer type, using an explicit cast, without loss of information.


Note: The ISO C standard does not require this, but it is required for POSIX conformance.



18556

Chapter 3

18557

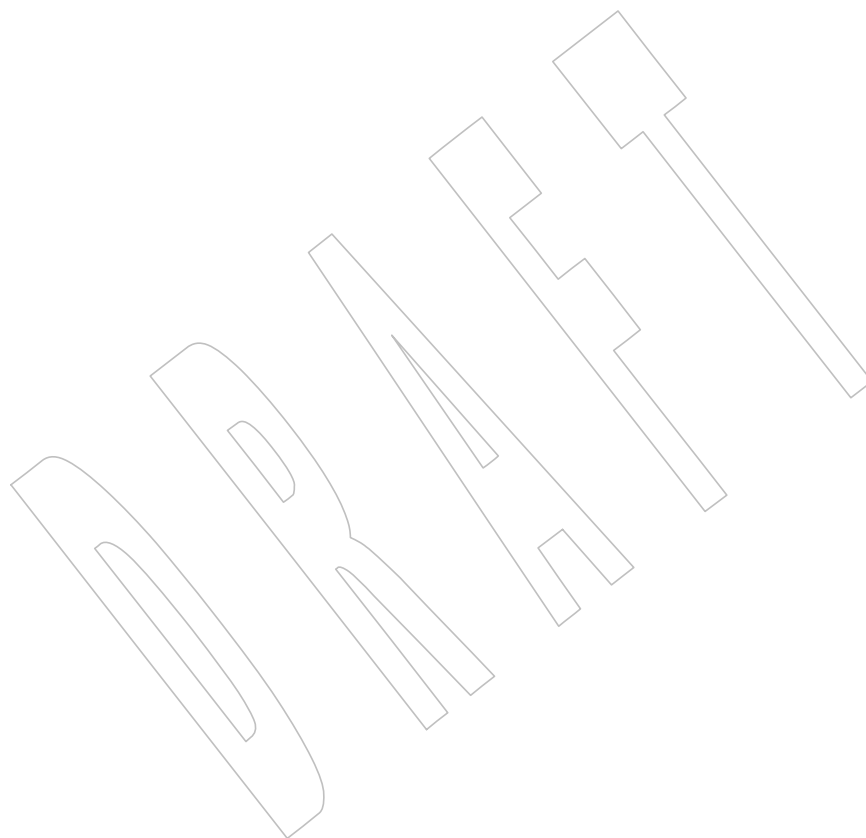


System Interfaces

18558

This chapter describes the functions, macros, and external variables to support applications portability at the C-language source level.

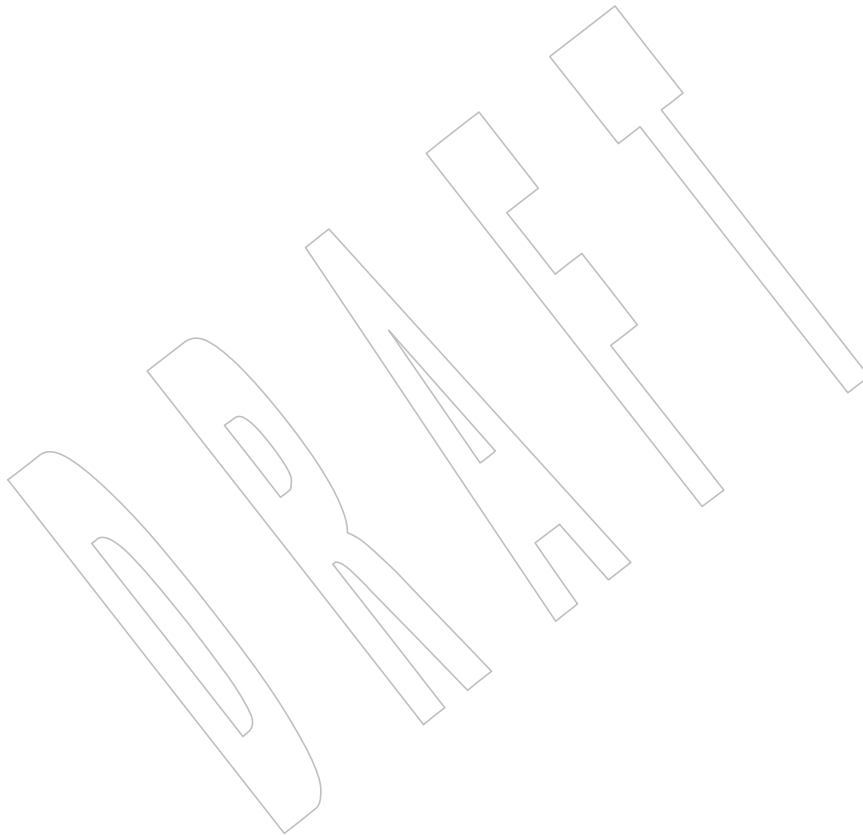
18559



18560 **NAME**
18561 FD_CLR — macros for synchronous I/O multiplexing

18562 **SYNOPSIS**
18563 #include <sys/select.h>
18564 void FD_CLR(int *fd*, fd_set **fdset*);
18565 int FD_ISSET(int *fd*, fd_set **fdset*);
18566 void FD_SET(int *fd*, fd_set **fdset*);
18567 void FD_ZERO(fd_set **fdset*);

18568 **DESCRIPTION**
18569 Refer to *pselect()*.



18570 **NAME**18571 `_Exit, _exit` — terminate a process18572 **SYNOPSIS**18573 `#include <stdlib.h>`18574 `void _Exit(int status);`18575 `#include <unistd.h>`18576 `void _exit(int status);`18577 **DESCRIPTION**18578 CX For `_Exit()`: The functionality described on this reference page is aligned with the ISO C
18579 standard. Any conflict between the requirements described here and the ISO C standard is
18580 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.18581 CX The value of *status* may be 0, `EXIT_SUCCESS`, `EXIT_FAILURE`, or any other value, though only
18582 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.18583 CX The `_Exit()` and `_exit()` functions shall be functionally equivalent.18584 CX The `_Exit()` and `_exit()` functions shall not call functions registered with `atexit()` nor any
18585 CX registered signal handlers. Open streams shall not be flushed. Whether open streams are
18586 closed (without flushing) is implementation-defined. Finally, the calling process is terminated
18587 with the consequences described below.18588 **Consequences of Process Termination**

18589 CX These functions shall terminate the calling process with the following consequences:

18590 **Note:** These consequences are all extensions to the ISO C standard and are not further CX shaded.
18591 However, functionality relating to the XSI option is shaded.18592 • All of the file descriptors, directory streams, conversion descriptors, and message catalog
18593 descriptors open in the calling process shall be closed.18594 XSI • If the parent process of the calling process is executing a `wait()`, `waitid()`, or `waitpid()`, and
18595 has neither set its `SA_NOCLDWAIT` flag nor set `SIGCHLD` to `SIG_IGN`, it shall be notified
18596 of termination of the calling process and the low-order eight bits (that is, bits 0377) of *status*
18597 shall be made available to it. If the parent is not waiting, the child's status shall be made
18598 available to it when the parent subsequently executes `wait()`, `waitid()`, or `waitpid()`.18599 The semantics of the `waitid()` function shall be equivalent to `wait()`.18600 XSI • If the parent process of the calling process is not executing a `wait()`, `waitid()`, or `waitpid()`,
18601 and has neither set its `SA_NOCLDWAIT` flag nor set `SIGCHLD` to `SIG_IGN`, the calling
18602 process shall be transformed into a *zombie process*. A *zombie process* is an inactive process
18603 and it shall be deleted at some later time when its parent process executes `wait()`, `waitid()`,
18604 or `waitpid()`.18605 XSI The semantics of the `waitid()` function shall be equivalent to `wait()`.18606 • Termination of a process does not directly terminate its children. The sending of a `SIGHUP`
18607 signal as described below indirectly terminates children in some circumstances.

18608 • Either:

18609 If the implementation supports the `SIGCHLD` signal, a `SIGCHLD` shall be sent to the
18610 parent process.

18611 Or:

_Exit()*System Interfaces*

- 18612 XSI If the parent process has set its SA_NOCLDWAIT flag, or set SIGCHLD to SIG_IGN, the
 18613 status shall be discarded, and the lifetime of the calling process shall end immediately. If
 18614 SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to
 18615 the parent process.
- The parent process ID of all of the existing child processes and zombie processes of the
 18616 calling process shall be set to the process ID of an implementation-defined system process.
 18617 That is, these processes shall be inherited by a special system process.
 18618
- 18619 XSI • Each attached shared-memory segment is detached and the value of *shm_nattch* (see
 18620 *shmget()*) in the data structure associated with its shared memory ID shall be decremented
 18621 by 1.
- 18622 XSI • For each semaphore for which the calling process has set a *semadj* value (see *semop()*), that
 18623 value shall be added to the *semval* of the specified semaphore.
- If the process is a controlling process, the SIGHUP signal shall be sent to each process in
 18624 the foreground process group of the controlling terminal belonging to the calling process.
 18625
 - If the process is a controlling process, the controlling terminal associated with the session
 18626 shall be disassociated from the session, allowing it to be acquired by a new controlling
 18627 process.
 18628
 - If the exit of the process causes a process group to become orphaned, and if any member of
 18629 the newly-orphaned process group is stopped, then a SIGHUP signal followed by a
 18630 SIGCONT signal shall be sent to each process in the newly-orphaned process group.
 18631
- All open named semaphores in the calling process shall be closed as if by appropriate calls
 18632 to *sem_close()*.
 18633
- 18634 ML • Any memory locks established by the process via calls to *mlockall()* or *mlock()* shall be
 18635 removed. If locked pages in the address space of the calling process are also mapped into
 18636 the address spaces of other processes and are locked by those processes, the locks
 18637 established by the other processes shall be unaffected by the call by this process to *_Exit()*
 18638 or *_exit()*.
- Memory mappings that were created in the process shall be unmapped before the process
 18639 is destroyed.
 18640
- 18641 TYM • Any blocks of typed memory that were mapped in the calling process shall be unmapped,
 18642 as if *munmap()* was implicitly called to unmap them.
- 18643 MSG • All open message queue descriptors in the calling process shall be closed as if by
 18644 appropriate calls to *mq_close()*.
- Any outstanding cancelable asynchronous I/O operations may be canceled. Those
 18645 asynchronous I/O operations that are not canceled shall complete as if the *_Exit()* or
 18646 *_exit()* operation had not yet occurred, but any associated signal notifications shall be
 18647 suppressed. The *_Exit()* or *_exit()* operation may block awaiting such I/O completion.
 18648 Whether any I/O is canceled, and which I/O may be canceled upon *_Exit()* or *_exit()*, is
 18649 implementation-defined.
 18650
- Threads terminated by a call to *_Exit()* or *_exit()* shall not invoke their cancellation
 18651 cleanup handlers or per-thread data destructors.
 18652
- 18653 OB TRC • If the calling process is a trace controller process, any trace streams that were created by
 18654 the calling process shall be shut down as described by the *posix_trace_shutdown()* function,
 18655 and mapping of trace event names to trace event type identifiers of any process built for
 18656 these trace streams may be deallocated.

18657 **RETURN VALUE**

18658 These functions do not return.

18659 **ERRORS**

18660 No errors are defined.

18661 **EXAMPLES**

18662 None.

18663 **APPLICATION USAGE**18664 Normally applications should use *exit()* rather than *_Exit()* or *_exit()*.18665 **RATIONALE**18666 **Process Termination**

18667 Early proposals drew a distinction between normal and abnormal process termination.
 18668 Abnormal termination was caused only by certain signals and resulted in implementation-
 18669 defined “actions”, as discussed below. Subsequent proposals distinguished three types of
 18670 termination: *normal termination* (as in the current specification), *simple abnormal termination*, and
 18671 *abnormal termination with actions*. Again the distinction between the two types of abnormal
 18672 termination was that they were caused by different signals and that implementation-defined
 18673 actions would result in the latter case. Given that these actions were completely implementation-
 18674 defined, the early proposals were only saying when the actions could occur and how their
 18675 occurrence could be detected, but not what they were. This was of little or no use to conforming
 18676 applications, and thus the distinction is not made in this volume of POSIX.1-200x.

18677 The implementation-defined actions usually include, in most historical implementations, the
 18678 creation of a file named **core** in the current working directory of the process. This file contains an
 18679 image of the memory of the process, together with descriptive information about the process,
 18680 perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

18681 There is a potential security problem in creating a **core** file if the process was set-user-ID and the
 18682 current user is not the owner of the program, if the process was set-group-ID and none of the
 18683 user’s groups match the group of the program, or if the user does not have permission to write
 18684 in the current directory. In this situation, an implementation either should not create a **core** file
 18685 or should make it unreadable by the user.

18686 Despite the silence of this volume of POSIX.1-200x on this feature, applications are advised not
 18687 to create files named **core** because of potential conflicts in many implementations. Some
 18688 implementations use a name other than **core** for the file; for example, by appending the process
 18689 ID to the filename.

18690 **Terminating a Process**

18691 It is important that the consequences of process termination as described occur regardless of
 18692 whether the process called *_exit()* (perhaps indirectly through *exit()*) or instead was terminated
 18693 due to a signal or for some other reason. Note that in the specific case of *exit()* this means that
 18694 the *status* argument to *exit()* is treated in the same way as the *status* argument to *_exit()*.

18695 A language other than C may have other termination primitives than the C-language *exit()*
 18696 function, and programs written in such a language should use its native termination primitives,
 18697 but those should have as part of their function the behavior of *_exit()* as described.
 18698 Implementations in languages other than C are outside the scope of this version of this volume
 18699 of POSIX.1-200x, however.

18700 As required by the ISO C standard, using **return** from *main()* has the same behavior (other than
 18701 with respect to language scope issues) as calling *exit()* with the returned value. Reaching the end
 18702 of the *main()* function has the same behavior as calling *exit(0)*.

18703 A value of zero (or `EXIT_SUCCESS`, which is required to be zero) for the argument *status*
18704 conventionally indicates successful termination. This corresponds to the specification for *exit()*
18705 in the ISO C standard. The convention is followed by utilities such as *make* and various shells,
18706 which interpret a zero status from a child process as success. For this reason, applications should
18707 not call *exit(0)* or *_exit(0)* when they terminate unsuccessfully; for example, in signal-catching
18708 functions.

18709 Historically, the implementation-defined process that inherits children whose parents have
18710 terminated without waiting on them is called *init* and has a process ID of 1.

18711 The sending of a `SIGHUP` to the foreground process group when a controlling process
18712 terminates corresponds to somewhat different historical implementations. In System V, the
18713 kernel sends a `SIGHUP` on termination of (essentially) a controlling process. In 4.2 BSD, the
18714 kernel does not send `SIGHUP` in a case like this, but the termination of a controlling process is
18715 usually noticed by a system daemon, which arranges to send a `SIGHUP` to the foreground
18716 process group with the *vhangup()* function. However, in 4.2 BSD, due to the behavior of the
18717 shells that support job control, the controlling process is usually a shell with no other processes
18718 in its process group. Thus, a change to make *_exit()* behave this way in such systems should not
18719 cause problems with existing applications.

18720 The termination of a process may cause a process group to become orphaned in either of two
18721 ways. The connection of a process group to its parent(s) outside of the group depends on both
18722 the parents and their children. Thus, a process group may be orphaned by the termination of the
18723 last connecting parent process outside of the group or by the termination of the last direct
18724 descendant of the parent process(es). In either case, if the termination of a process causes a
18725 process group to become orphaned, processes within the group are disconnected from their job
18726 control shell, which no longer has any information on the existence of the process group.
18727 Stopped processes within the group would languish forever. In order to avoid this problem,
18728 newly orphaned process groups that contain stopped processes are sent a `SIGHUP` signal and a
18729 `SIGCONT` signal to indicate that they have been disconnected from their session. The `SIGHUP`
18730 signal causes the process group members to terminate unless they are catching or ignoring
18731 `SIGHUP`. Under most circumstances, all of the members of the process group are stopped if any
18732 of them are stopped.

18733 The action of sending a `SIGHUP` and a `SIGCONT` signal to members of a newly orphaned
18734 process group is similar to the action of 4.2 BSD, which sends `SIGHUP` and `SIGCONT` to each
18735 stopped child of an exiting process. If such children exit in response to the `SIGHUP`, any
18736 additional descendants receive similar treatment at that time. In this volume of POSIX.1-200x,
18737 the signals are sent to the entire process group at the same time. Also, in this volume of
18738 POSIX.1-200x, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a
18739 process group that is not orphaned; therefore, the action taken at *_exit()* must consider processes
18740 other than child processes.

18741 It is possible for a process group to be orphaned by a call to *setpgid()* or *setsid()*, as well as by
18742 process termination. This volume of POSIX.1-200x does not require sending `SIGHUP` and
18743 `SIGCONT` in those cases, because, unlike process termination, those cases are not caused
18744 accidentally by applications that are unaware of job control. An implementation can choose to
18745 send `SIGHUP` and `SIGCONT` in those cases as an extension; such an extension must be
18746 documented as required in **<signal.h>**.

18747 The ISO/IEC 9899:1999 standard adds the *_Exit()* function that results in immediate program
18748 termination without triggering signals or *atexit()*-registered functions. In POSIX.1-200x, this is
18749 equivalent to the *_exit()* function.

18750
18751
18752
18753
18754
18755
18756
18757
18758
18759
18760
18761
18762
18763
18764
18765
18766
18767
18768
18769
18770
18771
18772
18773
18774
18775
18776
18777
18778
18779
18780

FUTURE DIRECTIONS

None.

SEE ALSO

atexit(), *exit()*, *mlock()*, *mlockall()*, *mq_close()*, *munmap()*, *posix_trace_create()*, *sem_close()*, *semop()*, *setpgid()*, *setsid()*, *shmget()*, *wait*, *waitid()*

XBD [<stdlib.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Interactions with the SA_NOCLDWAIT flag and SIGCHLD signal are further clarified.

The values of *status* from *exit()* are better described.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics for typed memory.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *_Exit()* function is included.
- The DESCRIPTION is updated.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

References to the *wait3()* function are removed.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/16 is applied, correcting grammar in the DESCRIPTION.

Issue 7

Austin Group Interpretation 1003.1-2001 #031 is applied, separating these functions from the *exit()* function.

Austin Group Interpretation 1003.1-2001 #085 is applied, clarifying the text regarding flushing of streams and closing of temporary files.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, and Semaphores options is moved to the Base.

18781 **NAME**

18782 _longjmp, _setjmp — non-local goto

18783 **SYNOPSIS**

```
18784 OB XSI #include <setjmp.h>
18785 void _longjmp(jmp_buf env, int val);
18786 int _setjmp(jmp_buf env);
```

18787 **DESCRIPTION**

18788 The *_longjmp()* and *_setjmp()* functions shall be equivalent to *longjmp()* and *setjmp()*,
 18789 respectively, with the additional restriction that *_longjmp()* and *_setjmp()* shall not manipulate
 18790 the signal mask.

18791 If *_longjmp()* is called even though *env* was never initialized by a call to *_setjmp()*, or when the
 18792 last such call was in a function that has since returned, the results are undefined.

18793 **RETURN VALUE**18794 Refer to *longjmp()* and *setjmp()*.18795 **ERRORS**

18796 No errors are defined.

18797 **EXAMPLES**

18798 None.

18799 **APPLICATION USAGE**

18800 If *_longjmp()* is executed and the environment in which *_setjmp()* was executed no longer exists,
 18801 errors can occur. The conditions under which the environment of the *_setjmp()* no longer exists
 18802 include exiting the function that contains the *_setjmp()* call, and exiting an inner block with
 18803 temporary storage. This condition might not be detectable, in which case the *_longjmp()* occurs
 18804 and, if the environment no longer exists, the contents of the temporary storage of an inner block
 18805 are unpredictable. This condition might also cause unexpected process termination. If the
 18806 function has returned, the results are undefined.

18807 Passing *longjmp()* a pointer to a buffer not created by *setjmp()*, passing *_longjmp()* a pointer to a
 18808 buffer not created by *_setjmp()*, passing *siglongjmp()* a pointer to a buffer not created by
 18809 *sigsetjmp()*, or passing any of these three functions a buffer that has been modified by the user
 18810 can cause all the problems listed above, and more.

18811 The *_longjmp()* and *_setjmp()* functions are included to support programs written to historical
 18812 system interfaces. New applications should use *siglongjmp()* and *sigsetjmp()* respectively.

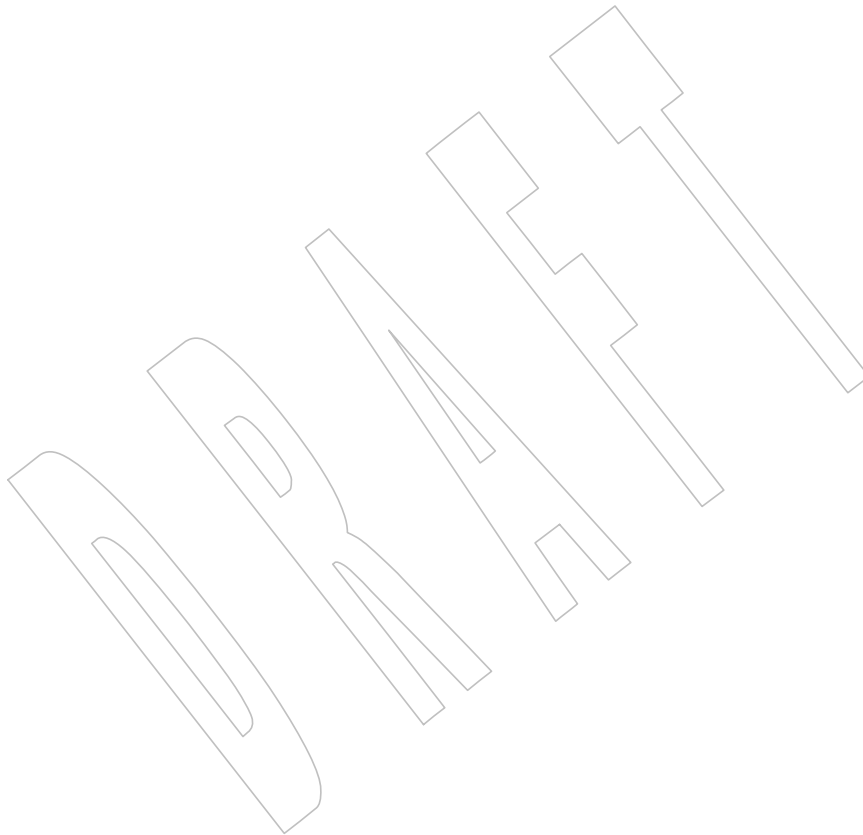
18813 **RATIONALE**

18814 None.

18815 **FUTURE DIRECTIONS**18816 The *_longjmp()* and *_setjmp()* functions may be removed in a future version.18817 **SEE ALSO**18818 *longjmp()*, *setjmp()*, *siglongjmp()*, *sigsetjmp()*

18819 XBD <setjmp.h>

18820	CHANGE HISTORY
18821	First released in Issue 4, Version 2.
18822	Issue 5
18823	Moved from X/OPEN UNIX extension to BASE.
18824	Issue 7
18825	The <code>_longjmp()</code> and <code>_setjmp()</code> functions are marked obsolescent.



18826 **NAME**
 18827 _toupper — transliterate uppercase characters to lowercase

18828 **SYNOPSIS**

18829 OB XSI `#include <ctype.h>`
 18830 `int _tolower(int c);`

18831 **DESCRIPTION**

18832 The *_tolower()* macro shall be equivalent to *tolower(c)* except that the application shall ensure
 18833 that the argument *c* is an uppercase letter.

18834 **RETURN VALUE**

18835 Upon successful completion, *_tolower()* shall return the lowercase letter corresponding to the
 18836 argument passed.

18837 **ERRORS**

18838 No errors are defined.

18839 **EXAMPLES**

18840 None.

18841 **APPLICATION USAGE**

18842 Applications should use the *tolower()* function instead of the obsolescent *_tolower()* function.

18843 **RATIONALE**

18844 None.

18845 **FUTURE DIRECTIONS**

18846 The *_tolower()* function may be removed in a future version.

18847 **SEE ALSO**

18848 *tolower()*, *isupper()*

18849 XBD [Chapter 7](#) (on page 121), [<ctype.h>](#)

18850 **CHANGE HISTORY**

18851 First released in Issue 1. Derived from Issue 1 of the SVID.

18852 **Issue 6**

18853 The normative text is updated to avoid use of the term “must” for application requirements.

18854 **Issue 7**

18855 The *_tolower()* function is marked obsolescent.

18856 **NAME**18857 `_toupper` — transliterate lowercase characters to uppercase18858 **SYNOPSIS**18859 OB XSI

```
#include <ctype.h>
18860 int _toupper(int c);
```

18861 **DESCRIPTION**18862 The `_toupper()` macro shall be equivalent to `toupper()` except that the application shall ensure
18863 that the argument `c` is a lowercase letter.18864 **RETURN VALUE**18865 Upon successful completion, `_toupper()` shall return the uppercase letter corresponding to the
18866 argument passed.18867 **ERRORS**

18868 No errors are defined.

18869 **EXAMPLES**

18870 None.

18871 **APPLICATION USAGE**18872 Applications should use the `toupper()` function instead of the obsolescent `_toupper()` function.18873 **RATIONALE**

18874 None.

18875 **FUTURE DIRECTIONS**18876 The `_toupper()` function may be removed in a future version.18877 **SEE ALSO**18878 [*islower\(\)*](#), [*toupper\(\)*](#)18879 XBD [Chapter 7](#) (on page 121), [<ctype.h>](#)18880 **CHANGE HISTORY**

18881 First released in Issue 1. Derived from Issue 1 of the SVID.

18882 **Issue 6**

18883 The normative text is updated to avoid use of the term “must” for application requirements.

18884 **Issue 7**18885 The `_toupper()` function is marked obsolescent.

18886 **NAME**
 18887 a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

18888 **SYNOPSIS**

```
18889 XSI #include <stdlib.h>
18890 long a64l(const char *s);
18891 char *l64a(long value);
```

18892 **DESCRIPTION**

18893 These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by
 18894 which 32-bit integers can be represented by up to six characters; each character represents a digit
 18895 in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall
 18896 be used for these operations.

18897 The characters used to represent digits are '.' (dot) for 0, '/' for 1, '0' through '9' for [2,11],
 18898 'A' through 'Z' for [12,37], and 'a' through 'z' for [38,63].

18899 The *a64l()* function shall take a pointer to a radix-64 representation, in which the first digit is the
 18900 least significant, and return the corresponding **long** value. If the string pointed to by *s* contains
 18901 more than six characters, *a64l()* shall use the first six. If the first six characters of the string
 18902 contain a null terminator, *a64l()* shall use only characters preceding the null terminator. The
 18903 *a64l()* function shall scan the character string from left to right with the least significant digit on
 18904 the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than
 18905 32 bits, the resulting value is sign-extended. The behavior of *a64l()* is unspecified if *s* is a null
 18906 pointer or the string pointed to by *s* was not generated by a previous call to *l64a()*.

18907 The *l64a()* function shall take a **long** argument and return a pointer to the corresponding
 18908 radix-64 representation. The behavior of *l64a()* is unspecified if *value* is negative.

18909 The value returned by *l64a()* may be a pointer into a static buffer. Subsequent calls to *l64a()* may
 18910 overwrite the buffer.

18911 The *l64a()* function need not be thread-safe. A function that is not required to be thread-safe is
 18912 not required to be reentrant.

18913 **RETURN VALUE**

18914 Upon successful completion, *a64l()* shall return the **long** value resulting from conversion of the
 18915 input string. If a string pointed to by *s* is an empty string, *a64l()* shall return 0L.

18916 The *l64a()* function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a()* shall
 18917 return a pointer to an empty string.

18918 **ERRORS**

18919 No errors are defined.

18920 **EXAMPLES**

18921 None.

18922 **APPLICATION USAGE**

18923 If the type **long** contains more than 32 bits, the result of *a64l(l64a(x))* is *x* in the low-order 32 bits.

18924 **RATIONALE**

18925 This is not the same encoding as used by either encoding variant of the *uuencode* utility.

18926 **FUTURE DIRECTIONS**

18927 None.

18928 **SEE ALSO**18929 *strtoul()*

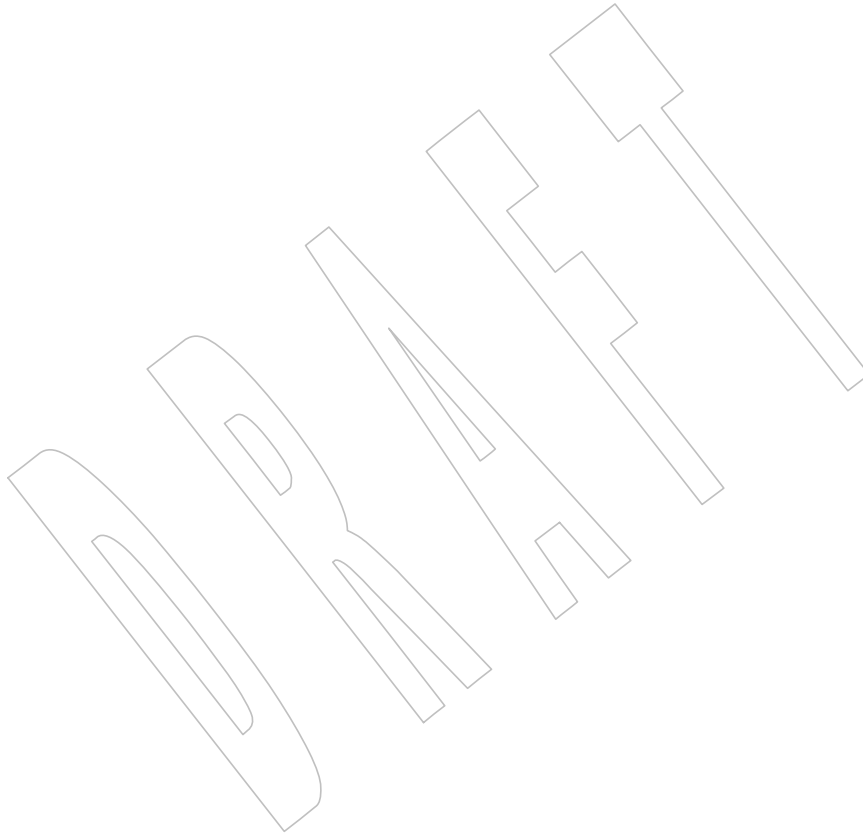
18930 XBD <stdlib.h>

18931 XCU *uuencode*18932 **CHANGE HISTORY**

18933 First released in Issue 4, Version 2.

18934 **Issue 5**

18935 Moved from X/OPEN UNIX extension to BASE.

18936 Normative text previously in the APPLICATION USAGE section is moved to the
18937 DESCRIPTION.18938 A note indicating that the *l64a()* function need not be reentrant is added to the DESCRIPTION.

18939 **NAME**

18940 abort — generate an abnormal process abort

18941 **SYNOPSIS**

18942 #include <stdlib.h>

18943 void abort(void);

18944 **DESCRIPTION**

18945 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 18946 conflict between the requirements described here and the ISO C standard is unintentional. This
 18947 volume of POSIX.1-200x defers to the ISO C standard.

18948 The *abort()* function shall cause abnormal process termination to occur, unless the signal
 18949 SIGABRT is being caught and the signal handler does not return.

18950 CX The abnormal termination processing shall include the default actions defined for SIGABRT and
 18951 may include an attempt to effect *fclose()* on all open streams.

18952 The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the
 18953 argument SIGABRT.

18954 CX The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort()* shall be that of a process
 18955 terminated by the SIGABRT signal. The *abort()* function shall override blocking or ignoring the
 18956 SIGABRT signal.

18957 **RETURN VALUE**18958 The *abort()* function shall not return.18959 **ERRORS**

18960 No errors are defined.

18961 **EXAMPLES**

18962 None.

18963 **APPLICATION USAGE**

18964 Catching the signal is intended to provide the application developer with a portable means to
 18965 abort processing, free from possible interference from any implementation-supplied functions.

18966 **RATIONALE**

18967 The ISO/IEC 9899:1999 standard requires the *abort()* function to be async-signal-safe. Since
 18968 POSIX.1-200x defers to the ISO C standard, this required a change to the DESCRIPTION from
 18969 “shall include the effect of *fclose()*” to “may include an attempt to effect *fclose()*.”

18970 The revised wording permits some backwards-compatibility and avoids a potential deadlock
 18971 situation.

18972 The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded
 18973 paragraph from the DESCRIPTION:

18974 “On XSI-conformant systems, in addition the abnormal termination processing shall include the
 18975 effect of *fclose()* on message catalog descriptors.”

18976 There were several reasons to remove this paragraph:

- 18977 • No special processing of open message catalogs needs to be performed prior to abnormal
 18978 process termination.
- 18979 • The main reason to specifically mention that *abort()* includes the effect of *fclose()* on open
 18980 streams is to flush output queued on the stream. Message catalogs in this context are read-
 18981 only and, therefore, do not need to be flushed.

18982
18983
18984
18985

18986
18987

18988
18989
18990

18991
18992

18993
18994
18995
18996
18997
18998
18999
19000
19001

- The effect of *fclose()* on a message catalog descriptor is unspecified. Message catalog descriptors are allowed, but not required to be implemented using a file descriptor, but there is no mention in POSIX.1-200x of a message catalog descriptor using a standard I/O stream FILE object as would be expected by *fclose()*.

FUTURE DIRECTIONS

None.

SEE ALSO

exit(), *kill*, *raise()*, *signal()*, *wait*, *waitid()*

XBD <stdlib.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

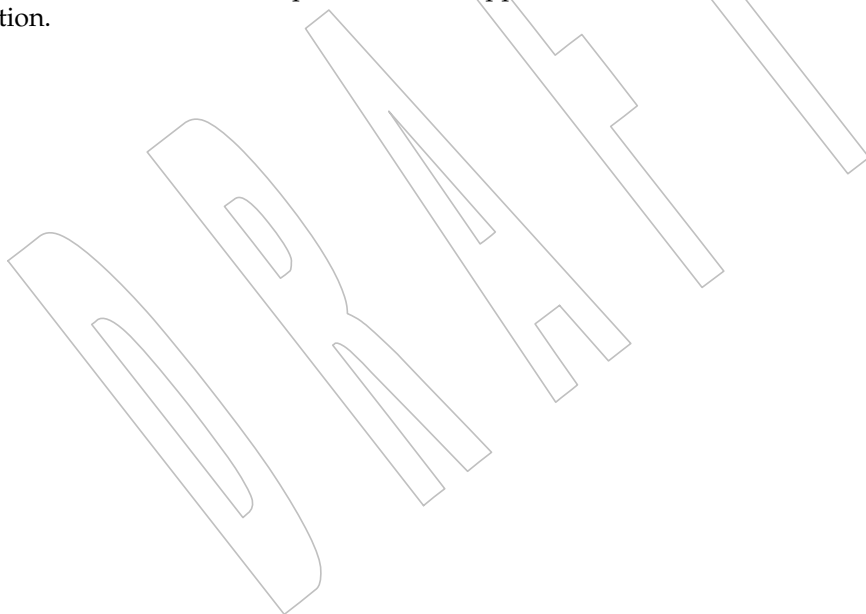
Extensions beyond the ISO C standard are marked.

Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

The Open Group Base Resolution bwg2002-003 is applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing “implementation-defined functions” to “implementation-supplied functions” in the APPLICATION USAGE section.



19002 **NAME**19003 `abs` — return an integer absolute value19004 **SYNOPSIS**19005 `#include <stdlib.h>`19006 `int abs(int i);`19007 **DESCRIPTION**

19008 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19009 conflict between the requirements described here and the ISO C standard is unintentional. This
 19010 volume of POSIX.1-200x defers to the ISO C standard.

19011 The `abs()` function shall compute the absolute value of its integer operand, *i*. If the result cannot
 19012 be represented, the behavior is undefined.

19013 **RETURN VALUE**19014 The `abs()` function shall return the absolute value of its integer operand.19015 **ERRORS**

19016 No errors are defined.

19017 **EXAMPLES**

19018 None.

19019 **APPLICATION USAGE**

19020 In two's-complement representation, the absolute value of the negative integer with largest
 19021 magnitude {INT_MIN} might not be representable.

19022 **RATIONALE**

19023 None.

19024 **FUTURE DIRECTIONS**

19025 None.

19026 **SEE ALSO**19027 *[fabs\(\)](#), [labs\(\)](#)*19028 XBD [<stdlib.h>](#)19029 **CHANGE HISTORY**

19030 First released in Issue 1. Derived from Issue 1 of the SVID.

19031 **Issue 6**

19032 Extensions beyond the ISO C standard are marked.

19033 **NAME**
 19034 accept — accept a new connection on a socket

19035 **SYNOPSIS**
 19036 #include <sys/socket.h>
 19037 int accept(int *socket*, struct sockaddr *restrict *address*,
 19038 socklen_t *restrict *address_len*);

19039 **DESCRIPTION**
 19040 The *accept()* function shall extract the first connection on the queue of pending connections,
 19041 create a new socket with the same socket type protocol and address family as the specified
 19042 socket, and allocate a new file descriptor for that socket.

19043 The *accept()* function takes the following arguments:

19044 *socket* Specifies a socket that was created with *socket()*, has been bound to an address
 19045 with *bind()*, and has issued a successful call to *listen()*.

19046 *address* Either a null pointer, or a pointer to a **sockaddr** structure where the address of
 19047 the connecting socket shall be returned.

19048 *address_len* Points to a **socklen_t** structure which on input specifies the length of the
 19049 supplied **sockaddr** structure, and on output specifies the length of the stored
 19050 address.

19051 If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored
 19052 in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in
 19053 the object pointed to by *address_len*.

19054 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 19055 the stored address shall be truncated.

19056 If the protocol permits connections by unbound clients, and the peer is not bound, then the
 19057 value stored in the object pointed to by *address* is unspecified.

19058 If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file
 19059 descriptor for the socket, *accept()* shall block until a connection is present. If the *listen()* queue is
 19060 empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket,
 19061 *accept()* shall fail and set *errno* to [EAGAIN] or [EWOULDBLOCK].

19062 The accepted socket cannot itself accept more connections. The original socket remains open and
 19063 can accept more connections.

19064 **RETURN VALUE**
 19065 Upon successful completion, *accept()* shall return the non-negative file descriptor of the accepted
 19066 socket. Otherwise, -1 shall be returned and *errno* set to indicate the error.

19067 **ERRORS**
 19068 The *accept()* function shall fail if:
 19069 [EAGAIN] or [EWOULDBLOCK]
 19070 O_NONBLOCK is set for the socket file descriptor and no connections are
 19071 present to be accepted.
 19072 [EBADF] The *socket* argument is not a valid file descriptor.
 19073 [ECONNABORTED]
 19074 A connection has been aborted.

accept()

19075	[EINTR]	The <i>accept()</i> function was interrupted by a signal that was caught before a valid connection arrived.
19076		
19077	[EINVAL]	The <i>socket</i> is not accepting connections.
19078	[EMFILE]	All file descriptors available to the process are currently open.
19079	[ENFILE]	The maximum number of file descriptors in the system are already open.
19080	[ENOBUFS]	No buffer space is available.
19081	[ENOMEM]	There was insufficient memory available to complete the operation.
19082	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
19083	[EOPNOTSUPP]	The socket type of the specified socket does not support accepting connections.
19084		
19085		The <i>accept()</i> function may fail if:
19086	OB XSR [EPROTO]	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
19087		

EXAMPLES

None.

APPLICATION USAGE

When a connection is available, *select()* indicates that the file descriptor for the socket is ready for reading.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*bind()*, *connect()*, *listen()*, *socket()*

XBD <sys/socket.h>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *accept()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS] and [ENOMEM] errors to become “shall fail” errors.

Functionality relating to XSI STREAMS is marked obsolescent.

19109 **NAME**

19110 access, faccessat — determine accessibility of a file relative to directory file descriptor

19111 **SYNOPSIS**

19112 #include <unistd.h>

19113 int access(const char *path, int amode);

19114 int faccessat(int fd, const char *path, int amode, int flag);

19115 **DESCRIPTION**19116 The *access()* function shall check the file named by the pathname pointed to by the *path*
19117 argument for accessibility according to the bit pattern contained in *amode*, using the real user ID
19118 in place of the effective user ID and the real group ID in place of the effective group ID.19119 The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked
19120 (R_OK, W_OK, X_OK) or the existence test (F_OK).19121 If any access permissions are checked, each shall be checked individually, as described in XBD |
19122 Section 4.4 (on page 96), except that where that description refers to execute permission for a |
19123 process with appropriate privileges, an implementation may indicate success for X_OK even if |
19124 execute permission is not granted to any user.19125 The *faccessat()* function shall be equivalent to the *access()* function, except in the case where *path*
19126 specifies a relative path. In this case the file whose accessibility is to be determined shall be
19127 located relative to the directory associated with the file descriptor *fd* instead of the current
19128 working directory. It is unspecified whether directory searches are permitted based on whether
19129 the file was opened with search permission or on the current permissions of the directory
19130 underlying the file descriptor.19131 If *faccessat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
19132 directory is used and the behavior shall be identical to a call to *access()*.19133 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
19134 in <fcntl.h>:19135 AT_EACCESS The checks for accessibility are performed using the effective user and group
19136 IDs instead of the real user and group ID as required in a call to *access()*.19137 **RETURN VALUE**19138 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
19139 return -1 and set *errno* to indicate the error.19140 **ERRORS**

19141 These functions shall fail if:

19142 [EACCES] Permission bits of the file mode do not permit the requested access, or search
19143 permission is denied on a component of the path prefix.19144 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
19145 argument.

19146 [ENAMETOOLONG]

19147 The length of the *path* argument exceeds {PATH_MAX} or a pathname
19148 component is longer than {NAME_MAX}.19149 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

19150 [ENOTDIR] A component of the path prefix is not a directory.

access()

- 19151 [EROFS] Write access is requested for a file on a read-only file system.
- 19152 The *faccessat()* function shall fail if:
- 19153 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
19154 neither AT_FDCWD nor a valid file descriptor.
- 19155 These functions may fail if:
- 19156 [EINVAL] The value of the *amode* argument is invalid.
- 19157 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
19158 resolution of the *path* argument.
- 19159 [ENAMETOOLONG]
19160 As a result of encountering a symbolic link in resolution of the *path* argument,
19161 the length of the substituted pathname string exceeded {PATH_MAX}.
- 19162 [ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being
19163 executed.
- 19164 The *faccessat()* function may fail if:
- 19165 [EINVAL] The value of the *flag* argument is not valid.
- 19166 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
19167 file descriptor associated with a directory.

EXAMPLES**Testing for the Existence of a File**

The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
19171 #include <unistd.h>
19172 ...
19173 int result;
19174 const char *filename = "/tmp/myfile";
19175 result = access (filename, F_OK);
```

APPLICATION USAGE

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

The use of the AT_EACCESS value for *flag* enables functionality not available in *access()*.

RATIONALE

In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()* function because:

- 19183 1. Historical implementations of *access()* do not test file access correctly when the process'
19184 real user ID is superuser. In particular, they always return zero when testing execute
19185 permissions without regard to whether the file is executable.
- 19186 2. The superuser has complete access to all files on a system. As a consequence, programs
19187 started by the superuser and switched to the effective user ID with lesser privileges
19188 cannot use *access()* to test their file access permissions.

19189 However, the historical model of *eaccess()* does not resolve problem (1), so this volume of
19190 POSIX.1-200x now allows *access()* to behave in the desired way because several implementations
19191 have corrected the problem. It was also argued that problem (2) is more easily solved by using
19192 *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather
19193 than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in

19194 this volume of POSIX.1-200x.

19195 The sentence concerning appropriate privileges and execute permission bits reflects the two
19196 possibilities implemented by historical implementations when checking superuser access for
19197 X_OK.

19198 New implementations are discouraged from returning X_OK unless at least one execution
19199 permission bit is set.

19200 The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in
19201 directories other than the current working directory without exposure to race conditions. Any
19202 part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified
19203 behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it
19204 can be guaranteed that the file tested for accessibility is located relative to the desired directory.

19205 FUTURE DIRECTIONS

19206 None.

19207 SEE ALSO

19208 *chmod*, *fstatat()*

19209 XBD Section 4.4 (on page 96), *<fcntl.h>*, *<unistd.h>*

19210 CHANGE HISTORY

19211 First released in Issue 1. Derived from Issue 1 of the SVID.

19212 Issue 6

19213 The following new requirements on POSIX implementations derive from alignment with the
19214 Single UNIX Specification:

- 19215 • The [ELOOP] mandatory error condition is added.
- 19216 • A second [ENAMETOOLONG] is added as an optional error condition.
- 19217 • The [ETXTBSY] optional error condition is added.

19218 The following changes were made to align with the IEEE P1003.1a draft standard:

- 19219 • The [ELOOP] optional error condition is added.

19220 Issue 7

19221 Austin Group Interpretation 1003.1-2001 #046 is applied.

19222 The *faccessat()* function is added from The Open Group Technical Standard, 2006, Extended API
19223 Set Part 2.

19224 **NAME**

19225 acos, acosf, acosl — arc cosine functions

19226 **SYNOPSIS**

```
19227     #include <math.h>
19228
19228     double acos(double x);
19229     float  acosf(float x);
19230     long double acosl(long double x);
```

19231 **DESCRIPTION**

19232 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19233 conflict between the requirements described here and the ISO C standard is unintentional. This
 19234 volume of POSIX.1-200x defers to the ISO C standard.

19235 These functions shall compute the principal value of the arc cosine of their argument x . The
 19236 value of x should be in the range $[-1,1]$.

19237 An application wishing to check for error situations should set *errno* to zero and call
 19238 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 19239 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 19240 zero, an error has occurred.

19241 **RETURN VALUE**

19242 Upon successful completion, these functions shall return the arc cosine of x , in the range $[0,\pi]$
 19243 radians.

19244 MX For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 19245 supported), or an implementation-defined value shall be returned.

19246 MX If x is NaN, a NaN shall be returned.

19247 If x is $+1$, $+0$ shall be returned.

19248 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 19249 defined value shall be returned.

19250 **ERRORS**

19251 These functions shall fail if:

19252 MX Domain Error The x argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.

19253 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 19254 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 19255 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 19256 shall be raised.

19257 **EXAMPLES**

19258 None.

19259 **APPLICATION USAGE**

19260 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 19261 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

19262 **RATIONALE**

19263 None.

19264 **FUTURE DIRECTIONS**

19265 None.

19266 **SEE ALSO**19267 *cos()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

19268 XBD Section 4.19 (on page 104), <math.h>

19269 **CHANGE HISTORY**

19270 First released in Issue 1. Derived from Issue 1 of the SVID.

19271 **Issue 5**19272 The DESCRIPTION is updated to indicate how an application should check for an error. This
19273 text was previously published in the APPLICATION USAGE section.19274 **Issue 6**19275 The *acosf()* and *acosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.19276 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
19277 revised to align with the ISO/IEC 9899:1999 standard.19278 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
19279 marked.

DRAFT

19280 **NAME**
 19281 `acosh`, `acoshf`, `acoshl` — inverse hyperbolic cosine functions

19282 **SYNOPSIS**
 19283 `#include <math.h>`
 19284 `double acosh(double x);`
 19285 `float acoshf(float x);`
 19286 `long double acoshl(long double x);`

19287 **DESCRIPTION**
 19288 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 19289 conflict between the requirements described here and the ISO C standard is unintentional. This
 19290 volume of POSIX.1-200x defers to the ISO C standard.

19291 These functions shall compute the inverse hyperbolic cosine of their argument x .
 19292 An application wishing to check for error situations should set `errno` to zero and call
 19293 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 19294 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 19295 zero, an error has occurred.

19296 **RETURN VALUE**
 19297 Upon successful completion, these functions shall return the inverse hyperbolic cosine of their
 19298 argument.

19299 MX For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an
 19300 implementation-defined value shall be returned.

19301 MX If x is NaN, a NaN shall be returned.
 19302 If x is $+1$, $+0$ shall be returned.
 19303 If x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.
 19304 If x is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 19305 defined value shall be returned.

19306 **ERRORS**
 19307 These functions shall fail if:
 19308 MX Domain Error The x argument is finite and less than $+1.0$, or is $-\text{Inf}$.
 19309 If the integer expression (`math_errhandling` & `MATH_ERRNO`) is non-zero,
 19310 then `errno` shall be set to `[EDOM]`. If the integer expression (`math_errhandling`
 19311 & `MATH_ERREXCEPT`) is non-zero, then the invalid floating-point exception
 19312 shall be raised.

19313 **EXAMPLES**
 19314 None.

19315 **APPLICATION USAGE**
 19316 On error, the expressions (`math_errhandling` & `MATH_ERRNO`) and (`math_errhandling` &
 19317 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

19318 **RATIONALE**
 19319 None.

19320
19321
19322
19323
19324
19325
19326
19327
19328
19329
19330
19331
19332
19333
19334
19335
19336

FUTURE DIRECTIONS

None.

SEE ALSO

cosh(), *feclearexcept()*, *fetestexcept()*

XBD Section 4.19 (on page 104), <math.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *acosh()* function is no longer marked as an extension.

The *acoshf()* and *acoshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

acosl()

19337

19338

NAME

19339

acosl — arc cosine functions

19340

SYNOPSIS

19341

#include <math.h>

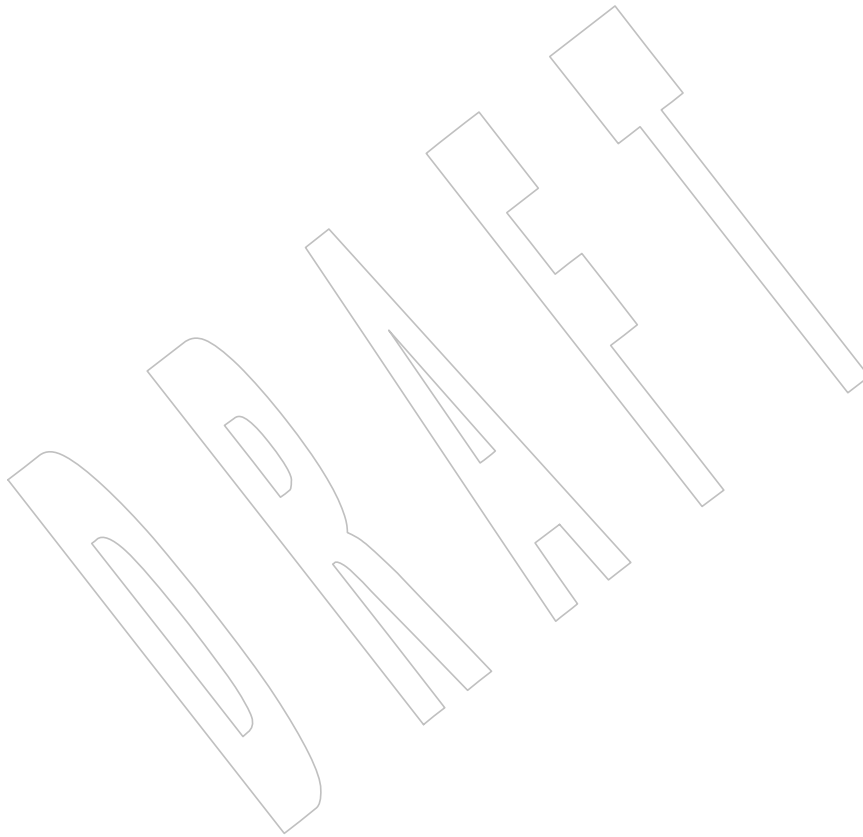
19342

long double acosl(long double x);

19343

DESCRIPTION

19344

Refer to *acos()*.

19345 **NAME**

19346 aio_cancel — cancel an asynchronous I/O request

19347 **SYNOPSIS**

19348 #include <aio.h>

19349 int aio_cancel(int *fildev*, struct aiocb **aiocbp*);19350 **DESCRIPTION**

19351 The *aio_cancel()* function shall attempt to cancel one or more asynchronous I/O requests
 19352 currently outstanding against file descriptor *fildev*. The *aiocbp* argument points to the
 19353 asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then
 19354 all outstanding cancelable asynchronous I/O requests against *fildev* shall be canceled.

19355 Normal asynchronous notification shall occur for asynchronous I/O operations that are
 19356 successfully canceled. If there are requests that cannot be canceled, then the normal
 19357 asynchronous completion process shall take place for those requests when they are completed.

19358 For requested operations that are successfully canceled, the associated error status shall be set to
 19359 [ECANCELED] and the return status shall be -1. For requested operations that are not
 19360 successfully canceled, the *aiocbp* shall not be modified by *aio_cancel()*.

19361 If *aiocbp* is not NULL, then if *fildev* does not have the same value as the file descriptor with which
 19362 the asynchronous operation was initiated, unspecified results occur.

19363 Which operations are cancelable is implementation-defined.

19364 **RETURN VALUE**

19365 The *aio_cancel()* function shall return the value AIO_CANCELED if the requested operation(s)
 19366 were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the
 19367 requested operation(s) cannot be canceled because it is in progress. In this case, the state of the
 19368 other operations, if any, referenced in the call to *aio_cancel()* is not indicated by the return value
 19369 of *aio_cancel()*. The application may determine the state of affairs for these operations by using
 19370 *aio_error()*. The value AIO_ALLDONE is returned if all of the operations have already
 19371 completed. Otherwise, the function shall return -1 and set *errno* to indicate the error.

19372 **ERRORS**

19373 The *aio_cancel()* function shall fail if:

19374 [EBADF] The *fildev* argument is not a valid file descriptor.

19375 **EXAMPLES**

19376 None.

19377 **APPLICATION USAGE**

19378 None.

19379 **RATIONALE**

19380 None.

19381 **FUTURE DIRECTIONS**

19382 None.

19383 **SEE ALSO**

19384 [aio_read\(\)](#), [aio_write\(\)](#)

19385 XBD [<aio.h>](#)

19386

CHANGE HISTORY

19387

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19388

Issue 6

19389

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

19390

19391

The APPLICATION USAGE section is added.

19392

19393

19394

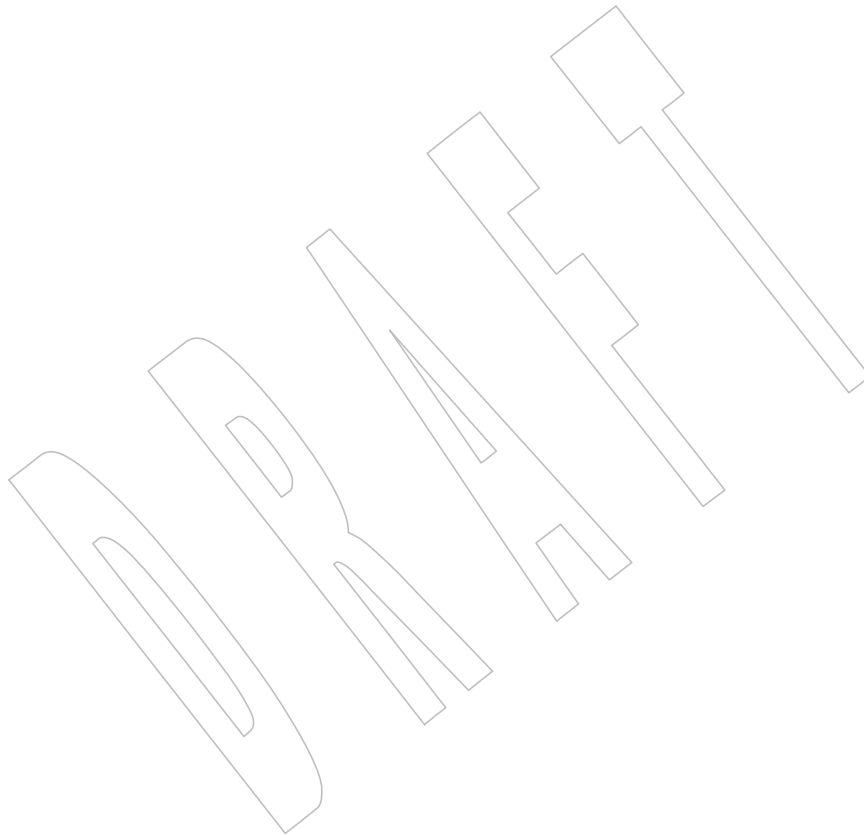
IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/10 is applied, removing the words “to the calling process” in the RETURN VALUE section. The term was unnecessary and precluded threads.

19395

Issue 7

19396

The *aio_cancel()* function is moved from the Asynchronous Input and Output option to the Base.



19397 **NAME**

19398 aio_error — retrieve errors status for an asynchronous I/O operation

19399 **SYNOPSIS**

19400 #include <aio.h>

19401 int aio_error(const struct aiocb *aiocbp);

19402 **DESCRIPTION**

19403 The *aio_error()* function shall return the error status associated with the **aiocb** structure
 19404 referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the
 19405 SIO *errno* value that would be set by the corresponding *read()*, *write()*, *fdatasync()*, or *fsync()*
 19406 operation. If the operation has not yet completed, then the error status shall be equal to
 19407 [EINPROGRESS].

19408 If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been
 19409 scheduled, the results are undefined.

19410 **RETURN VALUE**

19411 If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the
 19412 asynchronous operation has completed unsuccessfully, then the error status, as described for
 19413 SIO *read()*, *write()*, *fdatasync()*, and *fsync()*, shall be returned. If the asynchronous I/O operation has
 19414 not yet completed, then [EINPROGRESS] shall be returned.

19415 If the *aio_error()* function fails, it shall return -1 and set *errno* to indicate the error. +

19416 **ERRORS**19417 The *aio_error()* function may fail if:

19418 [EINVAL] The *aiocbp* argument does not refer to an asynchronous operation whose
 19419 return status has not yet been retrieved.

19420 **EXAMPLES**

19421 None.

19422 **APPLICATION USAGE**

19423 None.

19424 **RATIONALE**

19425 None.

19426 **FUTURE DIRECTIONS**

19427 None.

19428 **SEE ALSO**

19429 *aio_cancel()*, *aio_fsync()*, *aio_read()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*,
 19430 *lseek()*, *read*

19431 XBD <aio.h> +

19432 **CHANGE HISTORY**

19433 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19434 **Issue 6**

19435 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19436 implementation does not support the Asynchronous Input and Output option.

19437 The APPLICATION USAGE section is added.

19438

Issue 7

19439

Austin Group Interpretation 1003.1-2001 #045 is applied.

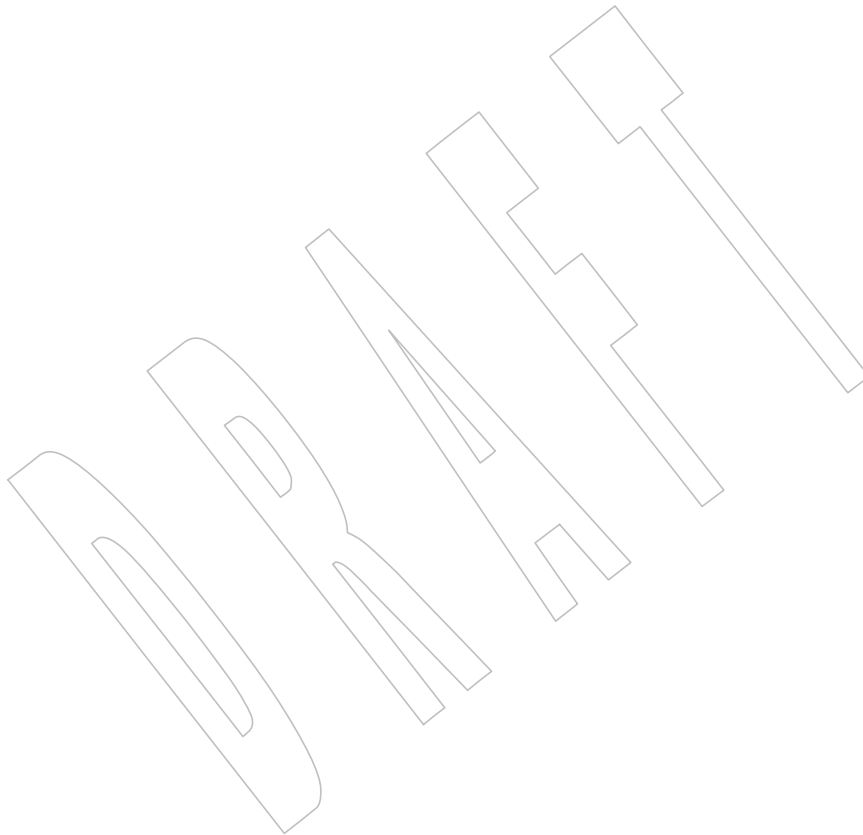
19440

SD5-XSH-ERN-148 is applied.

19441

The *aio_error()* function is moved from the Asynchronous Input and Output option to the Base.

+



19442 **NAME**

19443 aio_fsync — asynchronous file synchronization

19444 **SYNOPSIS**

19445 #include <aio.h>

19446 int aio_fsync(int op, struct aiocb *aiocbp);

19447 **DESCRIPTION**

19448 The *aio_fsync()* function shall asynchronously force all I/O operations associated with the file
 19449 indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp*
 19450 argument and queued at the time of the call to *aio_fsync()* to the synchronized I/O completion
 19451 state. The function call shall return when the synchronization request has been initiated or
 19452 queued to the file or device (even when the data cannot be synchronized immediately).

19453 If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to
 19454 *fdatasync()*; that is, as defined for synchronized I/O data integrity completion. If *op* is O_SYNC,
 19455 all currently queued I/O operations shall be completed as if by a call to *fsync()*; that is, as
 19456 defined for synchronized I/O file integrity completion. If the *aio_fsync()* function fails, or if the
 19457 operation queued by *aio_fsync()* fails, then, as for *fsync()* and *fdatasync()*, outstanding I/O
 19458 operations are not guaranteed to have been completed.

19459 If *aio_fsync()* succeeds, then it is only the I/O that was queued at the time of the call to
 19460 *aio_fsync()* that is guaranteed to be forced to the relevant completion state. The completion of
 19461 subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized
 19462 fashion.

19463 The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used
 19464 as an argument to *aio_error()* and *aio_return()* in order to determine the error status and return
 19465 status, respectively, of the asynchronous operation while it is proceeding. When the request is
 19466 queued, the error status for the operation is [EINPROGRESS]. When all data has been
 19467 successfully transferred, the error status shall be reset to reflect the success or failure of the
 19468 operation. If the operation does not complete successfully, the error status for the operation shall
 19469 be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification
 19470 to occur as specified in Section 2.4.1 (on page 463) when all operations have achieved synchronized
 19471 I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the
 19472 control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O
 19473 completion, then the behavior is undefined.

19474 If the *aio_fsync()* function fails or *aiocbp* indicates an error condition, data is not guaranteed to
 19475 have been successfully transferred.

19476 **RETURN VALUE**

19477 The *aio_fsync()* function shall return the value 0 if the I/O operation is successfully queued;
 19478 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

19479 **ERRORS**

19480 The *aio_fsync()* function shall fail if:

- | | | |
|-------|----------|--|
| 19481 | [EAGAIN] | The requested asynchronous operation was not queued due to temporary resource limitations. |
| 19482 | | |
| 19483 | [EBADF] | The <i>aio_fildes</i> member of the aiocb structure referenced by the <i>aiocbp</i> argument is not a valid file descriptor open for writing. |
| 19484 | | |
| 19485 | [EINVAL] | This implementation does not support synchronized I/O for this file. |

aio_fsync()

19486 [EINVAL] A value of *op* other than O_DSYNC or O_SYNC was specified.

19487 In the event that any of the queued I/O operations fail, *aio_fsync()* shall return the error
 19488 condition defined for *read()* and *write()*. The error is returned in the error status for the
 19489 asynchronous *fsync()* operation, which can be retrieved using *aio_error()*.

EXAMPLES

19490 None.
 19491

APPLICATION USAGE

19492 None.
 19493

RATIONALE

19494 None.
 19495

FUTURE DIRECTIONS

19496 None.
 19497

SEE ALSO

19498 *fcntl()*, *fdatasync()*, *fsync()*, *open()*, *read*, *write*
 19499

19500 XBD [<aio.h>](#)

CHANGE HISTORY

19501 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
 19502

Issue 6

19503 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19504 implementation does not support the Asynchronous Input and Output option.
 19505

19506 The APPLICATION USAGE section is added.

19507 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/11 is applied, removing the words “to the
 19508 calling process” in the RETURN VALUE section. The term was unnecessary and precluded
 19509 threads.

Issue 7

19510 The *aio_fsync()* function is moved from the Asynchronous Input and Output option to the Base.
 19511

19512 **NAME**

19513 aio_read — asynchronous read from a file

19514 **SYNOPSIS**

19515 #include <aio.h>

19516 int aio_read(struct aiocb *aiocbp);

19517 **DESCRIPTION**

19518 The *aio_read()* function shall read *aiocbp->aio_nbytes* from the file associated with
 19519 *aiocbp->aio_fildes* into the buffer pointed to by *aiocbp->aio_buf*. The function call shall return
 19520 when the read request has been initiated or queued to the file or device (even when the data
 19521 cannot be delivered immediately).

19522 **PIO** If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
 19523 at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution
 19524 Scheduling is not supported, then the base scheduling priority is that of the calling process;
 19525 **PIO TPS** otherwise, the base scheduling priority is that of the calling thread.

19526 The *aiocbp* value may be used as an argument to *aio_error()* and *aio_return()* in order to
 19527 determine the error status and return status, respectively, of the asynchronous operation while it
 19528 is proceeding. If an error condition is encountered during queuing, the function call shall return
 19529 without having initiated or queued the request. The requested operation takes place at the
 19530 absolute position in the file as given by *aio_offset*, as if *lseek()* were called immediately prior to
 19531 the operation with an *offset* equal to *aio_offset* and a *whence* equal to *SEEK_SET*. After a
 19532 successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file
 19533 is unspecified.

19534 The *aio_sigevent* member specifies the notification which occurs when the request is completed.

19535 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_read()*.

19536 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
 19537 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
 19538 completion, then the behavior is undefined.

19539 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

19540 **SIO** If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
 19541 function shall be according to the definitions of synchronized I/O data integrity completion and
 19542 synchronized I/O file integrity completion.

19543 For any system action that changes the process memory space while an asynchronous I/O is
 19544 outstanding to the address range being changed, the result of that action is undefined.

19545 For regular files, no data transfer shall occur past the offset maximum established in the open
 19546 file description associated with *aiocbp->aio_fildes*.

19547 **RETURN VALUE**

19548 The *aio_read()* function shall return the value zero if the I/O operation is successfully queued;
 19549 otherwise, the function shall return the value -1 and set *errno* to indicate the error.

19550 **ERRORS**

19551 The *aio_read()* function shall fail if:

19552 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
 19553 resource limitations.

19554 Each of the following conditions may be detected synchronously at the time of the call to
 19555 *aio_read()*, or asynchronously. If any of the conditions below are detected synchronously, the

aio_read()

19556 *aio_read()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 19557 conditions below are detected asynchronously, the return status of the asynchronous operation
 19558 is set to `-1`, and the error status of the asynchronous operation is set to the corresponding value.

19559 [EBADF] The *aiocbp*→*aio_fildes* argument is not a valid file descriptor open for reading.

19560 [EINVAL] The file offset value implied by *aiocbp*→*aio_offset* would be invalid,
 19561 PIO *aiocbp*→*aio_reqprio* is not a valid value, or *aiocbp*→*aio_nbytes* is an invalid
 19562 value.

19563 In the case that the *aio_read()* successfully queues the I/O operation but the operation is
 19564 subsequently canceled or encounters an error, the return status of the asynchronous operation is
 19565 one of the values normally returned by the *read()* function call. In addition, the error status of
 19566 the asynchronous operation is set to one of the error statuses normally set by the *read()* function
 19567 call, or one of the following values:

19568 [EBADF] The *aiocbp*→*aio_fildes* argument is not a valid file descriptor open for reading.

19569 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 19570 *aio_cancel()* request.

19571 [EINVAL] The file offset value implied by *aiocbp*→*aio_offset* would be invalid.

19572 The following condition may be detected synchronously or asynchronously:

19573 [EOVERFLOW] The file is a regular file, *aiocbp*→*aio_nbytes* is greater than 0, and the starting
 19574 offset in *aiocbp*→*aio_offset* is before the end-of-file and is at or beyond the
 19575 offset maximum in the open file description associated with *aiocbp*→*aio_fildes*.

EXAMPLES

19576 None.
 19577

APPLICATION USAGE

19578 None.
 19579

RATIONALE

19580 None.
 19581

FUTURE DIRECTIONS

19582 None.
 19583

SEE ALSO

19584 *aio_cancel()*, *aio_error()*, *lio_listio()*, *aio_return()*, *aio_write()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
 19585 *read*
 19586

19587 XBD <[aio.h](#)>

CHANGE HISTORY

19588 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
 19589

19590 Large File Summit extensions are added.

Issue 6

19591 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19592 implementation does not support the Asynchronous Input and Output option.
 19593

19594 The APPLICATION USAGE section is added.

19595

19596

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

19597

19598

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.

19599

19600

- In the ERRORS section, the [Eoverflow] condition is added. This change is to support large files.

19601

19602

19603

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/12 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words “to the calling process” in the RETURN VALUE section.

19604

19605

19606

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/13 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aio_cbp*→*aio_reqprio* is only required if the Prioritized Input and Output option is supported.

19607

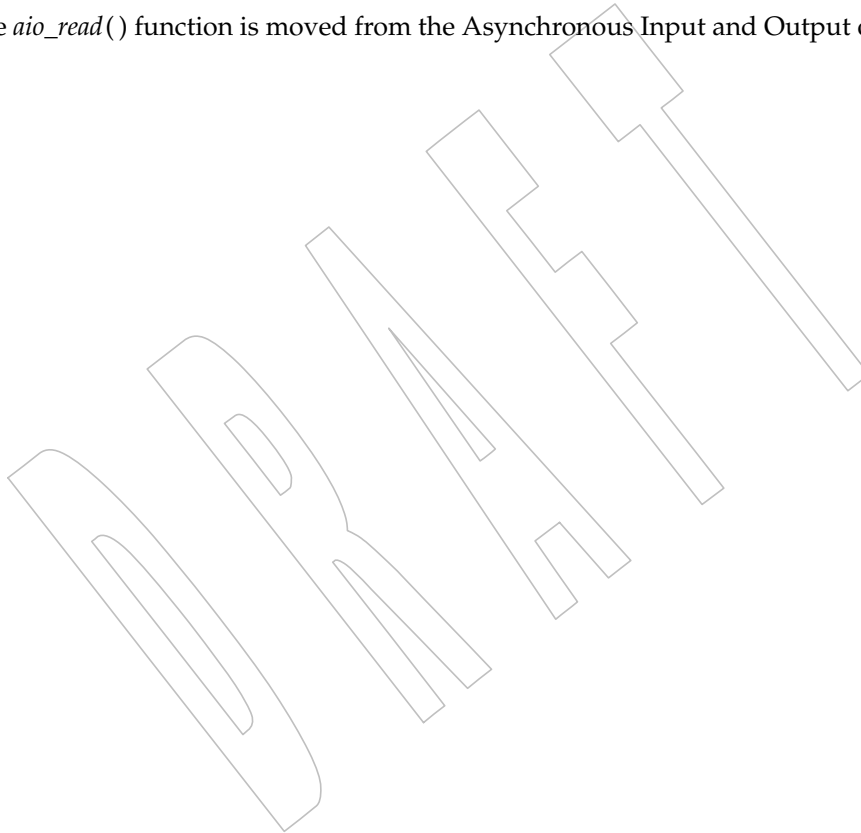
Issue 7

19608

Austin Group Interpretation 1003.1-2001 #082 is applied.

19609

The *aio_read()* function is moved from the Asynchronous Input and Output option to the Base.



19610 **NAME**19611 `aioreturn` — retrieve return status of an asynchronous I/O operation19612 **SYNOPSIS**19613 `#include <aiio.h>`19614 `ssize_t aiorreturn(struct aiocb *aiocbp);`19615 **DESCRIPTION**

19616 The `aioreturn()` function shall return the return status associated with the `aiocb` structure
 19617 referenced by the `aiocbp` argument. The return status for an asynchronous I/O operation is the
 19618 value that would be returned by the corresponding `read()`, `write()`, or `fsync()` function call. If the
 19619 error status for the operation is equal to [EINPROGRESS], then the return status for the
 19620 operation is undefined. The `aioreturn()` function may be called exactly once to retrieve the
 19621 return status of a given asynchronous operation; thereafter, if the same `aiocb` structure is used in
 19622 a call to `aioreturn()` or `aiorerror()`, an error may be returned. When the `aiocb` structure referred
 19623 to by `aiocbp` is used to submit another asynchronous operation, then `aioreturn()` may be
 19624 successfully used to retrieve the return status of that operation.

19625 **RETURN VALUE**

19626 If the asynchronous I/O operation has completed, then the return status, as described for `read()`,
 19627 `write()`, and `fsync()`, shall be returned. If the asynchronous I/O operation has not yet completed,
 19628 the results of `aioreturn()` are undefined.

19629 If the `aioreturn()` function fails, it shall return `-1` and set `errno` to indicate the error. +

19630 **ERRORS**19631 The `aioreturn()` function may fail if:

19632 [EINVAL] The `aiocbp` argument does not refer to an asynchronous operation whose
 19633 return status has not yet been retrieved.

19634 **EXAMPLES**

19635 None.

19636 **APPLICATION USAGE**

19637 None.

19638 **RATIONALE**

19639 None.

19640 **FUTURE DIRECTIONS**

19641 None.

19642 **SEE ALSO**

19643 [aiorecancel\(\)](#), [aioreerror\(\)](#), [aiorefsync\(\)](#), [aioreread\(\)](#), [aiorewrite\(\)](#), [close\(\)](#), [exec](#), [exit\(\)](#), [fork\(\)](#), [lio_listio\(\)](#),
 19644 [lseek\(\)](#), [read](#)

19645 XBD [<aiio.h>](#)19646 **CHANGE HISTORY**

19647 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19648 **Issue 6**

19649 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19650 implementation does not support the Asynchronous Input and Output option.

19651 The APPLICATION USAGE section is added.

19652 The [EINVAL] error condition is made optional. This is for consistency with the DESCRIPTION.

19653

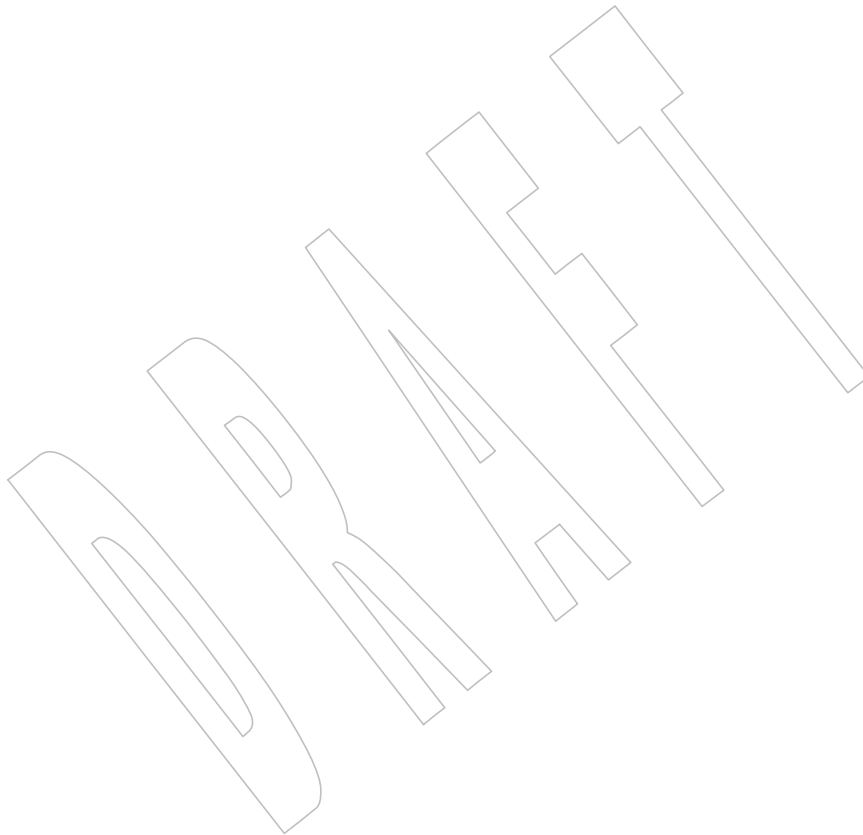
Issue 7

19654

SD5-XSH-ERN-148 is applied.

+

19655

The *aio_return()* function is moved from the Asynchronous Input and Output option to the Base.

19656 **NAME**
 19657 aio_suspend — wait for an asynchronous I/O request

19658 **SYNOPSIS**
 19659 #include <aio.h>
 19660 int aio_suspend(const struct aiocb *const list[], int nent,
 19661 const struct timespec *timeout);

19662 **DESCRIPTION**
 19663 The *aio_suspend()* function shall suspend the calling thread until at least one of the asynchronous
 19664 I/O operations referenced by the *list* argument has completed, until a signal interrupts the
 19665 function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any
 19666 of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is,
 19667 the error status for the operation is not equal to [EINPROGRESS]) at the time of the call, the
 19668 function shall return without suspending the calling thread. The *list* argument is an array of
 19669 pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of
 19670 elements in the array. Each **aiocb** structure pointed to has been used in initiating an
 19671 asynchronous I/O request via *aio_read()*, *aio_write()*, or *lio_listio()*. This array may contain
 19672 NULL pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures
 19673 that have not been used in submitting asynchronous I/O, the effect is undefined.

19674 If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of
 19675 the I/O operations referenced by *list* are completed, then *aio_suspend()* shall return with an error.

19676 MON If the Monotonic Clock option is supported, the clock that shall be used to measure this time
 19677 interval shall be the CLOCK_MONOTONIC clock.

19678 **RETURN VALUE**
 19679 If the *aio_suspend()* function returns after one or more asynchronous I/O operations have
 19680 completed, the function shall return zero. Otherwise, the function shall return a value of -1 and
 19681 set *errno* to indicate the error.

19682 The application may determine which asynchronous I/O completed by scanning the associated
 19683 error and return status using *aio_error()* and *aio_return()*, respectively.

19684 **ERRORS**
 19685 The *aio_suspend()* function shall fail if:

19686 [EAGAIN] No asynchronous I/O indicated in the list referenced by *list* completed in the
 19687 time interval indicated by *timeout*.

19688 [EINTR] A signal interrupted the *aio_suspend()* function. Note that, since each
 19689 asynchronous I/O operation may possibly provoke a signal when it
 19690 completes, this error return may be caused by the completion of one (or more)
 19691 of the very I/O operations being awaited.

19692 **EXAMPLES**
 19693 None.

19694 **APPLICATION USAGE**
 19695 None.

19696 **RATIONALE**
 19697 None.

19698
19699
19700
19701
19702
19703
19704
19705
19706
19707
19708
19709
19710
19711
19712
19713

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read(), *aio_write()*, *lio_listio()*

XBD <**aio.h**>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Asynchronous Input and Output option.

The APPLICATION USAGE section is added.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the CLOCK_MONOTONIC clock, if supported, is used.

Issue 7

The *aio_suspend()* function is moved from the Asynchronous Input and Output option to the Base.

DRAFT

19714 **NAME**

19715 aio_write — asynchronous write to a file

19716 **SYNOPSIS**

19717 #include <aio.h>

19718 int aio_write(struct aiocb *aiocbp);

19719 **DESCRIPTION**19720 The *aio_write()* function shall write *aiocbp->aio_nbytes* to the file associated with
19721 *aiocbp->aio_fildes* from the buffer pointed to by *aiocbp->aio_buf*. The function shall return when
19722 the write request has been initiated or, at a minimum, queued to the file or device.19723 **PIO** If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted
19724 at a priority equal to a base scheduling priority minus *aiocbp->aio_reqprio*. If Thread Execution
19725 Scheduling is not supported, then the base scheduling priority is that of the calling process;
19726 **PIO TPS** otherwise, the base scheduling priority is that of the calling thread.19727 The *aiocbp* argument may be used as an argument to *aio_error()* and *aio_return()* in order to
19728 determine the error status and return status, respectively, of the asynchronous operation while it
19729 is proceeding.19730 The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp->aio_buf* or
19731 the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O
19732 completion, then the behavior is undefined.19733 If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take
19734 place at the absolute position in the file as given by *aio_offset*, as if *lseek()* were called
19735 immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to
19736 SEEK_SET. If O_APPEND is set for the file descriptor, write operations append to the file in the
19737 same order as the calls were made. After a successful call to enqueue an asynchronous I/O
19738 operation, the value of the file offset for the file is unspecified.19739 The *aio_sigevent* member specifies the notification which occurs when the request is completed.19740 The *aiocbp->aio_lio_opcode* field shall be ignored by *aio_write()*.19741 Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.19742 **SIO** If synchronized I/O is enabled on the file associated with *aiocbp->aio_fildes*, the behavior of this
19743 function shall be according to the definitions of synchronized I/O data integrity completion, and
19744 synchronized I/O file integrity completion.19745 For any system action that changes the process memory space while an asynchronous I/O is
19746 outstanding to the address range being changed, the result of that action is undefined.19747 For regular files, no data transfer shall occur past the offset maximum established in the open
19748 file description associated with *aiocbp->aio_fildes*.19749 **RETURN VALUE**19750 The *aio_write()* function shall return the value zero if the I/O operation is successfully queued;
19751 otherwise, the function shall return the value -1 and set *errno* to indicate the error.19752 **ERRORS**19753 The *aio_write()* function shall fail if:19754 [EAGAIN] The requested asynchronous I/O operation was not queued due to system
19755 resource limitations.

19756 Each of the following conditions may be detected synchronously at the time of the call to

19757 *aio_write()*, or asynchronously. If any of the conditions below are detected synchronously, the
 19758 *aio_write()* function shall return `-1` and set *errno* to the corresponding value. If any of the
 19759 conditions below are detected asynchronously, the return status of the asynchronous operation
 19760 shall be set to `-1`, and the error status of the asynchronous operation is set to the corresponding
 19761 value.

19762 [EBADF] The *aioctx->aio_fildes* argument is not a valid file descriptor open for writing.

19763 [EINVAL] The file offset value implied by *aioctx->aio_offset* would be invalid,
 19764 PIO *aioctx->aio_reqprio* is not a valid value, or *aioctx->aio_nbytes* is an invalid
 19765 value.

19766 In the case that the *aio_write()* successfully queues the I/O operation, the return status of the
 19767 asynchronous operation shall be one of the values normally returned by the *write()* function call.
 19768 If the operation is successfully queued but is subsequently canceled or encounters an error, the
 19769 error status for the asynchronous operation contains one of the values normally set by the
 19770 *write()* function call, or one of the following:

19771 [EBADF] The *aioctx->aio_fildes* argument is not a valid file descriptor open for writing.

19772 [EINVAL] The file offset value implied by *aioctx->aio_offset* would be invalid.

19773 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
 19774 *aio_cancel()* request.

19775 The following condition may be detected synchronously or asynchronously:

19776 [EFBIG] The file is a regular file, *aioctx->aio_nbytes* is greater than 0, and the starting
 19777 offset in *aioctx->aio_offset* is at or beyond the offset maximum in the open file
 19778 description associated with *aioctx->aio_fildes*.

19779 EXAMPLES

19780 None.

19781 APPLICATION USAGE

19782 None.

19783 RATIONALE

19784 None.

19785 FUTURE DIRECTIONS

19786 None.

19787 SEE ALSO

19788 *aio_cancel()*, *aio_error()*, *aio_read()*, *aio_return()*, *close()*, *exec*, *exit()*, *fork()*, *lio_listio()*, *lseek()*,
 19789 *write*

19790 XBD <[aio.h](#)>

19791 CHANGE HISTORY

19792 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

19793 Large File Summit extensions are added.

19794 Issue 6

19795 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 19796 implementation does not support the Asynchronous Input and Output option.

19797 The APPLICATION USAGE section is added.

19798 The following new requirements on POSIX implementations derive from alignment with the
 19799 Single UNIX Specification:

19800
19801
19802
19803
19804
19805
19806
19807
19808
19809
19810
19811
19812

- In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs past the offset maximum established in the open file description associated with *aiocbp*→*aio_fildes*.
- The [EFBIG] error is added as part of the large file support extensions.

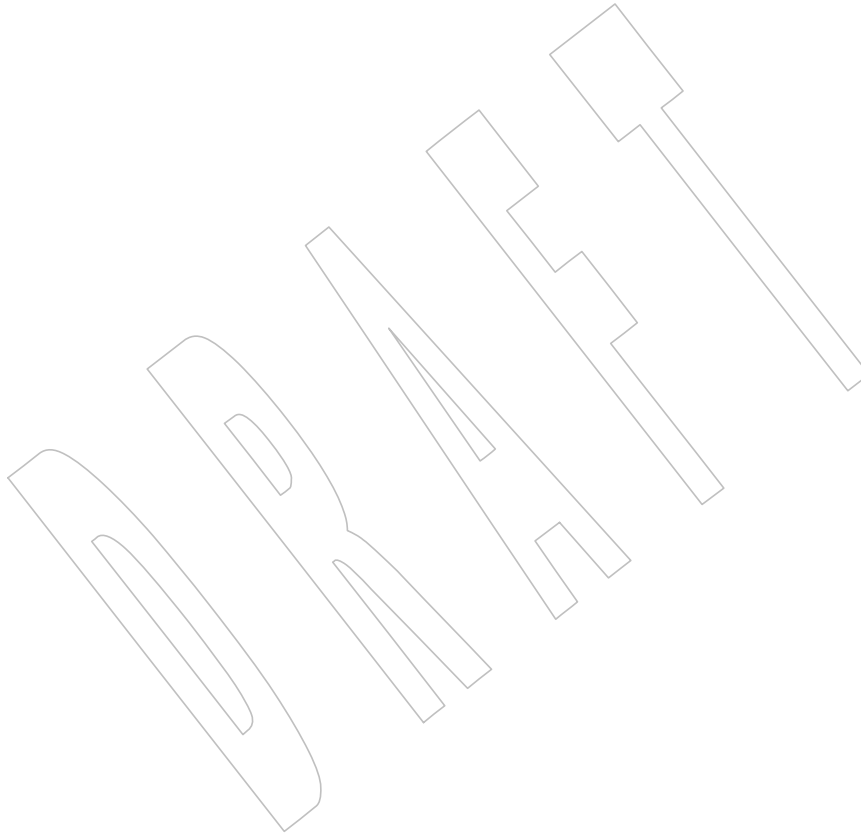
IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/14 is applied, rewording the DESCRIPTION when prioritized I/O is supported to account for threads, and removing the words “to the calling process” in the RETURN VALUE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/15 is applied, updating the [EINVAL] error, so that detection of an [EINVAL] error for an invalid value of *aiocbp*→*aio_reqprio* is only required if the Prioritized Input and Output option is supported.

Issue 7

Austin Group Interpretation 1003.1-2001 #082 is applied.

The *aio_write()* function is moved from the Asynchronous Input and Output option to the Base.



19813 **NAME**

19814 alarm — schedule an alarm signal

19815 **SYNOPSIS**

19816 #include <unistd.h>

19817 unsigned alarm(unsigned *seconds*);19818 **DESCRIPTION**

19819 The *alarm()* function shall cause the system to generate a SIGALRM signal for the process after
 19820 the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays
 19821 may prevent the process from handling the signal as soon as it is generated.

19822 If *seconds* is 0, a pending alarm request, if any, is canceled.

19823 Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner.
 19824 If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time
 19825 at which the SIGALRM signal is generated.

19826 XSI Interactions between *alarm()* and *setitimer()* are unspecified.

19827 **RETURN VALUE**

19828 If there is a previous *alarm()* request with time remaining, *alarm()* shall return a non-zero value
 19829 that is the number of seconds until the previous request would have generated a SIGALRM
 19830 signal. Otherwise, *alarm()* shall return 0.

19831 **ERRORS**

19832 The *alarm()* function is always successful, and no return value is reserved to indicate an error.

19833 **EXAMPLES**

19834 None.

19835 **APPLICATION USAGE**

19836 The *fork()* function clears pending alarms in the child process. A new process image created by
 19837 one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

19838 Application developers should note that the type of the argument *seconds* and the return value of
 19839 *alarm()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces
 19840 Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX},
 19841 which the ISO C standard sets as 65 535, and any application passing a larger value is restricting
 19842 its portability. A different type was considered, but historical implementations, including those
 19843 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

19844 Application developers should be aware of possible interactions when the same process uses
 19845 both the *alarm()* and *sleep()* functions.

19846 **RATIONALE**

19847 Many historical implementations (including Version 7 and System V) allow an alarm to occur up
 19848 to a second early. Other implementations allow alarms up to half a second or one clock tick
 19849 early or do not allow them to occur early at all. The latter is considered most appropriate, since it
 19850 gives the most predictable behavior, especially since the signal can always be delayed for an
 19851 indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument
 19852 as the minimum amount of time they wish to have elapse before the signal.

19853 The term “realtime” here and elsewhere (*sleep()*, *times()*) is intended to mean “wall clock” time
 19854 as common English usage, and has nothing to do with “realtime operating systems”. It is in
 19855 contrast to *virtual time*, which could be misinterpreted if just *time* were used.

19856 In some implementations, including 4.3 BSD, very large values of the *seconds* argument are

19857 silently rounded down to an implementation-specific maximum value. This maximum is large
 19858 enough (to the order of several months) that the effect is not noticeable.

19859 There were two possible choices for alarm generation in multi-threaded applications: generation
 19860 for the calling thread or generation for the process. The first option would not have been
 19861 particularly useful since the alarm state is maintained on a per-process basis and the alarm that
 19862 is established by the last invocation of *alarm()* is the only one that would be active.

19863 Furthermore, allowing generation of an asynchronous signal for a thread would have
 19864 introduced an exception to the overall signal model. This requires a compelling reason in order
 19865 to be justified.

19866 **FUTURE DIRECTIONS**

19867 None.

19868 **SEE ALSO**

19869 *alarm()*, *exec*, *fork()*, *getitimer()*, *pause()*, *sigaction()*, *sleep*

19870 XBD `<signal.h>`, `<unistd.h>`

19871 **CHANGE HISTORY**

19872 First released in Issue 1. Derived from Issue 1 of the SVID.

19873 **Issue 6**

19874 The following new requirements on POSIX implementations derive from alignment with the
 19875 Single UNIX Specification:

- 19876 • The DESCRIPTION is updated to indicate that interactions with the *setitimer()*, *ualarm()*,
 19877 and *usleep()* functions are unspecified.

19878 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/16 is applied, replacing “an
 19879 implementation-defined maximum value” with “an implementation-specific maximum value”
 19880 in the RATIONALE.

19881 **NAME**
 19882 alphasort, scandir — scan a directory

19883 **SYNOPSIS**

```
19884 #include <dirent.h>
19885
19885 int alphasort(const struct dirent **d1, const struct dirent **d2);
19886 int scandir(const char *dir, struct dirent ***namelist,
19887             int (*sel)(const struct dirent *),
19888             int (*compar)(const struct dirent **, const struct dirent **));
```

19889 **DESCRIPTION**

19890 The *alphasort()* function can be used as the comparison function for the *scandir()* function to sort
 19891 the directory entries into alphabetical order. Sorting happens as if by calling the *strcoll()*
 19892 function on the *d_name* element of the **dirent** structures passed as the two parameters. Its
 19893 parameters are the two directory entries, *d1* and *d2*, to compare.

19894 The *scandir()* function shall scan the directory *dir*, calling the function referenced by *sel* on each
 19895 directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored
 19896 in strings allocated as if by a call to *malloc()*, and sorted using *qsort()* with the comparison
 19897 function *compar*, and collected in array *namelist* which shall be allocated as if by a call to *malloc()*.
 19898 If *sel* is a null pointer, all entries shall be selected.

19899 **RETURN VALUE**

19900 Upon successful completion, the *alphasort()* function shall return an integer greater than, equal
 19901 to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is
 19902 lexically greater than, equal to, or less than the directory pointed to by *d2* when both are
 19903 interpreted as appropriate to the current locale. There is no return value reserved to indicate an
 19904 error.

19905 Upon successful completion, the *scandir()* function shall return the number of entries in the
 19906 array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir()*
 19907 function shall return -1.

19908 **ERRORS**

19909 The *scandir()* function shall fail if:

19910 [EACCES] Search permission is denied for the component of the path prefix of *dir* or read
 19911 permission is denied for *dir*.

19912 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dir*
 19913 argument.

19914 [ENAMETOOLONG] The length of the *dir* argument exceeds {PATH_MAX} or a pathname
 19915 component is longer than {NAME_MAX}.

19917 [ENOENT] A component of *dir* does not name an existing directory or *dir* is an empty
 19918 string.

19919 [ENOMEM] Insufficient storage space is available.

19920 [ENOTDIR] A component of *dir* is not a directory.

19921 The *scandir()* function may fail if:

19922 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 19923 resolution of the *dir* argument.

alphasort()

- 19924 [EMFILE] {OPEN_MAX} file descriptors are currently open in the calling process.
- 19925 [ENAMETOOLONG]
 19926 As a result of encountering a symbolic link in resolution of the *dir* argument,
 19927 the length of the substituted pathname string exceeded {PATH_MAX}.
- 19928 [ENFILE] Too many files are currently open in the system.

EXAMPLES

19929 An example to print the files in the current directory:

```

19931 #include <dirent.h>
19932 #include <stdio.h>
19933 #include <stdlib.h>
19934 ...
19935 struct dirent **namelist;
19936 int i,n;
19937
19938     n = scandir(".", &namelist, 0, alphasort);
19939     if (n < 0)
19940         perror("scandir");
19941     else {
19942         for (i = 0; i < n; i++) {
19943             printf("%s\n", namelist[i]->d_name);
19944             free(namelist[i]);
19945         }
19946         free(namelist);
19947     }

```

APPLICATION USAGE

19948 None.

RATIONALE

19949 None.

FUTURE DIRECTIONS

19950 None.

SEE ALSO

19951 [malloc\(\)](#), [qsort\(\)](#), [strcoll\(\)](#)

19952 XBD [<dirent.h>](#)

CHANGE HISTORY

19953 First released in Issue 7.

19959 **NAME**
 19960 asctime, asctime_r — convert date and time to a string

19961 **SYNOPSIS**

```
19962 OB #include <time.h>
19963 char *asctime(const struct tm *timeptr);
19964 OB CX char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

19965 **DESCRIPTION**

19966 CX For *asctime()*: The functionality described on this reference page is aligned with the ISO C |
 19967 standard. Any conflict between the requirements described here and the ISO C standard is |
 19968 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

19969 The *asctime()* function shall convert the broken-down time in the structure pointed to by *timeptr*
 19970 into a string in the form:

```
19971 Sun Sep 16 01:03:52 1973\n\0
```

19972 using the equivalent of the following algorithm:

```
19973 char *asctime(const struct tm *timeptr)
19974 {
19975     static char wday_name[7][3] = {
19976         "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
19977     };
19978     static char mon_name[12][3] = {
19979         "Jan", "Feb", "Mar", "Apr", "May", "Jun",
19980         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
19981     };
19982     static char result[26];
19983     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
19984         wday_name[timeptr->tm_wday],
19985         mon_name[timeptr->tm_mon],
19986         timeptr->tm_mday, timeptr->tm_hour,
19987         timeptr->tm_min, timeptr->tm_sec,
19988         1900 + timeptr->tm_year);
19989     return result;
19990 }
```

19991 However, the behavior is undefined if *timeptr->tm_wday* or *timeptr->tm_mon* are not within the
 19992 normal ranges as defined in **<time.h>**, or if *timeptr->tm_year* exceeds {INT_MAX}-1990, or if the
 19993 above algorithm would attempt to generate more than 26 bytes of output (including the
 19994 terminating null).

19995 The **tm** structure is defined in the **<time.h>** header.

19996 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 19997 objects: a broken-down time structure and an array of type **char**. Execution of any of the
 19998 functions may overwrite the information returned in either of these objects by any of the other
 19999 functions.

20000 The *asctime()* function need not be thread-safe. A function that is not required to be thread-safe
 20001 is not required to be reentrant.

20002 The *asctime_r()* function shall convert the broken-down time in the structure pointed to by *tm*

asctime()

System Interfaces

20003 into a string (of the same form as that returned by *asctime()*, and with the same undefined
 20004 behavior when input or output is out of range) that is placed in the user-supplied buffer pointed
 20005 to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

RETURN VALUE

20006
 20007 CX Upon successful completion, *asctime()* shall return a pointer to the string. If the function is
 20008 unsuccessful, it shall return NULL.

20009 Upon successful completion, *asctime_r()* shall return a pointer to a character string containing
 20010 the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful,
 20011 it shall return NULL.

ERRORS

20012 No errors are defined.
 20013

EXAMPLES

20014 None.
 20015

APPLICATION USAGE

20016 These functions are included only for compatibility with older implementations. They have
 20017 undefined behavior if the resulting string would be too long, so the use of these functions
 20018 should be discouraged. On implementations that do not detect output string length overflow, it
 20019 is possible to overflow the output buffers in such a way as to cause applications to fail, or
 20020 possible system security violations. Also, these functions do not support localized date and time
 20021 formats. To avoid these problems, applications should use *strftime()* to generate strings from
 20022 broken-down times.
 20023

20024 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

20025 The *asctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead
 20026 of possibly using a static data area that may be overwritten by each call.

RATIONALE

20027 The standard developers decided to mark the *asctime()* and *asctime_r()* functions obsolescent
 20028 even though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C
 20029 standard also provides the *strftime()* function which can be used to avoid these problems.
 20030

FUTURE DIRECTIONS

20031 These functions may be removed in a future version.
 20032

SEE ALSO

20033 *clock()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time*, *utime()*

20034 XBD <[time.h](#)>
 20035

CHANGE HISTORY

20036 First released in Issue 1. Derived from Issue 1 of the SVID.
 20037

Issue 5

20038 Normative text previously in the APPLICATION USAGE section is moved to the
 20039 DESCRIPTION.
 20040

20041 The *asctime_r()* function is included for alignment with the POSIX Threads Extension.

20042 A note indicating that the *asctime()* function need not be reentrant is added to the
 20043 DESCRIPTION.

Issue 6

20044 The *asctime_r()* function is marked as part of the Thread-Safe Functions option.
 20045

20046 Extensions beyond the ISO C standard are marked.
 20047

20047 The APPLICATION USAGE section is updated to include a note on the thread-safe function and

20048

its avoidance of possibly using a static data area.

20049

The DESCRIPTION of *asctime_r()* is updated to describe the format of the string returned.

20050

20051

The **restrict** keyword is added to the *asctime_r()* prototype for alignment with the ISO/IEC 9899:1999 standard.

20052

20053

20054

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/17 is applied, adding the CX extension in the RETURN VALUE section requiring that if the *asctime()* function is unsuccessful it returns NULL.

20055

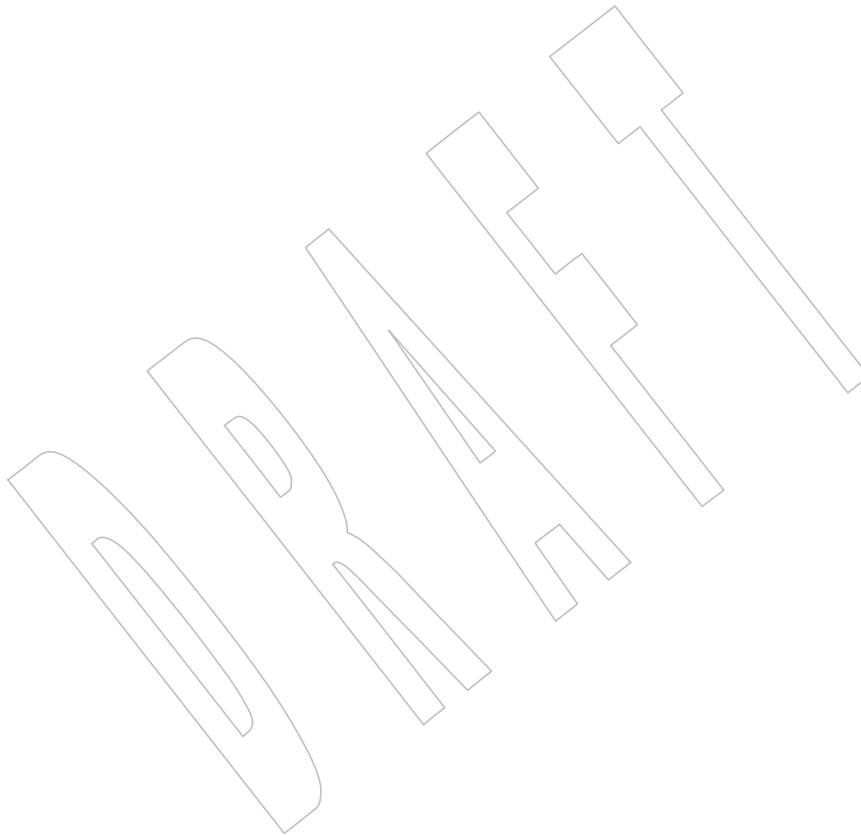
Issue 7

20056

Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

20057

The *asctime_r()* function is moved from the Thread-Safe Functions option to the Base.



20058 **NAME**
 20059 asin, asinf, asinl — arc sine function

20060 **SYNOPSIS**
 20061 #include <math.h>
 20062 double asin(double x);
 20063 float asinf(float x);
 20064 long double asinl(long double x);

20065 **DESCRIPTION**
 20066 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20067 conflict between the requirements described here and the ISO C standard is unintentional. This
 20068 volume of POSIX.1-200x defers to the ISO C standard.

20069 These functions shall compute the principal value of the arc sine of their argument x . The value
 20070 of x should be in the range $[-1,1]$.

20071 An application wishing to check for error situations should set *errno* to zero and call
 20072 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20073 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20074 zero, an error has occurred.

20075 **RETURN VALUE**
 20076 Upon successful completion, these functions shall return the arc sine of x , in the range
 20077 $[-\pi/2, \pi/2]$ radians.

20078 MX For finite values of x not in the range $[-1,1]$, a domain error shall occur, and either a NaN (if
 20079 supported), or an implementation-defined value shall be returned.

20080 MX If x is NaN, a NaN shall be returned.

20081 If x is ± 0 , x shall be returned.

20082 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 20083 defined value shall be returned.

20084 If x is subnormal, a range error may occur and x should be returned.

20085 **ERRORS**
 20086 These functions shall fail if:

20087 MX Domain Error The x argument is finite and is not in the range $[-1,1]$, or is $\pm\text{Inf}$.
 20088 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20089 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 20090 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20091 shall be raised.

20092 These functions may fail if:

20093 MX Range Error The value of x is subnormal.
 20094 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20095 then *errno* shall be set to [ERANGE]. If the integer expression
 20096 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 20097 floating-point exception shall be raised.

20098
20099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *isnan()*, *sin()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *asinf()* and *asinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

20121 **NAME**

20122 asinh, asinhf, asinhl — inverse hyperbolic sine functions

20123 **SYNOPSIS**

```
20124 #include <math.h>
20125
20125 double asinh(double x);
20126 float asinhf(float x);
20127 long double asinhl(long double x);
```

20128 **DESCRIPTION**

20129 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20130 conflict between the requirements described here and the ISO C standard is unintentional. This
 20131 volume of POSIX.1-200x defers to the ISO C standard.

20132 These functions shall compute the inverse hyperbolic sine of their argument x .

20133 An application wishing to check for error situations should set *errno* to zero and call
 20134 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20135 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20136 zero, an error has occurred.

20137 **RETURN VALUE**

20138 Upon successful completion, these functions shall return the inverse hyperbolic sine of their
 20139 argument.

20140 MX If x is NaN, a NaN shall be returned.

20141 If x is ± 0 , or $\pm\text{Inf}$, x shall be returned.

20142 If x is subnormal, a range error may occur and x should be returned.

20143 **ERRORS**

20144 These functions may fail if:

20145 MX **Range Error** The value of x is subnormal.

20146 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20147 then *errno* shall be set to [ERANGE]. If the integer expression
 20148 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 20149 floating-point exception shall be raised.

20150 **EXAMPLES**

20151 None.

20152 **APPLICATION USAGE**

20153 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 20154 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20155 **RATIONALE**

20156 None.

20157 **FUTURE DIRECTIONS**

20158 None.

20159 **SEE ALSO**

20160 *feclearexcept()*, *fetestexcept()*, *sinh()*

20161 XBD Section 4.19 (on page 104), **<math.h>**

20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

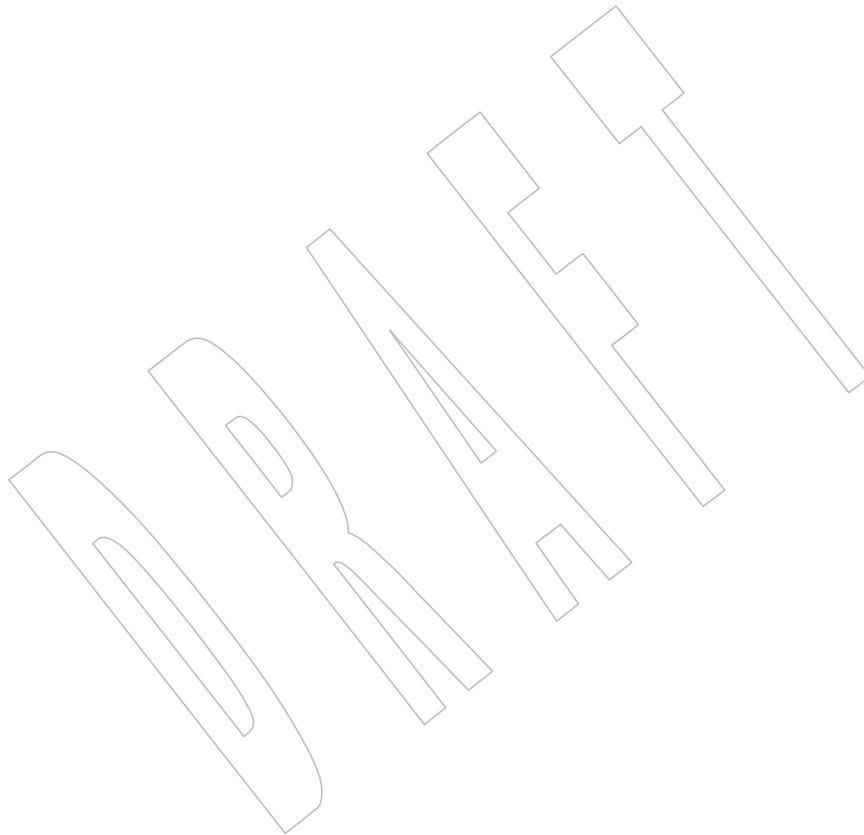
Issue 6

The *asinh()* function is no longer marked as an extension.

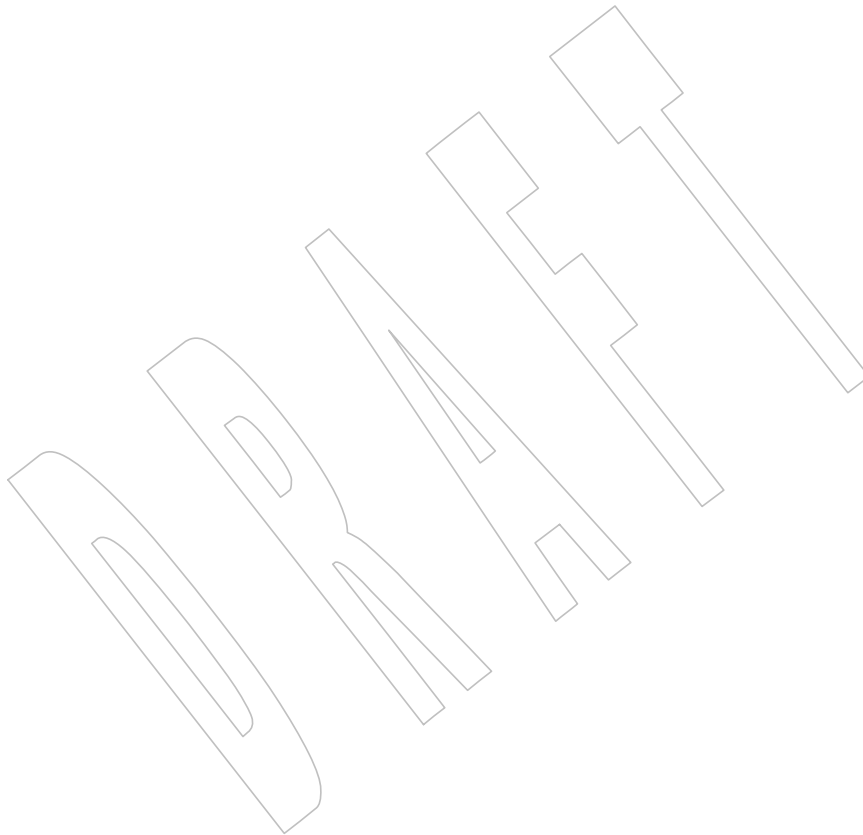
The *asinhf()* and *asinhll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.



20174

NAME20175 *asinl* — arc sine function
20176**SYNOPSIS**20177 `#include <math.h>`20178 `long double asinl(long double x);`
20179**DESCRIPTION**20180 Refer to *asin()*.
20181

20182 **NAME**

20183 assert — insert program diagnostics

20184 **SYNOPSIS**

20185 #include <assert.h>

20186 void assert(*scalar expression*);20187 **DESCRIPTION**

20188 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20189 conflict between the requirements described here and the ISO C standard is unintentional. This
 20190 volume of POSIX.1-200x defers to the ISO C standard.

20191 The *assert()* macro shall insert diagnostics into programs; it shall expand to a **void** expression.
 20192 When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal
 20193 to 0), *assert()* shall write information about the particular call that failed on *stderr* and shall call
 20194 *abort()*.

20195 The information written about the call that failed shall include the text of the argument, the
 20196 name of the source file, the source file line number, and the name of the enclosing function; the
 20197 latter are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__` and of
 20198 the identifier `__func__`.

20199 Forcing a definition of the name `NDEBUG`, either from the compiler command line or with the
 20200 preprocessor control statement `#define NDEBUG` ahead of the `#include <assert.h>` statement,
 20201 shall stop assertions from being compiled into the program.

20202 **RETURN VALUE**20203 The *assert()* macro shall not return a value.20204 **ERRORS**

20205 No errors are defined.

20206 **EXAMPLES**

20207 None.

20208 **APPLICATION USAGE**

20209 None.

20210 **RATIONALE**

20211 None.

20212 **FUTURE DIRECTIONS**

20213 None.

20214 **SEE ALSO**20215 *abort()*, *stdin*

20216 XBD <assert.h>

20217 **CHANGE HISTORY**

20218 First released in Issue 1. Derived from Issue 1 of the SVID.

20219 **Issue 6**20220 The prototype for the *expression* argument to *assert()* is changed from **int** to **scalar** for alignment
 20221 with the ISO/IEC 9899:1999 standard.20222 The DESCRIPTION of *assert()* is updated for alignment with the ISO/IEC 9899:1999 standard.

20223 **NAME**

20224 atan, atanf, atanl — arc tangent function

20225 **SYNOPSIS**

```
20226 #include <math.h>
20227
20227 double atan(double x);
20228 float atanf(float x);
20229 long double atanl(long double x);
```

20230 **DESCRIPTION**

20231 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20232 conflict between the requirements described here and the ISO C standard is unintentional. This
 20233 volume of POSIX.1-200x defers to the ISO C standard.

20234 These functions shall compute the principal value of the arc tangent of their argument x .

20235 An application wishing to check for error situations should set *errno* to zero and call
 20236 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20237 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20238 zero, an error has occurred.

20239 **RETURN VALUE**

20240 Upon successful completion, these functions shall return the arc tangent of x in the range
 20241 $[-\pi/2, \pi/2]$ radians.

20242 MX If x is NaN, a NaN shall be returned.

20243 If x is ± 0 , x shall be returned.

20244 If x is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.

20245 If x is subnormal, a range error may occur and x should be returned.

20246 **ERRORS**

20247 These functions may fail if:

20248 MX **Range Error** The value of x is subnormal.

20249 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20250 then *errno* shall be set to [ERANGE]. If the integer expression
 20251 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 20252 floating-point exception shall be raised.

20253 **EXAMPLES**

20254 None.

20255 **APPLICATION USAGE**

20256 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 20257 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20258 **RATIONALE**

20259 None.

20260 **FUTURE DIRECTIONS**

20261 None.

20262
20263
20264
20265
20266
20267
20268
20269
20270
20271
20272
20273
20274
20275

SEE ALSO

atan2(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

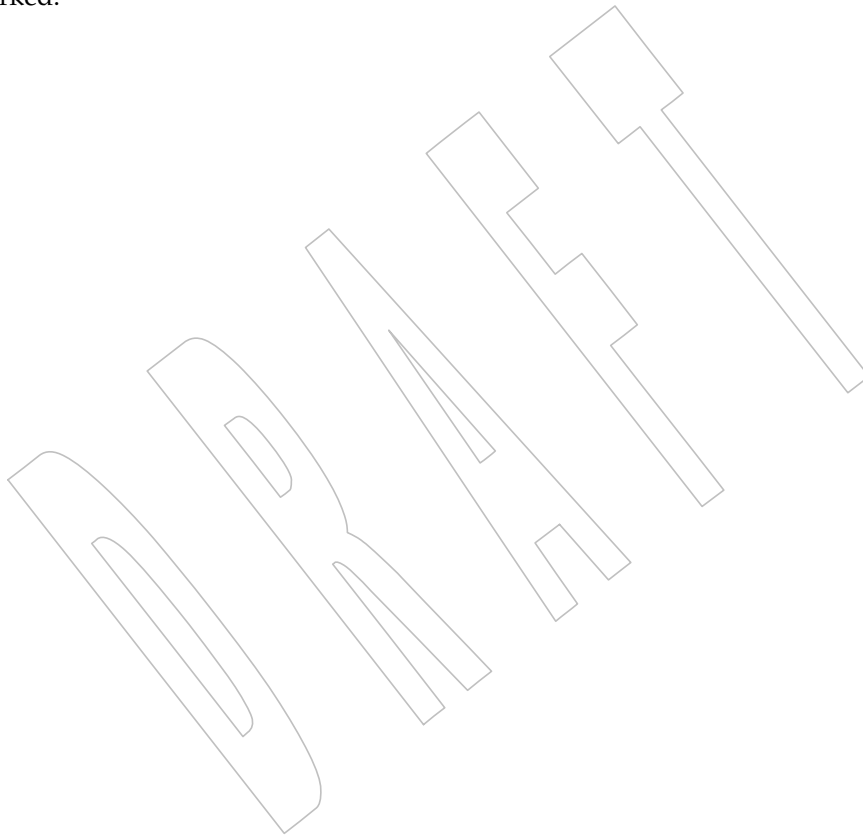
The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *atanf()* and *atanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.



20276 **NAME**

20277 atan2, atan2f, atan2l — arc tangent functions

20278 **SYNOPSIS**

20279 #include <math.h>

20280 double atan2(double y, double x);

20281 float atan2f(float y, float x);

20282 long double atan2l(long double y, long double x);

20283 **DESCRIPTION**

20284 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20285 conflict between the requirements described here and the ISO C standard is unintentional. This
 20286 volume of POSIX.1-200x defers to the ISO C standard.

20287 These functions shall compute the principal value of the arc tangent of y/x , using the signs of
 20288 both arguments to determine the quadrant of the return value.

20289 An application wishing to check for error situations should set *errno* to zero and call
 20290 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20291 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20292 zero, an error has occurred.

20293 **RETURN VALUE**

20294 Upon successful completion, these functions shall return the arc tangent of y/x in the range
 20295 $[-\pi, \pi]$ radians.

20296 If y is ± 0 and x is < 0 , $\pm\pi$ shall be returned.20297 If y is ± 0 and x is > 0 , ± 0 shall be returned.20298 If y is < 0 and x is ± 0 , $-\pi/2$ shall be returned.20299 If y is > 0 and x is ± 0 , $\pi/2$ shall be returned.20300 If x is 0, a pole error shall not occur.20301 MX If either x or y is NaN, a NaN shall be returned.20302 If the result underflows, a range error may occur and y/x should be returned.20303 If y is ± 0 and x is -0 , $\pm\pi$ shall be returned.20304 If y is ± 0 and x is $+0$, ± 0 shall be returned.20305 For finite values of $\pm y > 0$, if x is $-\text{Inf}$, $\pm\pi$ shall be returned.20306 For finite values of $\pm y > 0$, if x is $+\text{Inf}$, ± 0 shall be returned.20307 For finite values of x , if y is $\pm\text{Inf}$, $\pm\pi/2$ shall be returned.20308 If y is $\pm\text{Inf}$ and x is $-\text{Inf}$, $\pm 3\pi/4$ shall be returned.20309 If y is $\pm\text{Inf}$ and x is $+\text{Inf}$, $\pm\pi/4$ shall be returned.

20310 If both arguments are 0, a domain error shall not occur.

20311 **ERRORS**

20312 These functions may fail if:

20313 MX **Range Error** The result underflows.

20314 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20315 then *errno* shall be set to [ERANGE]. If the integer expression

20316
20317

`(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow floating-point exception shall be raised.

20318

EXAMPLES

20319

Converting Cartesian to Polar Coordinates System20320
20321
20322

The function below uses `atan2()` to convert a 2d vector expressed in cartesian coordinates (x,y) to the polar coordinates $(rho,theta)$. There are other ways to compute the angle $theta$, using `asin()` or `atan()`. However, `atan2()` presents here two advantages:

20323

- The angle's quadrant is automatically determined.

20324

- The singular cases $(0,y)$ are taken into account.

20325
20326

Finally, this example uses `hypot()` rather than `sqrt()` since it is better for special cases; see `hypot()` for more information.

20327

```
#include <math.h>
```

20328

```
void
```

20329

```
cartesian_to_polar(const double x, const double y,
```

20330

```
double *rho, double *theta
```

20331

```
)
```

20332

```
{
```

20333

```
*rho = hypot (x,y); /* better than sqrt(x*x+y*y) */
```

20334

```
*theta = atan2 (y,x);
```

20335

```
}
```

20336

APPLICATION USAGE20337
20338

On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling & MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

20339

RATIONALE

20340

None.

20341

FUTURE DIRECTIONS

20342

None.

20343

SEE ALSO

20344

[acos\(\)](#), [asin\(\)](#), [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [hypot\(\)](#), [isnan\(\)](#), [sqrt\(\)](#), [tan\(\)](#)

20345

XBD Section 4.19 (on page 104), [<math.h>](#)

20346

CHANGE HISTORY

20347

First released in Issue 1. Derived from Issue 1 of the SVID.

20348

Issue 5

20349

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

20350

20351

Issue 6

20352

The `atan2f()` and `atan2l()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

20353

20354

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard, and the IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

20355

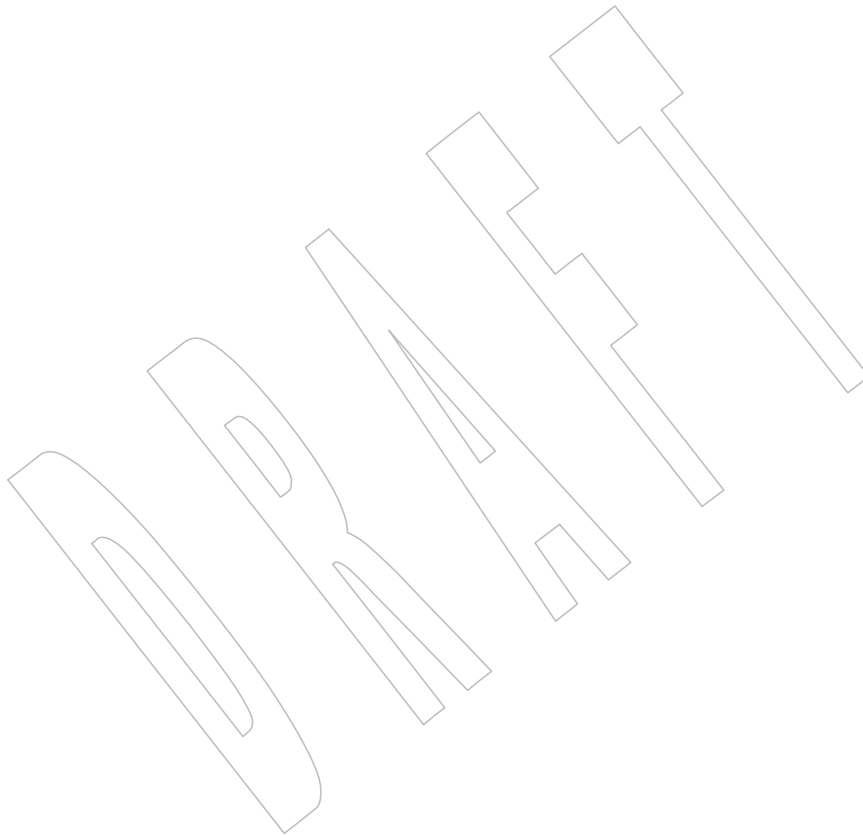
20356

20357

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/18 is applied, adding to the EXAMPLES section.

20358

20359

NAME20361 `atanf` — arc tangent function**SYNOPSIS**20363 `#include <math.h>`20364 `float atanf(float x);`**DESCRIPTION**20366 Refer to *atan()*.

20367 **NAME**

20368 atanh, atanhf, atanh1 — inverse hyperbolic tangent functions

20369 **SYNOPSIS**

```
20370 #include <math.h>
20371
20371 double atanh(double x);
20372 float atanhf(float x);
20373 long double atanh1(long double x);
```

20374 **DESCRIPTION**

20375 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20376 conflict between the requirements described here and the ISO C standard is unintentional. This
 20377 volume of POSIX.1-200x defers to the ISO C standard.

20378 These functions shall compute the inverse hyperbolic tangent of their argument x .

20379 An application wishing to check for error situations should set *errno* to zero and call
 20380 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 20381 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 20382 zero, an error has occurred.

20383 **RETURN VALUE**

20384 Upon successful completion, these functions shall return the inverse hyperbolic tangent of their
 20385 argument.

20386 If x is ± 1 , a pole error shall occur, and *atanh()*, *atanhf()*, and *atanh1()* shall return the value of the
 20387 macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the
 20388 correct value of the function.

20389 MX For finite $|x| > 1$, a domain error shall occur, and either a NaN (if supported), or an
 20390 implementation-defined value shall be returned.

20391 MX If x is NaN, a NaN shall be returned.

20392 If x is ± 0 , x shall be returned.

20393 If x is $\pm \text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 20394 defined value shall be returned.

20395 If x is subnormal, a range error may occur and x should be returned.

20396 **ERRORS**

20397 These functions shall fail if:

20398 MX Domain Error The x argument is finite and not in the range $[-1, 1]$, or is $\pm \text{Inf}$.
 20399 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20400 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 20401 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 20402 shall be raised.

20403 Pole Error The x argument is ± 1 .

20404 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 20405 then *errno* shall be set to [ERANGE]. If the integer expression
 20406 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 20407 floating-point exception shall be raised.

20408

These functions may fail if:

20409

MX

Range Error The value of x is subnormal.

20410

20411

20412

20413

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to [ERANGE]. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

20414

EXAMPLES

20415

None.

20416

APPLICATION USAGE

20417

20418

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

20419

RATIONALE

20420

None.

20421

FUTURE DIRECTIONS

20422

None.

20423

SEE ALSO

20424

feclearexcept(), *fetestexcept()*, *tanh()*

20425

XBD Section 4.19 (on page 104), **<math.h>**

20426

CHANGE HISTORY

20427

First released in Issue 4, Version 2.

20428

Issue 5

20429

Moved from X/OPEN UNIX extension to BASE.

20430

Issue 6

20431

The *atanh()* function is no longer marked as an extension.

20432

20433

The *atanhf()* and *atanhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

20434

20435

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

20436

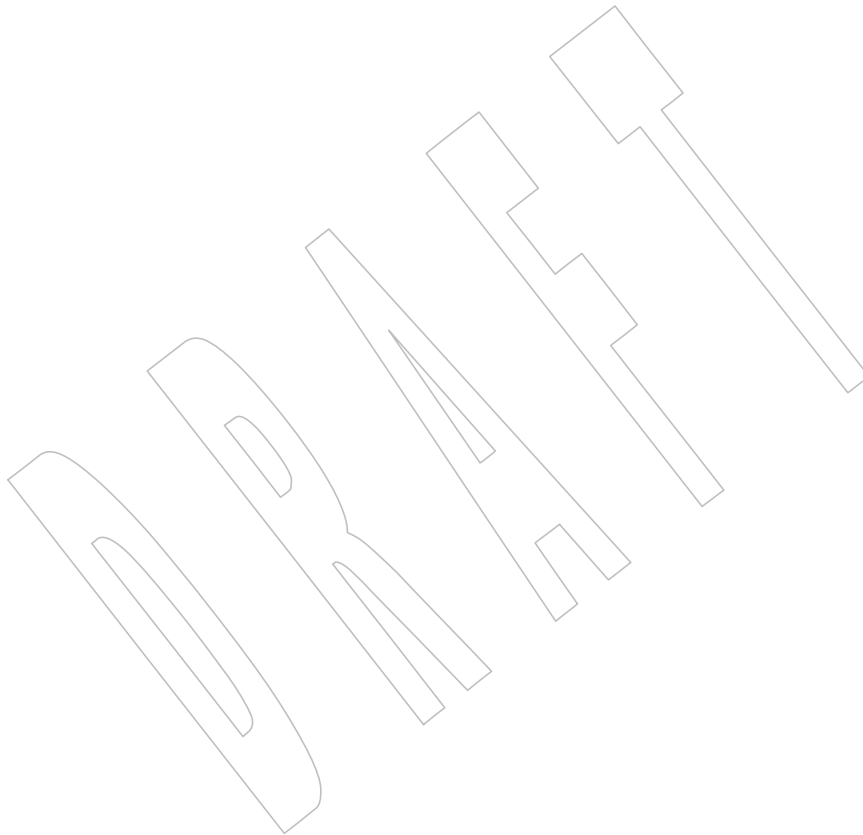
20437

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

20438 **NAME**
20439 atanl — arc tangent function

20440 **SYNOPSIS**
20441 #include <math.h>
20442 long double atanl(long double x);

20443 **DESCRIPTION**
20444 Refer to *atan()*.



20445 **NAME**
 20446 `atexit` — register a function to run at process termination

20447 **SYNOPSIS**
 20448 `#include <stdlib.h>`
 20449 `int atexit(void (*func)(void));`

20450 **DESCRIPTION**

20451 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20452 conflict between the requirements described here and the ISO C standard is unintentional. This
 20453 volume of POSIX.1-200x defers to the ISO C standard.

20454 The `atexit()` function shall register the function pointed to by `func`, to be called without
 20455 arguments at normal program termination. At normal program termination, all functions
 20456 registered by the `atexit()` function shall be called, in the reverse order of their registration, except
 20457 that a function is called after any previously registered functions that had already been called at
 20458 the time it was registered. Normal termination occurs either by a call to `exit()` or a return from
 20459 `main()`.

20460 At least 32 functions can be registered with `atexit()`.

20461 CX After a successful call to any of the `exec` functions, any functions previously registered by `atexit()`
 20462 shall no longer be registered.

20463 **RETURN VALUE**

20464 Upon successful completion, `atexit()` shall return 0; otherwise, it shall return a non-zero value.

20465 **ERRORS**

20466 No errors are defined.

20467 **EXAMPLES**

20468 None.

20469 **APPLICATION USAGE**

20470 The functions registered by a call to `atexit()` must return to ensure that all registered functions
 20471 are called.

20472 The application should call `sysconf()` to obtain the value of {ATEXIT_MAX}, the number of
 20473 functions that can be registered. There is no way for an application to tell how many functions
 20474 have already been registered with `atexit()`.

20475 Since the behavior is undefined if the `exit()` function is called more than once, portable
 20476 applications calling `atexit()` must ensure that the `exit()` function is not called at normal process
 20477 termination when all functions registered by the `atexit()` function are called.

20478 All functions registered by the `atexit()` function are called at normal process termination, which
 20479 occurs by a call to the `exit()` function or a return from `main()` or on the last thread termination,
 20480 when the behavior is as if the implementation called `exit()` with a zero argument at thread
 20481 termination time.

20482 If, at normal process termination, a function registered by the `atexit()` function is called and a
 20483 portable application needs to stop further `exit()` processing, it must call the `_exit()` function or
 20484 the `_Exit()` function or one of the functions which cause abnormal process termination.

20485
20486
20487
20488
20489
20490
20491
20492
20493
20494
20495
20496
20497
20498

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit()*, *sysconf()*

XBD <stdlib.h>

CHANGE HISTORY

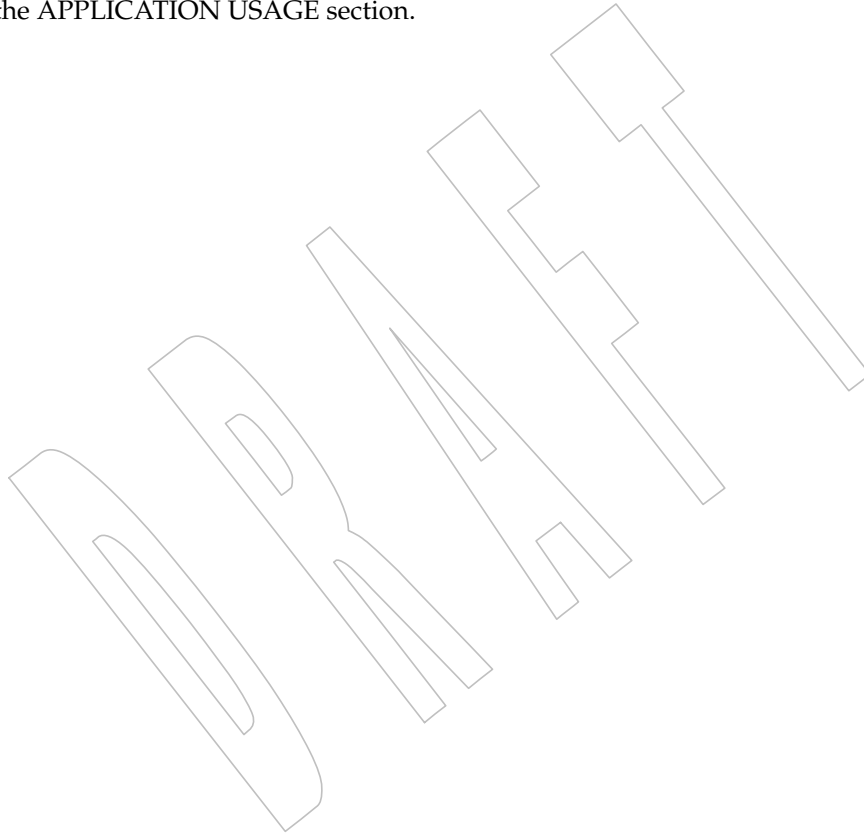
First released in Issue 4. Derived from the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/19 is applied, adding further clarification to the APPLICATION USAGE section.



20499 **NAME**
 20500 `atof` — convert a string to a double-precision number

20501 **SYNOPSIS**
 20502 `#include <stdlib.h>`

20503 `double atof(const char *str);`

20504 **DESCRIPTION**

20505 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20506 conflict between the requirements described here and the ISO C standard is unintentional. This
 20507 volume of POSIX.1-200x defers to the ISO C standard.

20508 The call `atof(str)` shall be equivalent to:

20509 `strtod(str, (char **)NULL),`

20510 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 20511 undefined.

20512 **RETURN VALUE**

20513 The `atof()` function shall return the converted value if the value can be represented.

20514 **ERRORS**

20515 No errors are defined.

20516 **EXAMPLES**

20517 None.

20518 **APPLICATION USAGE**

20519 The `atof()` function is subsumed by `strtod()` but is retained because it is used extensively in
 20520 existing code. If the number is not known to be in range, `strtod()` should be used because `atof()`
 20521 is not required to perform any error checking.

20522 **RATIONALE**

20523 None.

20524 **FUTURE DIRECTIONS**

20525 None.

20526 **SEE ALSO**

20527 [*strtod\(\)*](#)

20528 XBD [*<stdlib.h>*](#)

20529 **CHANGE HISTORY**

20530 First released in Issue 1. Derived from Issue 1 of the SVID.

20531 **NAME**
 20532 `atoi` — convert a string to an integer

20533 **SYNOPSIS**
 20534 `#include <stdlib.h>`
 20535 `int atoi(const char *str);`

20536 **DESCRIPTION**
 20537 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20538 conflict between the requirements described here and the ISO C standard is unintentional. This
 20539 volume of POSIX.1-200x defers to the ISO C standard.

20540 The call `atoi(str)` shall be equivalent to:
 20541 `(int) strtol(str, (char **)NULL, 10)`
 20542 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 20543 undefined.

20544 **RETURN VALUE**
 20545 The `atoi()` function shall return the converted value if the value can be represented.

20546 **ERRORS**
 20547 No errors are defined.

20548 **EXAMPLES**

20549 **Converting an Argument**

20550 The following example checks for proper usage of the program. If there is an argument and the
 20551 decimal conversion of this argument (obtained using `atoi()`) is greater than 0, then the program
 20552 has a valid number of minutes to wait for an event.

```
20553 #include <stdlib.h>
20554 #include <stdio.h>
20555 ...
20556 int minutes_to_event;
20557 ...
20558 if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
20559     fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
20560 }
20561 ...
```

20562 **APPLICATION USAGE**
 20563 The `atoi()` function is subsumed by `strtol()` but is retained because it is used extensively in
 20564 existing code. If the number is not known to be in range, `strtol()` should be used because `atoi()` is
 20565 not required to perform any error checking.

20566 **RATIONALE**
 20567 None.

20568 **FUTURE DIRECTIONS**
 20569 None.

atoi()

20570

SEE ALSO

20571

strtol()

20572

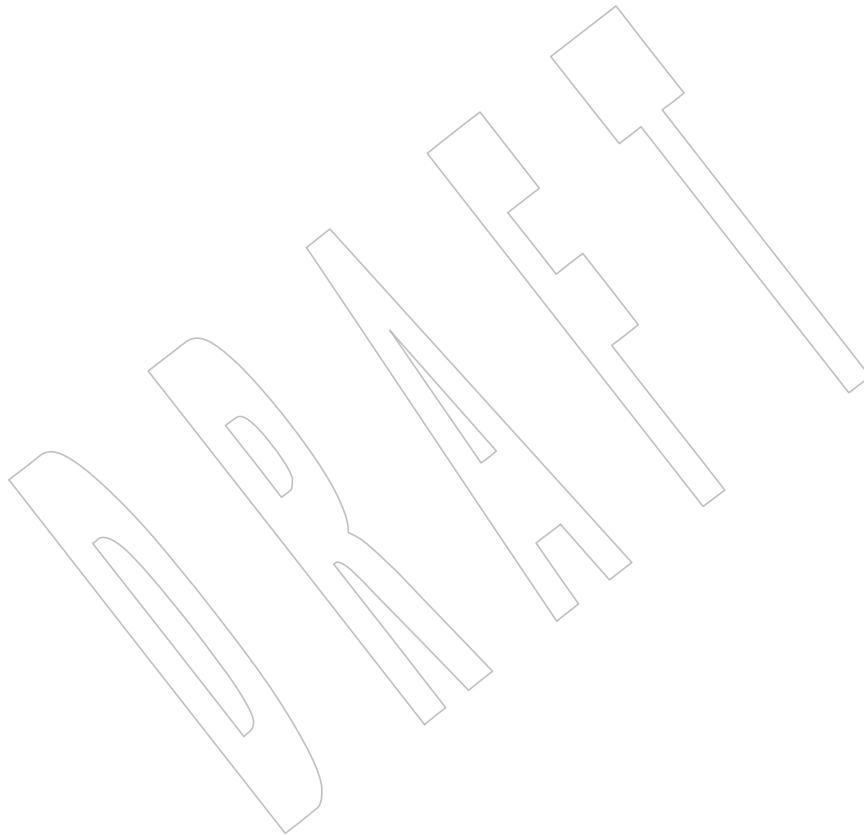
XBD <stdlib.h>

20573

CHANGE HISTORY

20574

First released in Issue 1. Derived from Issue 1 of the SVID.



20575 **NAME**
 20576 `atol, atoll` — convert a string to a long integer

20577 **SYNOPSIS**
 20578 `#include <stdlib.h>`
 20579 `long atol(const char *str);`
 20580 `long long atoll(const char *nptr);`

20581 **DESCRIPTION**
 20582 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20583 conflict between the requirements described here and the ISO C standard is unintentional. This
 20584 volume of POSIX.1-200x defers to the ISO C standard.

20585 The call `atol(str)` shall be equivalent to:
 20586 `strtol(str, (char **)NULL, 10)`

20587 The call `atoll(nptr)` shall be equivalent to:
 20588 `strtoll(nptr, (char **)NULL, 10)`

20589 except that the handling of errors may differ. If the value cannot be represented, the behavior is
 20590 undefined.

20591 **RETURN VALUE**
 20592 These functions shall return the converted value if the value can be represented.

20593 **ERRORS**
 20594 No errors are defined.

20595 **EXAMPLES**
 20596 None.

20597 **APPLICATION USAGE**
 20598 The `atol()` function is subsumed by `strtol()` but is retained because it is used extensively in
 20599 existing code. If the number is not known to be in range, `strtol()` should be used because `atol()` is
 20600 not required to perform any error checking.

20601 **RATIONALE**
 20602 None.

20603 **FUTURE DIRECTIONS**
 20604 None.

20605 **SEE ALSO**
 20606 [strtol\(\)](#)
 20607 XBD [<stdlib.h>](#)

20608 **CHANGE HISTORY**
 20609 First released in Issue 1. Derived from Issue 1 of the SVID.

20610 **Issue 6**
 20611 The `atoll()` function is added for alignment with the ISO/IEC 9899:1999 standard.

20612 **Issue 7**
 20613 SD5-XSH-ERN-61 is applied, correcting the DESCRIPTION of `atoll()`.

20614 **NAME**20615 `basename` — return the last component of a pathname20616 **SYNOPSIS**

```
20617 XSI #include <libgen.h>
20618 char *basename(char *path);
```

20619 **DESCRIPTION**

20620 The `basename()` function shall take the pathname pointed to by `path` and return a pointer to the
 20621 final component of the pathname, deleting any trailing `'/'` characters.

20622 If the string pointed to by `path` consists entirely of the `'/'` character, `basename()` shall return a
 20623 pointer to the string `"/"`. If the string pointed to by `path` is exactly `"/"`, it is implementation-
 20624 defined whether `'/'` or `"/"` is returned.

20625 If `path` is a null pointer or points to an empty string, `basename()` shall return a pointer to the
 20626 string `"."`.

20627 The `basename()` function may modify the string pointed to by `path`, and may return a pointer to
 20628 static storage that may then be overwritten by a subsequent call to `basename()`.

20629 The `basename()` function need not be thread-safe. A function that is not required to be thread-safe
 20630 is not required to be reentrant.

20631 **RETURN VALUE**

20632 The `basename()` function shall return a pointer to the final component of `path`.

20633 **ERRORS**

20634 No errors are defined.

20635 **EXAMPLES**20636 **Using basename()**

20637 The following program fragment returns a pointer to the value `lib`, which is the base name of
 20638 `/usr/lib`.

```
20639 #include <libgen.h>
20640 ...
20641 char *name = "/usr/lib";
20642 char *base;
20643
20644 base = basename(name);
20645 ...
```

20645 **Sample Input and Output Strings for basename()**

20646 In the following table, the input string is the value pointed to by `path`, and the output string is
 20647 the return value of the `basename()` function.

Input String	Output String
<code>"/usr/lib"</code>	<code>"lib"</code>
<code>"/usr/"</code>	<code>"usr"</code>
<code>"/"</code>	<code>"/"</code>
<code>"/" / " /" / " /"</code>	<code>"/"</code>
<code>"/usr//lib//"</code>	<code>"lib"</code>

20654 **APPLICATION USAGE**

20655 None.

20656 **RATIONALE**

20657 None.

20658 **FUTURE DIRECTIONS**

20659 None.

20660 **SEE ALSO**20661 *dirname*

20662 XBD <libgen.h>

20663 XCU *basename*20664 **CHANGE HISTORY**

20665 First released in Issue 4, Version 2.

20666 **Issue 5**

20667 Moved from X/OPEN UNIX extension to BASE.

20668 Normative text previously in the APPLICATION USAGE section is moved to the
20669 DESCRIPTION.

20670 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

20671 **Issue 6**

20672 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

20673 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/20 is applied, changing the
20674 DESCRIPTION to make it clear that the string referenced is the string pointed to by *path*.

bind()20675 **NAME**20676 `bind` — bind a name to a socket20677 **SYNOPSIS**

```
20678 #include <sys/socket.h>
20679
20679 int bind(int socket, const struct sockaddr *address,
20680         socklen_t address_len);
```

20681 **DESCRIPTION**

20682 The `bind()` function shall assign a local socket address *address* to a socket identified by descriptor
 20683 *socket* that has no local socket address assigned. Sockets created with the `socket()` function are
 20684 initially unnamed; they are identified only by their address family.

20685 The `bind()` function takes the following arguments:

20686 <i>socket</i>	Specifies the file descriptor of the socket to be bound.
20687 <i>address</i>	Points to a sockaddr structure containing the address to be bound to the 20688 socket. The length and format of the address depend on the address family of 20689 the socket.
20690 <i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 20691 argument.

20692 The socket specified by *socket* may require the process to have appropriate privileges to use the
 20693 `bind()` function.

20694 If the socket address cannot be assigned immediately and `O_NONBLOCK` is set for the file +
 20695 descriptor for the socket, `bind()` shall fail and set *errno* to `[EINPROGRESS]`, but the assignment +
 20696 request shall not be aborted, and the assignment shall be completed asynchronously. +
 20697 Subsequent calls to `bind()` for the same socket, before the assignment is completed, shall fail and +
 20698 set *errno* to `[EALREADY]`. +

20699 When the assignment has been performed asynchronously, `pselect()`, `select()`, and `poll()` shall +
 20700 indicate that the file descriptor for the socket is ready for reading and writing.

20701 **RETURN VALUE**

20702 Upon successful completion, `bind()` shall return 0; otherwise, `-1` shall be returned and *errno* set
 20703 to indicate the error.

20704 **ERRORS**

20705 The `bind()` function shall fail if:

20706 <code>[EADDRINUSE]</code>	The specified address is already in use.
20707 <code>[EADDRNOTAVAIL]</code>	The specified address is not available from the local machine.
20708 <code>[EAFNOSUPPORT]</code>	The specified address is not a valid address for the address family of the 20709 specified socket.
20710 <code>[EBADF]</code>	The <i>socket</i> argument is not a valid file descriptor.
20711 <code>[EINVAL]</code>	The socket is already bound to an address, and the protocol does not support 20712 binding to a new address; or the socket has been shut down.
20713 <code>[ENOBUFS]</code>	Insufficient resources were available to complete the call.

- 20716 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 20717 [EOPNOTSUPP] The socket type of the specified socket does not support binding to an address.
- 20718 If the address family of the socket is AF_UNIX, then *bind()* shall fail if:
- 20719 [EACCES] A component of the path prefix denies search permission, or the requested
20720 name requires writing in a directory with a mode that denies write
20721 permission.
- 20722 [EDESTADDRREQ] or [EISDIR]
20723 The *address* argument is a null pointer.
- 20724 [EIO] An I/O error occurred.
- 20725 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
20726 in *address*.
- 20727 [ENAMETOOLONG]
20728 A component of a pathname exceeded {NAME_MAX} characters, or an entire
20729 pathname exceeded {PATH_MAX} characters.
- 20730 [ENOENT] A component of the pathname does not name an existing file or the pathname
20731 is an empty string.
- 20732 [ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory.
- 20733 [EROFS] The name would reside on a read-only file system.
- 20734 The *bind()* function may fail if:
- 20735 [EACCES] The specified address is protected and the current user does not have
20736 permission to bind to it.
- 20737 [EALREADY] An assignment request is already in progress for the specified socket. +
- 20738 [EINVAL] The *address_len* argument is not a valid length for the address family.
- 20739 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the assignment +
20740 cannot be immediately performed; the assignment shall be performed +
20741 asynchronously.
- 20742 [EISCONN] The socket is already connected.
- 20743 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
20744 resolution of the pathname in *address*.
- 20745 [ENAMETOOLONG]
20746 Pathname resolution of a symbolic link produced an intermediate result
20747 whose length exceeds {PATH_MAX}.

EXAMPLES

None.

APPLICATION USAGEAn application program can retrieve the assigned socket name with the *getsockname()* function.**RATIONALE**

None.

FUTURE DIRECTIONS

None.

bind()

20756

SEE ALSO

20757

connect(), *getsockname()*, *listen()*, *socket()*

20758

XBD <[sys/socket.h](#)>

20759

CHANGE HISTORY

20760

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

20761

Issue 7

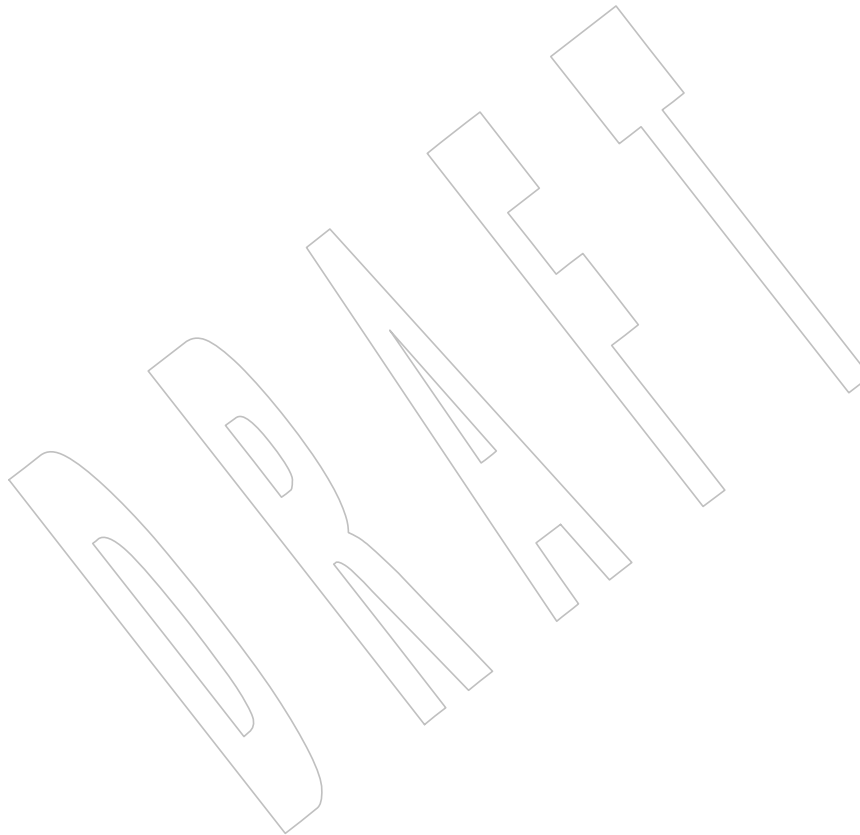
20762

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOBUFS] error to become a “shall fail” error.

20763

20764

SD5-XSH-ERN-185 is applied.



20765 **NAME**20766 `bsearch` — binary search a sorted table20767 **SYNOPSIS**20768 `#include <stdlib.h>`20769 `void *bsearch(const void *key, const void *base, size_t nel,`
20770 `size_t width, int (*compar)(const void *, const void *));`20771 **DESCRIPTION**20772 CX The functionality described on this reference page is aligned with the ISO C standard. Any
20773 conflict between the requirements described here and the ISO C standard is unintentional. This
20774 volume of POSIX.1-200x defers to the ISO C standard.20775 The `bsearch()` function shall search an array of `nel` objects, the initial element of which is pointed
20776 to by `base`, for an element that matches the object pointed to by `key`. The size of each element in
20777 the array is specified by `width`. If the `nel` argument has the value zero, the comparison function
20778 pointed to by `compar` shall not be called and no match shall be found.20779 The comparison function pointed to by `compar` shall be called with two arguments that point to
20780 the `key` object and to an array element, in that order.20781 The application shall ensure that the comparison function pointed to by `compar` does not alter the
20782 contents of the array. The implementation may reorder elements of the array between calls to the
20783 comparison function, but shall not alter the contents of any individual element.

20784 The implementation shall ensure that the first argument is always a pointer to the key.

20785 When the same objects (consisting of `width` bytes, irrespective of their current positions in the
20786 array) are passed more than once to the comparison function, the results shall be consistent with
20787 one another. That is, the same object shall always compare the same way with the key.20788 The application shall ensure that the function returns an integer less than, equal to, or greater
20789 than 0 if the `key` object is considered, respectively, to be less than, to match, or to be greater than
20790 the array element. The application shall ensure that the array consists of all the elements that
20791 compare less than, all the elements that compare equal to, and all the elements that compare
20792 greater than the `key` object, in that order.20793 **RETURN VALUE**20794 The `bsearch()` function shall return a pointer to a matching member of the array, or a null pointer
20795 if no match is found. If two or more members compare equal, which member is returned is
20796 unspecified.20797 **ERRORS**

20798 No errors are defined.

20799 **EXAMPLES**20800 The example below searches a table containing pointers to nodes consisting of a string and its
20801 length. The table is ordered alphabetically on the string in the node pointed to by each entry.20802 The code fragment below reads in strings and either finds the corresponding node and prints
20803 out the string and its length, or prints an error message.20804 `#include <stdio.h>`
20805 `#include <stdlib.h>`
20806 `#include <string.h>`
20807 `#define TABSIZE 1000`
20808 `struct node { /* These are stored in the table. */`

```

20809     char *string;
20810     int length;
20811 };
20812 struct node table[TABSIZE];    /* Table to be searched. */
20813     .
20814     .
20815     .
20816 {
20817     struct node *node_ptr, node;
20818     /* Routine to compare 2 nodes. */
20819     int node_compare(const void *, const void *);
20820     .
20821     .
20822     .
20823     while (scanf("%Ms", &node.string) != EOF) {
20824         node_ptr = (struct node *)bsearch((void *)&node,
20825             (void *)table, TABSIZE,
20826             sizeof(struct node), node_compare);
20827         if (node_ptr != NULL) {
20828             (void)printf("string = %20s, length = %d\n",
20829                 node_ptr->string, node_ptr->length);
20830         } else {
20831             (void)printf("not found: %s\n", node.string);
20832         }
20833         free(node.string);
20834     }
20835 }
20836 /*
20837     This routine compares two nodes based on an
20838     alphabetical ordering of the string field.
20839 */
20840 int
20841 node_compare(const void *node1, const void *node2)
20842 {
20843     return strcoll(((const struct node *)node1)->string,
20844         ((const struct node *)node2)->string);
20845 }

```

APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

RATIONALE

The requirement that the second argument (hereafter referred to as *p*) to the comparison function is a pointer to an element of the array implies that for every call all of the following expressions are non-zero:

```

20856 ((char *)p - (char *(base) % width == 0
20857 (char *)p >= (char *)base
20858 (char *)p < (char *)base + nel * width

```

20859

FUTURE DIRECTIONS

20860

None.

20861

SEE ALSO

20862

hcreate(), lsearch(), qsort(), tdelete()

20863

XBD <stdlib.h>

20864

CHANGE HISTORY

20865

First released in Issue 1. Derived from Issue 1 of the SVID.

20866

Issue 6

20867

The normative text is updated to avoid use of the term “must” for application requirements.

20868

20869

20870

20871

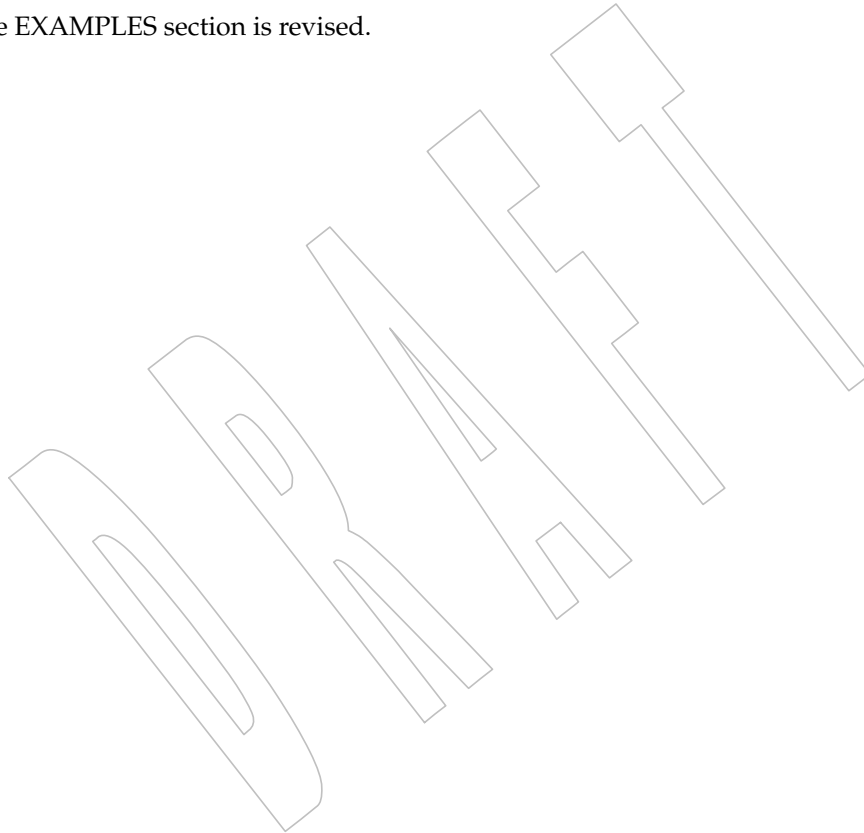
IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/11 is applied, adding to the DESCRIPTION the last sentence of the first non-shaded paragraph, and the following three paragraphs. The RATIONALE section is also updated. These changes are for alignment with the ISO C standard.

20872

Issue 7

20873

The EXAMPLES section is revised.



20874 **NAME**
 20875 `btowc` — single byte to wide character conversion

20876 **SYNOPSIS**
 20877 `#include <stdio.h>`
 20878 `#include <wchar.h>`
 20879 `wint_t btowc(int c);`

20880 **DESCRIPTION**
 20881 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20882 conflict between the requirements described here and the ISO C standard is unintentional. This
 20883 volume of POSIX.1-200x defers to the ISO C standard.

20884 The `btowc()` function shall determine whether `c` constitutes a valid (one-byte) character in the
 20885 initial shift state.

20886 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

20887 **RETURN VALUE**
 20888 The `btowc()` function shall return `WEOF` if `c` has the value `EOF` or if (**unsigned char**) `c` does not
 20889 constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the
 20890 wide-character representation of that character.

20891 **ERRORS**
 20892 No errors are defined.

20893 **EXAMPLES**
 20894 None.

20895 **APPLICATION USAGE**
 20896 None.

20897 **RATIONALE**
 20898 None.

20899 **FUTURE DIRECTIONS**
 20900 None.

20901 **SEE ALSO**
 20902 [wctob\(\)](#)
 20903 XBD [<wchar.h>](#)

20904 **CHANGE HISTORY**
 20905 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 20906 (E).

20907 **NAME**
 20908 cabs, cabsf, cabsl — return a complex absolute value

20909 **SYNOPSIS**
 20910 #include <complex.h>
 20911 double cabs(double complex z);
 20912 float cabsf(float complex z);
 20913 long double cabsl(long double complex z);

20914 **DESCRIPTION**
 20915 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20916 conflict between the requirements described here and the ISO C standard is unintentional. This
 20917 volume of POSIX.1-200x defers to the ISO C standard.

20918 These functions shall compute the complex absolute value (also called norm, modulus, or
 20919 magnitude) of z.

20920 **RETURN VALUE**
 20921 These functions shall return the complex absolute value.

20922 **ERRORS**
 20923 No errors are defined.

20924 **EXAMPLES**
 20925 None.

20926 **APPLICATION USAGE**
 20927 None.

20928 **RATIONALE**
 20929 None.

20930 **FUTURE DIRECTIONS**
 20931 None.

20932 **SEE ALSO**
 20933 XBD [<complex.h>](#)

20934 **CHANGE HISTORY**
 20935 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20936 **NAME**
 20937 `cacos`, `cacosf`, `cacosl` — complex arc cosine functions

20938 **SYNOPSIS**
 20939 `#include <complex.h>`
 20940 `double complex cacos(double complex z);`
 20941 `float complex cacosf(float complex z);`
 20942 `long double complex cacosl(long double complex z);`

20943 **DESCRIPTION**
 20944 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20945 conflict between the requirements described here and the ISO C standard is unintentional. This
 20946 volume of POSIX.1-200x defers to the ISO C standard.

20947 These functions shall compute the complex arc cosine of z , with branch cuts outside the interval
 20948 $[-1, +1]$ along the real axis.

20949 **RETURN VALUE**
 20950 These functions shall return the complex arc cosine value, in the range of a strip mathematically
 20951 unbounded along the imaginary axis and in the interval $[0, \pi]$ along the real axis.

20952 **ERRORS**
 20953 No errors are defined.

20954 **EXAMPLES**
 20955 None.

20956 **APPLICATION USAGE**
 20957 None.

20958 **RATIONALE**
 20959 None.

20960 **FUTURE DIRECTIONS**
 20961 None.

20962 **SEE ALSO**
 20963 [ccos\(\)](#)
 20964 XBD [<complex.h>](#)

20965 **CHANGE HISTORY**
 20966 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20967 **NAME**
 20968 cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

20969 **SYNOPSIS**
 20970 #include <complex.h>
 20971 double complex cacosh(double complex z);
 20972 float complex cacoshf(float complex z);
 20973 long double complex cacoshl(long double complex z);

20974 **DESCRIPTION**
 20975 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 20976 conflict between the requirements described here and the ISO C standard is unintentional. This
 20977 volume of POSIX.1-200x defers to the ISO C standard.

20978 These functions shall compute the complex arc hyperbolic cosine of z , with a branch cut at
 20979 values less than 1 along the real axis.

20980 **RETURN VALUE**
 20981 These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip
 20982 of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

20983 **ERRORS**
 20984 No errors are defined.

20985 **EXAMPLES**
 20986 None.

20987 **APPLICATION USAGE**
 20988 None.

20989 **RATIONALE**
 20990 None.

20991 **FUTURE DIRECTIONS**
 20992 None.

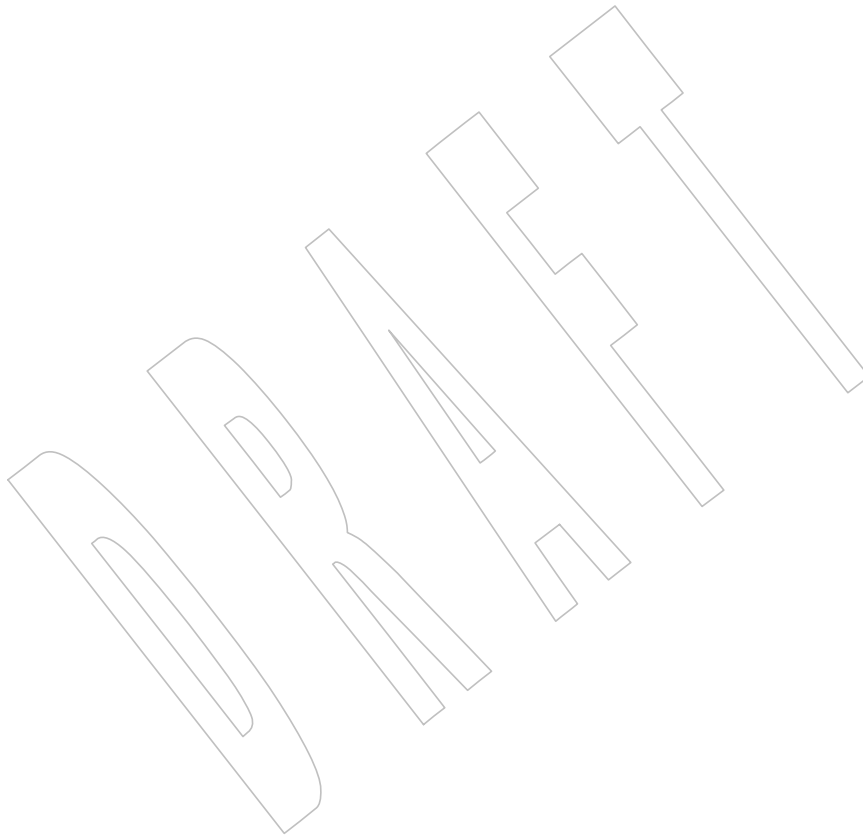
20993 **SEE ALSO**
 20994 [ccosh\(\)](#)
 20995 XBD [<complex.h>](#)

20996 **CHANGE HISTORY**
 20997 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

20998 **NAME**
20999 `cacosl` — complex arc cosine functions

21000 **SYNOPSIS**
21001 `#include <complex.h>`
21002 `long double complex cacosl(long double complex z);`

21003 **DESCRIPTION**
21004 Refer to *cacos()*.



21005 **NAME**

21006 calloc — a memory allocator

21007 **SYNOPSIS**

21008 #include <stdlib.h>

21009 void *calloc(size_t *nelem*, size_t *elsize*);21010 **DESCRIPTION**21011 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21012 conflict between the requirements described here and the ISO C standard is unintentional. This
21013 volume of POSIX.1-200x defers to the ISO C standard.21014 The *calloc()* function shall allocate unused space for an array of *nelem* elements each of whose
21015 size in bytes is *elsize*. The space shall be initialized to all bits 0.21016 The order and contiguity of storage allocated by successive calls to *calloc()* is unspecified. The
21017 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
21018 a pointer to any type of object and then used to access such an object or an array of such objects
21019 in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall
21020 yield a pointer to an object disjoint from any other object. The pointer returned shall point to the
21021 start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer
21022 shall be returned. If the size of the space requested is 0, the behavior is implementation-defined:
21023 the value returned shall be either a null pointer or a unique pointer.21024 **RETURN VALUE**21025 Upon successful completion with both *nelem* and *elsize* non-zero, *calloc()* shall return a pointer to
21026 the allocated space. If either *nelem* or *elsize* is 0, then either a null pointer or a unique pointer
21027 value that can be successfully passed to *free()* shall be returned. Otherwise, it shall return a null
21028 pointer and set *errno* to indicate the error.21029 **ERRORS**21030 The *calloc()* function shall fail if:

21031 CX [ENOMEM] Insufficient memory is available.

21032 **EXAMPLES**

21033 None.

21034 **APPLICATION USAGE**

21035 There is now no requirement for the implementation to support the inclusion of <malloc.h>.

21036 **RATIONALE**

21037 None.

21038 **FUTURE DIRECTIONS**

21039 None.

21040 **SEE ALSO**21041 *free()*, *malloc()*, *realloc()*

21042 XBD <stdlib.h>

21043 **CHANGE HISTORY**

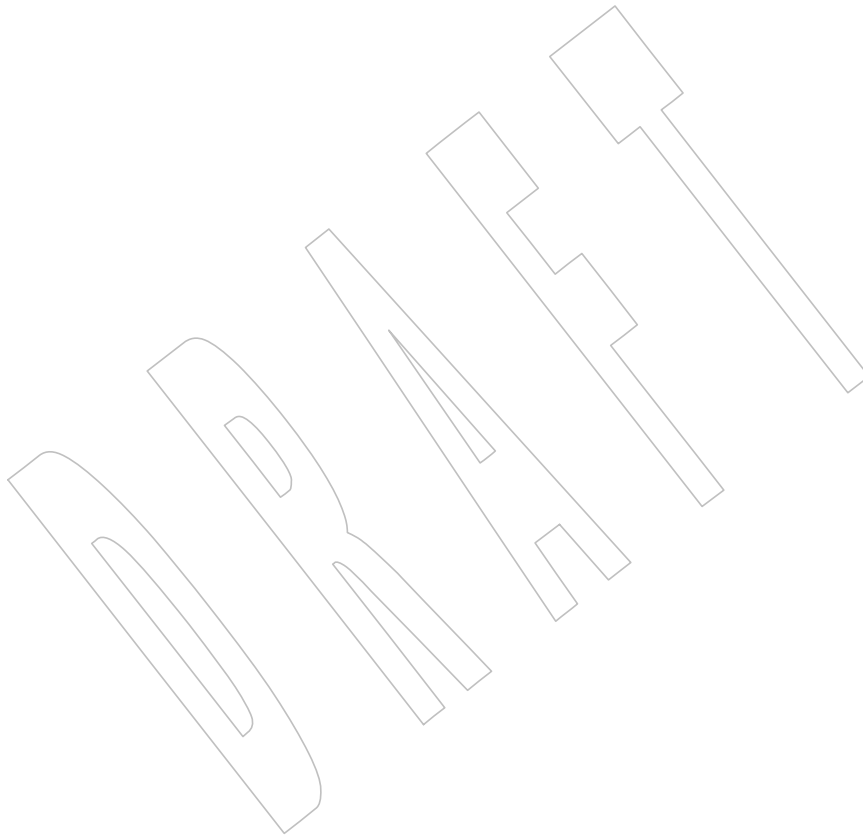
21044 First released in Issue 1. Derived from Issue 1 of the SVID.

21045
21046
21047
21048
21049
21050**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of *errno* and the [ENOMEM] error condition are mandatory if an insufficient memory condition occurs.



21051 **NAME**
 21052 `carg, cargf, cargl` — complex argument functions

21053 **SYNOPSIS**
 21054 `#include <complex.h>`
 21055 `double carg(double complex z);`
 21056 `float cargf(float complex z);`
 21057 `long double cargl(long double complex z);`

21058 **DESCRIPTION**
 21059 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21060 conflict between the requirements described here and the ISO C standard is unintentional. This
 21061 volume of POSIX.1-200x defers to the ISO C standard.

21062 These functions shall compute the argument (also called phase angle) of z , with a branch cut
 21063 along the negative real axis.

21064 **RETURN VALUE**
 21065 These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.

21066 **ERRORS**
 21067 No errors are defined.

21068 **EXAMPLES**
 21069 None.

21070 **APPLICATION USAGE**
 21071 None.

21072 **RATIONALE**
 21073 None.

21074 **FUTURE DIRECTIONS**
 21075 None.

21076 **SEE ALSO**
 21077 *cimag(), conj(), cproj()*

21078 XBD `<complex.h>`

21079 **CHANGE HISTORY**
 21080 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21081 **NAME**
 21082 `casin, casinf, casinl` — complex arc sine functions

21083 **SYNOPSIS**
 21084 `#include <complex.h>`
 21085 `double complex casin(double complex z);`
 21086 `float complex casinf(float complex z);`
 21087 `long double complex casinl(long double complex z);`

21088 **DESCRIPTION**
 21089 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21090 conflict between the requirements described here and the ISO C standard is unintentional. This
 21091 volume of POSIX.1-200x defers to the ISO C standard.

21092 These functions shall compute the complex arc sine of z , with branch cuts outside the interval
 21093 $[-1, +1]$ along the real axis.

21094 **RETURN VALUE**
 21095 These functions shall return the complex arc sine value, in the range of a strip mathematically
 21096 unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the real axis.

21097 **ERRORS**
 21098 No errors are defined.

21099 **EXAMPLES**
 21100 None.

21101 **APPLICATION USAGE**
 21102 None.

21103 **RATIONALE**
 21104 None.

21105 **FUTURE DIRECTIONS**
 21106 None.

21107 **SEE ALSO**
 21108 [csin\(\)](#)
 21109 XBD [<complex.h>](#)

21110 **CHANGE HISTORY**
 21111 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21112 **NAME**

21113 casinh, casinhf, casinhl — complex arc hyperbolic sine functions

21114 **SYNOPSIS**

21115 #include <complex.h>

21116 double complex casinh(double complex z);

21117 float complex casinhf(float complex z);

21118 long double complex casinhl(long double complex z);

21119 **DESCRIPTION**

21120 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21121 conflict between the requirements described here and the ISO C standard is unintentional. This
 21122 volume of POSIX.1-200x defers to the ISO C standard.

21123 These functions shall compute the complex arc hyperbolic sine of z , with branch cuts outside the
 21124 interval $[-i, +i]$ along the imaginary axis.

21125 **RETURN VALUE**

21126 These functions shall return the complex arc hyperbolic sine value, in the range of a strip
 21127 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
 21128 imaginary axis.

21129 **ERRORS**

21130 No errors are defined.

21131 **EXAMPLES**

21132 None.

21133 **APPLICATION USAGE**

21134 None.

21135 **RATIONALE**

21136 None.

21137 **FUTURE DIRECTIONS**

21138 None.

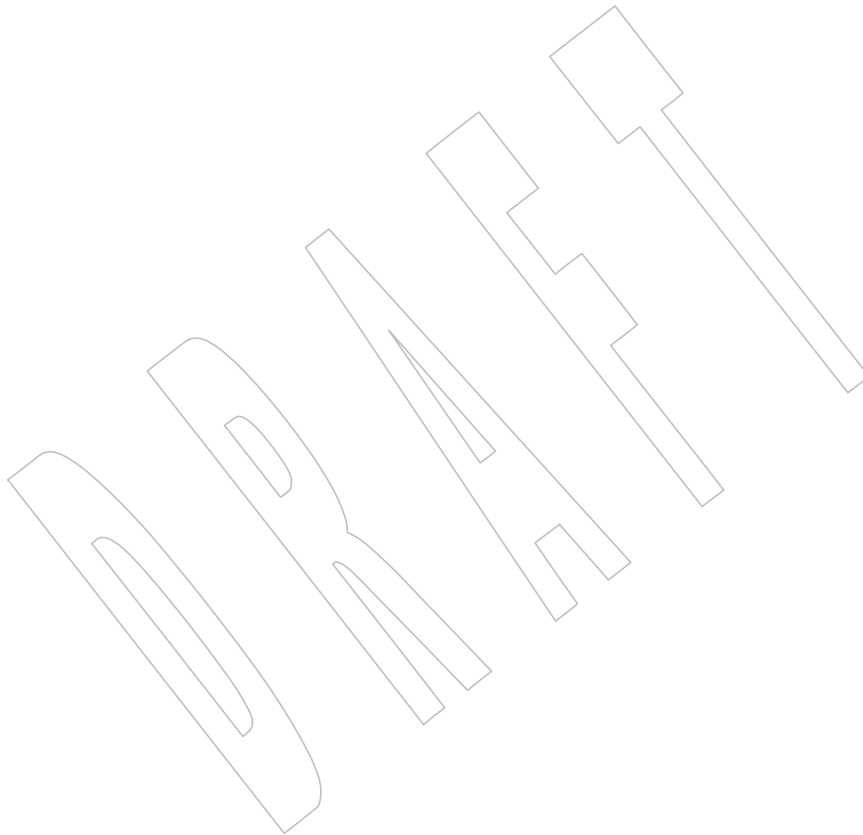
21139 **SEE ALSO**21140 [csinh\(\)](#)21141 XBD [<complex.h>](#)21142 **CHANGE HISTORY**

21143 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21144 **NAME**
21145 **casinl** — complex arc sine functions

21146 **SYNOPSIS**
21147 `#include <complex.h>`
21148 `long double complex casinl(long double complex z);`

21149 **DESCRIPTION**
21150 Refer to *casin()*.



21151 **NAME**
 21152 catan, catanf, catanl — complex arc tangent functions

21153 **SYNOPSIS**

21154 #include <complex.h>
 21155 double complex catan(double complex z);
 21156 float complex catanf(float complex z);
 21157 long double complex catanl(long double complex z);

21158 **DESCRIPTION**

21159 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21160 conflict between the requirements described here and the ISO C standard is unintentional. This
 21161 volume of POSIX.1-200x defers to the ISO C standard.

21162 These functions shall compute the complex arc tangent of z , with branch cuts outside the
 21163 interval $[-i, +i]$ along the imaginary axis.

21164 **RETURN VALUE**

21165 These functions shall return the complex arc tangent value, in the range of a strip
 21166 mathematically unbounded along the imaginary axis and in the interval $[-\pi/2, +\pi/2]$ along the
 21167 real axis.

21168 **ERRORS**

21169 No errors are defined.

21170 **EXAMPLES**

21171 None.

21172 **APPLICATION USAGE**

21173 None.

21174 **RATIONALE**

21175 None.

21176 **FUTURE DIRECTIONS**

21177 None.

21178 **SEE ALSO**

21179 [ctan\(\)](#)

21180 XBD [<complex.h>](#)

21181 **CHANGE HISTORY**

21182 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21183 **NAME**
 21184 `catanh, catanhf, catanhl` — complex arc hyperbolic tangent functions

21185 **SYNOPSIS**
 21186 `#include <complex.h>`
 21187 `double complex catanh(double complex z);`
 21188 `float complex catanhf(float complex z);`
 21189 `long double complex catanhl(long double complex z);`

21190 **DESCRIPTION**
 21191 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21192 conflict between the requirements described here and the ISO C standard is unintentional. This
 21193 volume of POSIX.1-200x defers to the ISO C standard.

21194 These functions shall compute the complex arc hyperbolic tangent of z , with branch cuts outside
 21195 the interval $[-1, +1]$ along the real axis.

21196 **RETURN VALUE**
 21197 These functions shall return the complex arc hyperbolic tangent value, in the range of a strip
 21198 mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the
 21199 imaginary axis.

21200 **ERRORS**
 21201 No errors are defined.

21202 **EXAMPLES**
 21203 None.

21204 **APPLICATION USAGE**
 21205 None.

21206 **RATIONALE**
 21207 None.

21208 **FUTURE DIRECTIONS**
 21209 None.

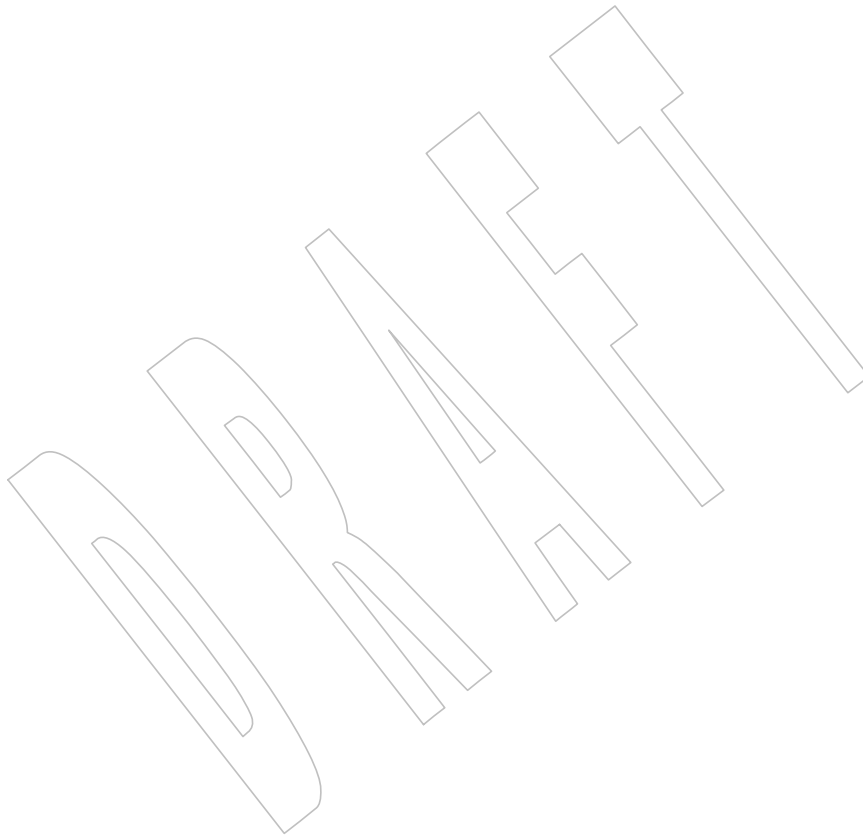
21210 **SEE ALSO**
 21211 [*ctanh\(\)*](#)
 21212 XBD [<complex.h>](#)

21213 **CHANGE HISTORY**
 21214 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

21215 **NAME**
21216 catanl — complex arc tangent functions

21217 **SYNOPSIS**
21218 #include <complex.h>
21219 long double complex catanl(long double complex z);

21220 **DESCRIPTION**
21221 Refer to *catan()*.



21222 **NAME**21223 `catclose` — close a message catalog descriptor21224 **SYNOPSIS**21225 `#include <nl_types.h>`21226 `int catclose(nl_catd catd);`21227 **DESCRIPTION**21228 The `catclose()` function shall close the message catalog identified by `catd`. If a file descriptor is
21229 used to implement the type **nl_catd**, that file descriptor shall be closed.21230 **RETURN VALUE**21231 Upon successful completion, `catclose()` shall return 0; otherwise, -1 shall be returned, and `errno`
21232 set to indicate the error.21233 **ERRORS**21234 The `catclose()` function may fail if:

21235 [EBADF] The catalog descriptor is not valid.

21236 [EINTR] The `catclose()` function was interrupted by a signal.21237 **EXAMPLES**

21238 None.

21239 **APPLICATION USAGE**

21240 None.

21241 **RATIONALE**

21242 None.

21243 **FUTURE DIRECTIONS**

21244 None.

21245 **SEE ALSO**21246 [catgets\(\)](#), [catopen\(\)](#)21247 XBD [<nl_types.h>](#)21248 **CHANGE HISTORY**

21249 First released in Issue 2.

21250 **Issue 7**21251 The `catclose()` function is moved from the XSI option to the Base.

21252 **NAME**21253 `catgets` — read a program message21254 **SYNOPSIS**21255 `#include <nl_types.h>`21256 `char *catgets(nl_catd catd, int set_id, int msg_id, const char *s);`21257 **DESCRIPTION**

21258 The `catgets()` function shall attempt to read message `msg_id`, in set `set_id`, from the message
 21259 catalog identified by `catd`. The `catd` argument is a message catalog descriptor returned from an
 21260 earlier call to `catopen()`. The results are undefined if `catd` is not a value returned by `catopen()` for
 21261 a message catalog still open in the process. The `s` argument points to a default message string
 21262 which shall be returned by `catgets()` if it cannot retrieve the identified message.

21263 The `catgets()` function need not be thread-safe. A function that is not required to be thread-safe is
 21264 not required to be reentrant.

21265 **RETURN VALUE**

21266 If the identified message is retrieved successfully, `catgets()` shall return a pointer to an internal
 21267 buffer area containing the null-terminated message string. If the call is unsuccessful for any
 21268 reason, `s` shall be returned and `errno` shall be set to indicate the error.

21269 **ERRORS**21270 The `catgets()` function shall fail if:

21271 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 21272 was transferred.

21273 [ENOMSG] The message identified by `set_id` and `msg_id` is not in the message catalog.

21274 The `catgets()` function may fail if:

21275 [EBADF] The `catd` argument is not a valid message catalog descriptor open for reading.

21276 [EBADMSG] The message identified by `set_id` and `msg_id` in the specified message catalog
 21277 did not satisfy implementation-defined security criteria.

21278 [EINVAL] The message catalog identified by `catd` is corrupted.

21279 **EXAMPLES**

21280 None.

21281 **APPLICATION USAGE**

21282 None.

21283 **RATIONALE**

21284 None.

21285 **FUTURE DIRECTIONS**

21286 None.

21287 **SEE ALSO**21288 `catclose()`, `catopen()`21289 XBD `<nl_types.h>`

21290
21291

21292
21293

21294
21295

21296
21297
21298
21299
21300
21301**CHANGE HISTORY**

First released in Issue 2.

Issue 5

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

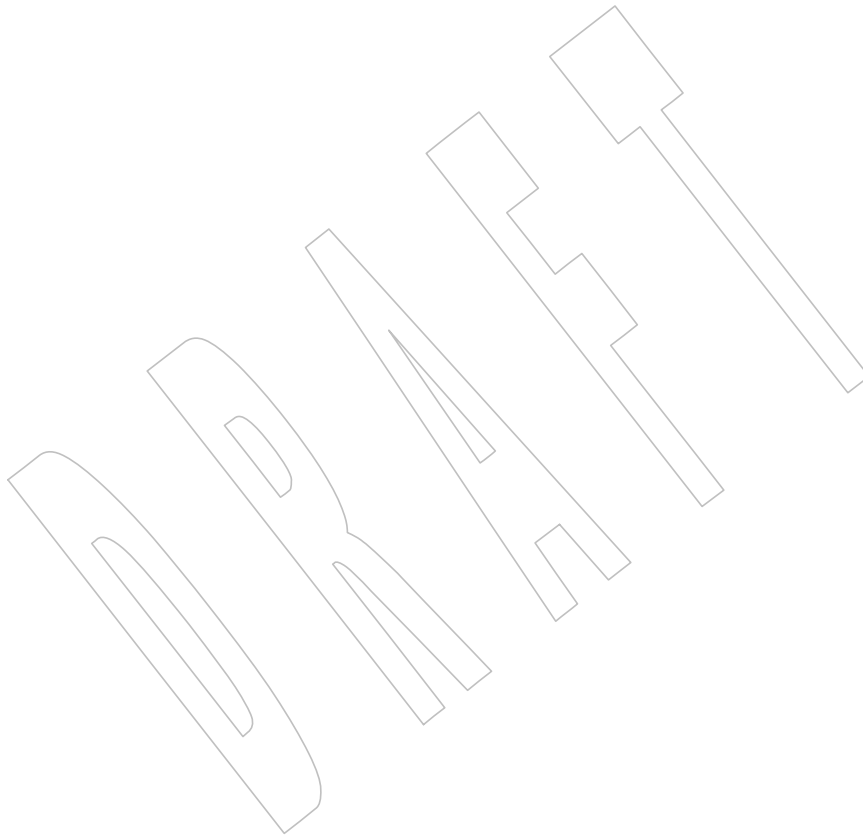
Issue 6

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [EINTR] and [ENOMSG] errors to become “shall fail” errors, updating the RETURN VALUE section, and updating the DESCRIPTION to note that: “The results are undefined if *catd* is not a value returned by *catopen()* for a message catalog still open in the process.

The *catgets()* function is moved from the XSI option to the Base.



21302 **NAME**

21303 catopen — open a message catalog

21304 **SYNOPSIS**

21305 #include <nl_types.h>

21306 nl_catd catopen(const char *name, int oflag);

21307 **DESCRIPTION**

21308 The *catopen()* function shall open a message catalog and return a message catalog descriptor.
 21309 The *name* argument specifies the name of the message catalog to be opened. If *name* contains a
 21310 ' / ', then *name* specifies a complete name for the message catalog. Otherwise, the environment
 21311 variable *NLSPATH* is used with *name* substituted for the %N conversion specification (see XBD
 21312 Chapter 8, on page 159). If *NLSPATH* exists in the environment when the process starts, then if
 21313 the process has appropriate privileges, the behavior of *catopen()* is undefined. If *NLSPATH* does
 21314 not exist in the environment, or if a message catalog cannot be found in any of the components
 21315 specified by *NLSPATH*, then an implementation-defined default path shall be used. This default
 21316 may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is *NL_CAT_LOCALE*, or
 21317 the *LANG* environment variable if *oflag* is 0.

21318 A message catalog descriptor shall remain valid in a process until that process closes it, or a
 21319 successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES*
 21320 category may invalidate existing open catalogs.

21321 If a file descriptor is used to implement message catalog descriptors, the *FD_CLOEXEC* flag
 21322 shall be set; see <fcntl.h>.

21323 If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the
 21324 catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is
 21325 *NL_CAT_LOCALE*, the *LC_MESSAGES* category is used to locate the message catalog (see XBD
 21326 Section 8.2, on page 160).

21327 **RETURN VALUE**

21328 Upon successful completion, *catopen()* shall return a message catalog descriptor for use on
 21329 subsequent calls to *catgets()* and *catclose()*. Otherwise, *catopen()* shall return (*nl_catd*) -1 and set
 21330 *errno* to indicate the error.

21331 **ERRORS**21332 The *catopen()* function may fail if:

21333 [EACCES] Search permission is denied for the component of the path prefix of the
 21334 message catalog or read permission is denied for the message catalog.

21335 [EMFILE] All file descriptors available to the process are currently open.

21336 [ENAMETOOLONG]

21337 The length of a pathname of the message catalog exceeds {PATH_MAX} or a
 21338 pathname component is longer than {NAME_MAX}.

21339 [ENAMETOOLONG]

21340 Pathname resolution of a symbolic link produced an intermediate result
 21341 whose length exceeds {PATH_MAX}.

21342 [ENFILE] Too many files are currently open in the system.

21343 [ENOENT] The message catalog does not exist or the *name* argument points to an empty
 21344 string.

catopen()

21345 [ENOMEM] Insufficient storage space is available.

21346 [ENOTDIR] A component of the path prefix of the message catalog is not a directory.

EXAMPLES

21347 None.

21348

APPLICATION USAGE

21349 Some implementations of *catopen()* use *malloc()* to allocate space for internal buffer areas. The
21350 *catopen()* function may fail if there is insufficient storage space available to accommodate these
21351 buffers.
21352

21353 Conforming applications must assume that message catalog descriptors are not valid after a call
21354 to one of the *exec* functions.

21355 Application developers should be aware that guidelines for the location of message catalogs
21356 have not yet been developed. Therefore they should take care to avoid conflicting with catalogs
21357 used by other applications and the standard utilities.

RATIONALE

21358 None.

21359

FUTURE DIRECTIONS

21360 None.

21361

SEE ALSO

21362 *catclose()*, *catgets()*

21363

21364 XBD Chapter 8 (on page 159), [<fcntl.h>](#), [<nl_types.h>](#),

CHANGE HISTORY

21365 First released in Issue 2.

21366

Issue 7

21367 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

21368

21369 The *catopen()* function is moved from the XSI option to the Base.

21370 **NAME**
 21371 `cbrt`, `cbrtf`, `cbrtl` — cube root functions

21372 **SYNOPSIS**
 21373 `#include <math.h>`
 21374 `double cbrt(double x);`
 21375 `float cbrtf(float x);`
 21376 `long double cbrtl(long double x);`

21377 **DESCRIPTION**
 21378 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21379 conflict between the requirements described here and the ISO C standard is unintentional. This
 21380 volume of POSIX.1-200x defers to the ISO C standard.

21381 These functions shall compute the real cube root of their argument x .

21382 **RETURN VALUE**
 21383 Upon successful completion, these functions shall return the cube root of x .

21384 MX If x is NaN, a NaN shall be returned.
 21385 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

21386 **ERRORS**
 21387 No errors are defined.

21388 **EXAMPLES**
 21389 None.

21390 **APPLICATION USAGE**
 21391 None.

21392 **RATIONALE**
 21393 For some applications, a true cube root function, which returns negative results for negative
 21394 arguments, is more appropriate than $\text{pow}(x, 1.0/3.0)$, which returns a NaN for x less than 0.

21395 **FUTURE DIRECTIONS**
 21396 None.

21397 **SEE ALSO**
 21398 XBD [<math.h>](#)

21399 **CHANGE HISTORY**
 21400 First released in Issue 4, Version 2.

21401 **Issue 5**
 21402 Moved from X/OPEN UNIX extension to BASE.

21403 **Issue 6**
 21404 The `cbrt()` function is no longer marked as an extension.
 21405 The `cbrtf()` and `cbrtl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.
 21406 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 21407 revised to align with the ISO/IEC 9899:1999 standard.
 21408 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 21409 marked.

21410 **NAME**

21411 `ccos, ccosf, ccosl` — complex cosine functions

21412 **SYNOPSIS**

21413 `#include <complex.h>`

21414 `double complex ccos(double complex z);`

21415 `float complex ccosf(float complex z);`

21416 `long double complex ccosl(long double complex z);`

21417 **DESCRIPTION**

21418 CX The functionality described on this reference page is aligned with the ISO C standard. Any

21419 conflict between the requirements described here and the ISO C standard is unintentional. This

21420 volume of POSIX.1-200x defers to the ISO C standard.

21421 These functions shall compute the complex cosine of *z*.

21422 **RETURN VALUE**

21423 These functions shall return the complex cosine value.

21424 **ERRORS**

21425 No errors are defined.

21426 **EXAMPLES**

21427 None.

21428 **APPLICATION USAGE**

21429 None.

21430 **RATIONALE**

21431 None.

21432 **FUTURE DIRECTIONS**

21433 None.

21434 **SEE ALSO**

21435 [*ccos\(\)*](#)

21436 XBD [`<complex.h>`](#)

21437 **CHANGE HISTORY**

21438 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

21439 **NAME**
 21440 ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

21441 **SYNOPSIS**

21442 #include <complex.h>
 21443 double complex ccosh(double complex z);
 21444 float complex ccoshf(float complex z);
 21445 long double complex ccoshl(long double complex z);

21446 **DESCRIPTION**

21447 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 21448 conflict between the requirements described here and the ISO C standard is unintentional. This
 21449 volume of POSIX.1-200x defers to the ISO C standard.

21450 These functions shall compute the complex hyperbolic cosine of *z*.

21451 **RETURN VALUE**

21452 These functions shall return the complex hyperbolic cosine value.

21453 **ERRORS**

21454 No errors are defined.

21455 **EXAMPLES**

21456 None.

21457 **APPLICATION USAGE**

21458 None.

21459 **RATIONALE**

21460 None.

21461 **FUTURE DIRECTIONS**

21462 None.

21463 **SEE ALSO**

21464 *cacosh()*

21465 XBD <complex.h>

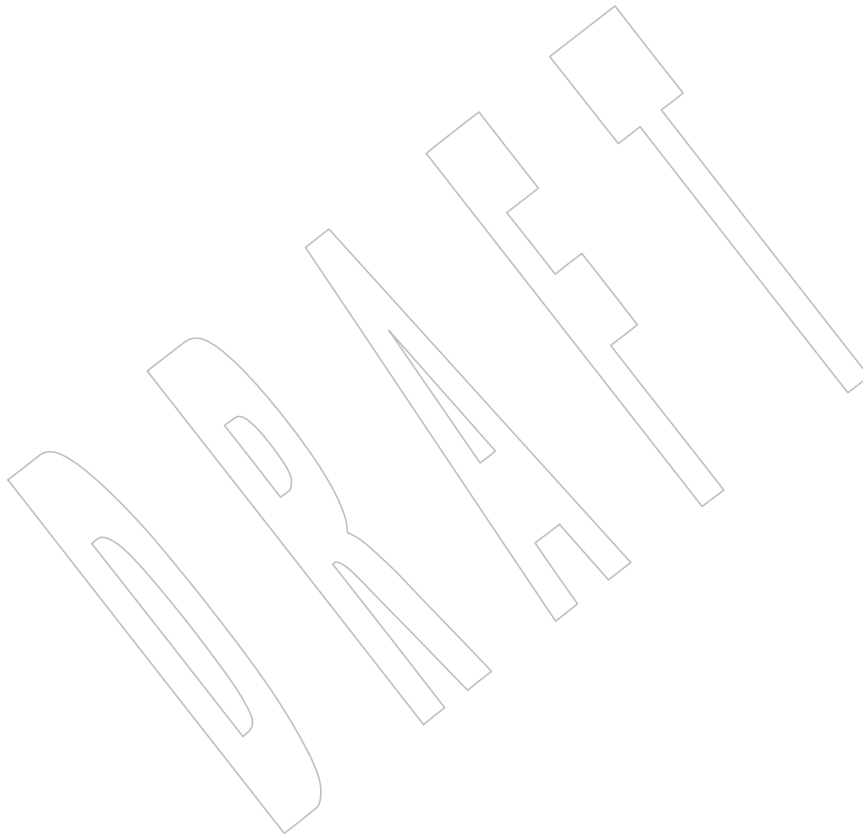
21466 **CHANGE HISTORY**

21467 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21468 **NAME**
21469 ccosl — complex cosine functions

21470 **SYNOPSIS**
21471 #include <complex.h>
21472 long double complex ccosl(long double complex z);

21473 **DESCRIPTION**
21474 Refer to *ccos()*.



21475 **NAME**

21476 ceil, ceilf, ceill — ceiling value function

21477 **SYNOPSIS**

```
21478       #include <math.h>
21479
21479       double ceil(double x);
21480       float ceilf(float x);
21481       long double ceill(long double x);
```

21482 **DESCRIPTION**

21483 CX The functionality described on this reference page is aligned with the ISO C standard. Any
21484 conflict between the requirements described here and the ISO C standard is unintentional. This
21485 volume of POSIX.1-200x defers to the ISO C standard.

21486 These functions shall compute the smallest integral value not less than x .

21487 An application wishing to check for error situations should set *errno* to zero and call
21488 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
21489 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
21490 zero, an error has occurred.

21491 **RETURN VALUE**

21492 Upon successful completion, *ceil()*, *ceilf()*, and *ceill()* shall return the smallest integral value not
21493 less than x , expressed as a type **double**, **float**, or **long double**, respectively.

21494 MX If x is NaN, a NaN shall be returned.

21495 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

21496 XSI If the correct value would cause overflow, a range error shall occur and *ceil()*, *ceilf()*, and *ceill()*
21497 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

21498 **ERRORS**

21499 These functions shall fail if:

21500 XSI **Range Error** The result overflows.

21501 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
21502 then *errno* shall be set to [ERANGE]. If the integer expression
21503 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
21504 floating-point exception shall be raised.

21505 **EXAMPLES**

21506 None.

21507 **APPLICATION USAGE**

21508 The integral value returned by these functions need not be expressible as an **int** or **long**. The
21509 return value should be tested before assigning it to an integer type to avoid the undefined
21510 results of an integer overflow.

21511 The *ceil()* function can only overflow when the floating-point representation has
21512 DBL_MANT_DIG > DBL_MAX_EXP.

21513 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
21514 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

21515
21516

21517
21518

21519
21520
21521

21522
21523

21524
21525
21526

21527
21528

21529
21530

21531
21532

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *floor()*, *isnan()*

XBD [Section 4.19](#) (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ceilf()* and *ceilll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

21533

NAME

21534

cexp, cexpf, cexpl — complex exponential functions

21535

SYNOPSIS

21536

#include <complex.h>

21537

double complex cexp(double complex z);

21538

float complex cexpf(float complex z);

21539

long double complex cexpl(long double complex z);

21540

DESCRIPTION

21541

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

21542

21543

21544

These functions shall compute the complex exponent of z , defined as e^z .

21545

RETURN VALUE

21546

These functions shall return the complex exponential value of z .

21547

ERRORS

21548

No errors are defined.

21549

EXAMPLES

21550

None.

21551

APPLICATION USAGE

21552

None.

21553

RATIONALE

21554

None.

21555

FUTURE DIRECTIONS

21556

None.

21557

SEE ALSO

21558

clog()

21559

XBD <complex.h>

21560

CHANGE HISTORY

21561

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

21562 **NAME**

21563 cfgetispeed — get input baud rate

21564 **SYNOPSIS**

21565 #include <termios.h>

21566 speed_t cfgetispeed(const struct termios *termios_p);

21567 **DESCRIPTION**21568 The *cfgetispeed()* function shall extract the input baud rate from the **termios** structure to which
21569 the *termios_p* argument points.21570 This function shall return exactly the value in the **termios** data structure, without interpretation.21571 **RETURN VALUE**21572 Upon successful completion, *cfgetispeed()* shall return a value of type **speed_t** representing the
21573 input baud rate.21574 **ERRORS**

21575 No errors are defined.

21576 **EXAMPLES**

21577 None.

21578 **APPLICATION USAGE**

21579 None.

21580 **RATIONALE**21581 The term “baud” is used historically here, but is not technically correct. This is properly “bits per
21582 second”, which may not be the same as baud. However, the term is used because of the
21583 historical usage and understanding.21584 The *cfgetospeed()*, *cfgetispeed()*, *cfsetospeed()*, and *cfsetispeed()* functions do not take arguments as
21585 numbers, but rather as symbolic names. There are two reasons for this:

- 21586 1. Historically, numbers were not used because of the way the rate was stored in the data
-
- 21587 structure. This is retained even though a function is now used.
-
- 21588 2. More importantly, only a limited set of possible rates is at all portable, and this constrains
-
- 21589 the application to that set.

21590 There is nothing to prevent an implementation accepting as an extension a number (such as 126),
21591 and since the encoding of the Bxxx symbols is not specified, this can be done to avoid
21592 introducing ambiguity.21593 Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications
21594 in this volume of POSIX.1-200x have made it possible to determine whether split rates are
21595 supported and to support them without having to treat zero as a special case. Since this
21596 functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is
21597 the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-200x does not
21598 preclude B0 from being defined as the value 0; in fact, implementations would likely benefit
21599 from the two being equivalent. This volume of POSIX.1-200x does not fully specify whether the
21600 previous *cfsetispeed()* value is retained after a *tcgetattr()* as the actual value or as zero. Therefore,
21601 conforming applications should always set both the input speed and output speed when setting
21602 either.21603 In historical implementations, the baud rate information is traditionally kept in **c_cflag**.
21604 Applications should be written to presume that this might be the case (and thus not blindly copy
21605 **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag**

21606 field absolutely after setting a baud rate is a non-portable action because of this. In general, the
21607 unused parts of the flag fields might be used by the implementation and should not be blindly
21608 copied from the descriptions of one terminal device to another.

FUTURE DIRECTIONS

21609 None.

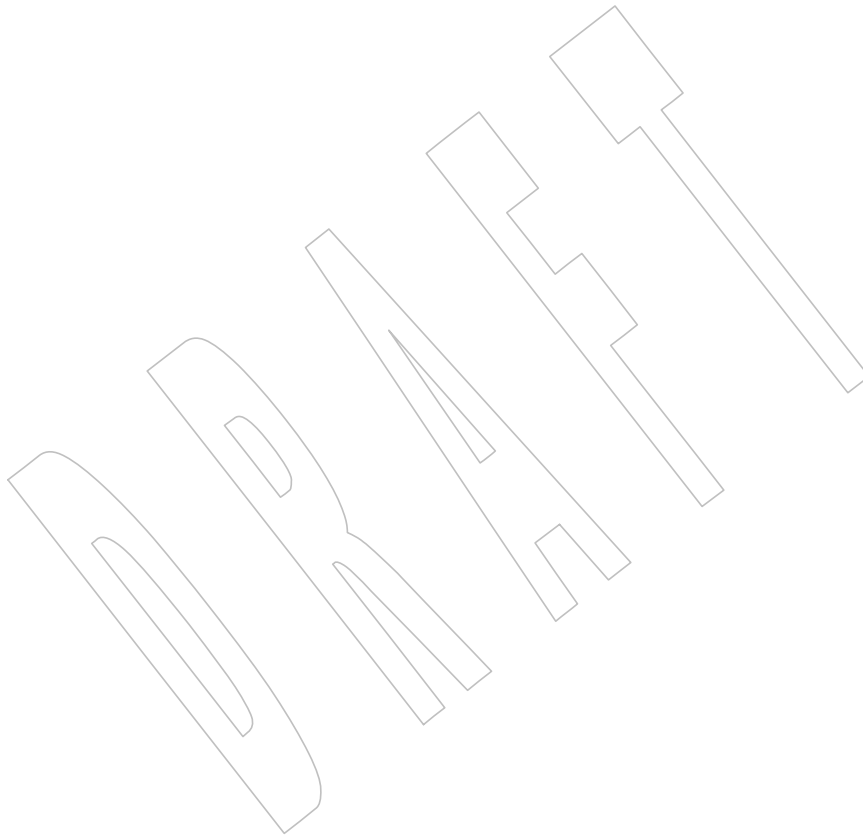
SEE ALSO

21611 *cfgetospeed()*, *cfsetispeed()*, *cfsetospeed()*, *tcgetattr()*

21613 XBD Chapter 11 (on page 185), **<termios.h>**

CHANGE HISTORY

21614 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.
21615



21616 **NAME**21617 `cfgetospeed` — get output baud rate21618 **SYNOPSIS**21619 `#include <termios.h>`21620 `speed_t cfgetospeed(const struct termios *termios_p);`21621 **DESCRIPTION**21622 The `cfgetospeed()` function shall extract the output baud rate from the **termios** structure to which
21623 the `termios_p` argument points.21624 This function shall return exactly the value in the **termios** data structure, without interpretation.21625 **RETURN VALUE**21626 Upon successful completion, `cfgetospeed()` shall return a value of type **speed_t** representing the
21627 output baud rate.21628 **ERRORS**

21629 No errors are defined.

21630 **EXAMPLES**

21631 None.

21632 **APPLICATION USAGE**

21633 None.

21634 **RATIONALE**21635 Refer to `cfgetispeed()`.21636 **FUTURE DIRECTIONS**

21637 None.

21638 **SEE ALSO**21639 `cfgetispeed()`, `cfsetispeed()`, `cfsetospeed()`, `tcgetattr()`21640 XBD Chapter 11 (on page 185), `<termios.h>`21641 **CHANGE HISTORY**

21642 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

21643 **NAME**

21644 cfsetispeed — set input baud rate

21645 **SYNOPSIS**

21646 #include <termios.h>

21647 int cfsetispeed(struct termios *termios_p, speed_t speed);

21648 **DESCRIPTION**21649 The *cfsetispeed()* function shall set the input baud rate stored in the structure pointed to by
21650 *termios_p* to *speed*.21651 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
21652 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
21653 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
21654 function is called.21655 **RETURN VALUE**21656 Upon successful completion, *cfsetispeed()* shall return 0; otherwise, -1 shall be returned, and
21657 *errno* may be set to indicate the error.21658 **ERRORS**21659 The *cfsetispeed()* function may fail if:21660 [EINVAL] The *speed* value is not a valid baud rate.21661 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
21662 <termios.h>.21663 **EXAMPLES**

21664 None.

21665 **APPLICATION USAGE**

21666 None.

21667 **RATIONALE**21668 Refer to *cfgetispeed()*.21669 **FUTURE DIRECTIONS**

21670 None.

21671 **SEE ALSO**21672 *cfgetispeed()*, *cfgetospeed()*, *cfsetospeed()*, *tcsetattr()*

21673 XBD Chapter 11 (on page 185), <termios.h>

21674 **CHANGE HISTORY**

21675 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

21676 **Issue 6**21677 The following new requirements on POSIX implementations derive from alignment with the
21678 Single UNIX Specification:

- 21679
- The optional setting of *errno* and the [EINVAL] error conditions are added.

21680 **NAME**

21681 cfsetospeed — set output baud rate

21682 **SYNOPSIS**

21683 #include <termios.h>

21684 int cfsetospeed(struct termios *termios_p, speed_t speed);

21685 **DESCRIPTION**21686 The *cfsetospeed()* function shall set the output baud rate stored in the structure pointed to by
21687 *termios_p* to *speed*.21688 There shall be no effect on the baud rates set in the hardware until a subsequent successful call
21689 to *tcsetattr()* with the same **termios** structure. Similarly, errors resulting from attempts to set
21690 baud rates not supported by the terminal device need not be detected until the *tcsetattr()*
21691 function is called.21692 **RETURN VALUE**21693 Upon successful completion, *cfsetospeed()* shall return 0; otherwise, it shall return -1 and *errno*
21694 may be set to indicate the error.21695 **ERRORS**21696 The *cfsetospeed()* function may fail if:21697 [EINVAL] The *speed* value is not a valid baud rate.21698 [EINVAL] The value of *speed* is outside the range of possible speed values as specified in
21699 <termios.h>.21700 **EXAMPLES**

21701 None.

21702 **APPLICATION USAGE**

21703 None.

21704 **RATIONALE**21705 Refer to *cfgetispeed()*.21706 **FUTURE DIRECTIONS**

21707 None.

21708 **SEE ALSO**21709 *cfgetispeed()*, *cfgetospeed()*, *cfsetispeed()*, *tcsetattr()*

21710 XBD Chapter 11 (on page 185), <termios.h>

21711 **CHANGE HISTORY**

21712 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

21713 **Issue 6**21714 The following new requirements on POSIX implementations derive from alignment with the
21715 Single UNIX Specification:

- 21716
- The optional setting of *errno* and the [EINVAL] error conditions are added.

21717 **NAME**

21718 chdir — change working directory

21719 **SYNOPSIS**

21720 #include <unistd.h>

21721 int chdir(const char *path);

21722 **DESCRIPTION**

21723 The *chdir()* function shall cause the directory named by the pathname pointed to by the *path*
 21724 argument to become the current working directory; that is, the starting point for path searches
 21725 for pathnames not beginning with '/ '.

21726 **RETURN VALUE**

21727 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned, the current
 21728 working directory shall remain unchanged, and *errno* shall be set to indicate the error.

21729 **ERRORS**21730 The *chdir()* function shall fail if:

21731 [EACCES] Search permission is denied for any component of the pathname.

21732 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 21733 argument.

21734 [ENAMETOOLONG]

21735 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 21736 component is longer than {NAME_MAX}.

21737 [ENOENT]

21738 A component of *path* does not name an existing directory or *path* is an empty
 string.

21739 [ENOTDIR]

A component of the pathname is not a directory.

21740 The *chdir()* function may fail if:

21741 [ELOOP]

21742 More than {SYMLOOP_MAX} symbolic links were encountered during
 resolution of the *path* argument.

21743 [ENAMETOOLONG]

21744 As a result of encountering a symbolic link in resolution of the *path* argument,
 21745 the length of the substituted pathname string exceeded {PATH_MAX}.

21746 **EXAMPLES**21747 **Changing the Current Working Directory**

21748 The following example makes the value pointed to by **directory**, */tmp*, the current working
 21749 directory.

21750 #include <unistd.h>

21751 ...

21752 char *directory = "/tmp";

21753 int ret;

21754 ret = chdir (directory);

21755 **APPLICATION USAGE**

21756 None.

21757 **RATIONALE**21758 The *chdir()* function only affects the working directory of the current process.21759 **FUTURE DIRECTIONS**

21760 None.

21761 **SEE ALSO**21762 *getcwd()*21763 XBD <[unistd.h](#)>21764 **CHANGE HISTORY**

21765 First released in Issue 1. Derived from Issue 1 of the SVID.

21766 **Issue 6**

21767 The APPLICATION USAGE section is added.

21768 The following new requirements on POSIX implementations derive from alignment with the
21769 Single UNIX Specification:

- 21770
- The [ELOOP] mandatory error condition is added.
 - A second [ENAMETOOLONG] is added as an optional error condition.
- 21771

21772 The following changes were made to align with the IEEE P1003.1a draft standard:

- 21773
- The [ELOOP] optional error condition is added.

21774 **NAME**
 21775 `chmod, fchmodat` — change mode of a file relative to directory file descriptor

21776 **SYNOPSIS**
 21777 `#include <sys/stat.h>`
 21778 `int chmod(const char *path, mode_t mode);`
 21779 `int fchmodat(int fd, const char *path, mode_t mode, int flag);`

21780 DESCRIPTION

21781 XSI The `chmod()` function shall change `S_ISUID`, `S_ISGID`, `S_ISVTX`, and the file permission bits of
 21782 the file named by the pathname pointed to by the `path` argument to the corresponding bits in the
 21783 `mode` argument. The application shall ensure that the effective user ID of the process matches the
 21784 owner of the file or the process has appropriate privileges in order to do this.

21785 XSI `S_ISUID`, `S_ISGID`, `S_ISVTX`, and the file permission bits are described in `<sys/stat.h>`.

21786 If the calling process does not have appropriate privileges, and if the group ID of the file does
 21787 not match the effective group ID or one of the supplementary group IDs and if the file is a
 21788 regular file, bit `S_ISGID` (set-group-ID on execution) in the file's mode shall be cleared upon
 21789 successful return from `chmod()`.

21790 Additional implementation-defined restrictions may cause the `S_ISUID` and `S_ISGID` bits in
 21791 `mode` to be ignored.

21792 The effect on file descriptors for files open at the time of a call to `chmod()` is implementation-
 21793 defined.

21794 Upon successful completion, `chmod()` shall mark for update the last file status change timestamp
 21795 of the file.

21796 The `fchmodat()` function shall be equivalent to the `chmod()` function except in the case where `path`
 21797 specifies a relative path. In this case the file to be changed is determined relative to the directory
 21798 associated with the file descriptor `fd` instead of the current working directory. It is unspecified
 21799 whether directory searches are permitted based on whether the file was opened with search
 21800 permission or on the current permissions of the directory underlying the file descriptor.

21801 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined
 21802 in `<fcntl.h>`:

21803 `AT_SYMLINK_NOFOLLOW`

21804 If `path` names a symbolic link, then the mode of the symbolic link is changed.

21805 If `fchmodat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working
 21806 directory is used. If also `flag` is zero, the behavior shall be identical to a call to `chmod()`.

21807 RETURN VALUE

21808 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 21809 return `-1` and set `errno` to indicate the error. If `-1` is returned, no change to the file mode occurs.

21810 ERRORS

21811 These functions shall fail if:

21812 [EACCES] Search permission is denied on a component of the path prefix.

21813 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
 21814 argument.

21815	[ENAMETOOLONG]	
21816		The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
21817		
21818	[ENOTDIR]	A component of the path prefix is not a directory.
21819	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
21820	[EPERM]	The effective user ID does not match the owner of the file and the process does not have appropriate privileges.
21821		
21822	[EROFS]	The named file resides on a read-only file system.
21823		The <i>fchmodat()</i> function shall fail if:
21824	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading.
21825		
21826		These functions may fail if:
21827	[EINTR]	A signal was caught during execution of the function.
21828	[EINVAL]	The value of the <i>mode</i> argument is invalid.
21829	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
21830		
21831	[ENAMETOOLONG]	
21832		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname strings exceeded {PATH_MAX}.
21833		
21834		The <i>fchmodat()</i> function may fail if:
21835	[EINVAL]	The value of the <i>flag</i> argument is invalid.
21836	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
21837		
21838	[EOPNOTSUPP]	The AT_SYMLINK_NOFOLLOW bit is set in the <i>flag</i> argument, <i>path</i> names a symbolic link, and the system does not support changing the mode of a symbolic link.
21839		
21840		

EXAMPLES**Setting Read Permissions for User, Group, and Others**

The following example sets read permissions for the owner, group, and others.

```
#include <sys/stat.h>
const char *path;
...
chmod(path, S_IRUSR | S_IRGRP | S_IROTH);
```

Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

```
#include <sys/stat.h>
const char *path;
...
chmod(path, S_IRWXU);
```


21855 **Setting Different Permissions for Owner, Group, and Other**

21856 The following example sets owner permissions for CHANGEFILE to read, write, and execute,
21857 group permissions to read and execute, and other permissions to read.

```
21858 #include <sys/stat.h>
21859 #define CHANGEFILE "/etc/myfile"
21860 ...
21861 chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

21862 **Setting and Checking File Permissions**

21863 The following example sets the file permission bits for a file named `/home/cnd/mod1`, then calls
21864 the `stat()` function to verify the permissions.

```
21865 #include <sys/types.h>
21866 #include <sys/stat.h>
21867 int status;
21868 struct stat buffer
21869 ...
21870 chmod("home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
21871 status = stat("home/cnd/mod1", &buffer);
```

21872 **APPLICATION USAGE**

21873 In order to ensure that the `S_ISUID` and `S_ISGID` bits are set, an application requiring this
21874 should use `stat()` after a successful `chmod()` to verify this.

21875 Any file descriptors currently open by any process on the file could possibly become invalid if
21876 the mode of the file is changed to a value which would deny access to that process. One
21877 situation where this could occur is on a stateless file system. This behavior will not occur in a
21878 conforming environment.

21879 **RATIONALE**

21880 This volume of POSIX.1-200x specifies that the `S_ISGID` bit is cleared by `chmod()` on a regular file
21881 under certain conditions. This is specified on the assumption that regular files may be executed,
21882 and the system should prevent users from making executable `setgid()` files perform with
21883 privileges that the caller does not have. On implementations that support execution of other file
21884 types, the `S_ISGID` bit should be cleared for those file types under the same circumstances.

21885 Implementations that use the `S_ISUID` bit to indicate some other function (for example,
21886 mandatory record locking) on non-executable files need not clear this bit on writing. They
21887 should clear the bit for executable files and any other cases where the bit grants special powers
21888 to processes that change the file contents. Similar comments apply to the `S_ISGID` bit.

21889 The purpose of the `fchmodat()` function is to enable changing the mode of files in directories
21890 other than the current working directory without exposure to race conditions. Any part of the
21891 path of a file could be changed in parallel to a call to `chmod()`, resulting in unspecified behavior.
21892 By opening a file descriptor for the target directory and using the `fchmodat()` function it can be
21893 guaranteed that the changed file is located relative to the desired directory. Some
21894 implementations might allow changing the mode of symbolic links. This is not supported by the
21895 interfaces in the POSIX specification. Systems with such support provide an interface named
21896 `lchmod()`. To support such implementations `fchmodat()` has a `flag` parameter.

21897 **FUTURE DIRECTIONS**

21898 None.

chmod()

21899 **SEE ALSO**
 21900 *access(), chown, exec, fstatat(), fstatofs(), mkdir, mkfifo, mknod(), open()*
 21901 XBD *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>* |

CHANGE HISTORY

21902 First released in Issue 1. Derived from Issue 1 of the SVID.
 21903

Issue 6

21904 The following new requirements on POSIX implementations derive from alignment with the
 21905 Single UNIX Specification:
 21906

- 21907 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was
 21908 required for conforming implementations of previous POSIX specifications, it was not
 21909 required for UNIX applications.
- 21910 • The [EINVAL] and [EINTR] optional error conditions are added.
- 21911 • A second [ENAMETOOLONG] is added as an optional error condition.

21912 The following changes were made to align with the IEEE P1003.1a draft standard:

- 21913 • The [ELOOP] optional error condition is added.

21914 The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

21915 The *fchmodat()* function is added from The Open Group Technical Standard, 2006, Extended API
 21916 Set Part 2.
 21917

21918 Changes are made related to support for finegrained timestamps. +

21919 **NAME**

21920 chown, fchownat — change owner and group of a file relative to directory file descriptor

21921 **SYNOPSIS**

```
21922 #include <unistd.h>
21923
21924 int chown(const char *path, uid_t owner, gid_t group);
21925 int fchownat(int fd, const char *path, uid_t owner, gid_t group,
21926             int flag);
```

21926 **DESCRIPTION**21927 The *chown()* function shall change the user and group ownership of a file.21928 The *path* argument points to a pathname naming a file. The user ID and group ID of the named
21929 file shall be set to the numeric values contained in *owner* and *group*, respectively.21930 Only processes with an effective user ID equal to the user ID of the file or with appropriate
21931 privileges may change the ownership of a file. If `_POSIX_CHOWN_RESTRICTED` is in effect for
21932 *path*:

- 21933 • Changing the user ID is restricted to processes with appropriate privileges.
- 21934 • Changing the group ID is permitted to a process with an effective user ID equal to the user
21935 ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's
21936 user ID or `(uid_t)-1` and *group* is equal either to the calling process' effective group ID or to
21937 one of its supplementary group IDs.

21938 If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of
21939 the file mode are set, and the process does not have appropriate privileges, the set-user-ID
21940 (`S_ISUID`) and set-group-ID (`S_ISGID`) bits of the file mode shall be cleared upon successful
21941 return from *chown()*. If the specified file is a regular file, one or more of the `S_IXUSR`, `S_IXGRP`,
21942 or `S_IXOTH` bits of the file mode are set, and the process has appropriate privileges, it is
21943 implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown()*
21944 function is successfully invoked on a file that is not a regular file and one or more of the
21945 `S_IXUSR`, `S_IXGRP`, or `S_IXOTH` bits of the file mode are set, the set-user-ID and set-group-ID
21946 bits may be cleared.

21947 If *owner* or *group* is specified as `(uid_t)-1` or `(gid_t)-1`, respectively, the corresponding ID of the
21948 file shall not be changed. If both *owner* and *group* are `-1`, the times need not be updated.21949 Upon successful completion, *chown()* shall mark for update the last file status change timestamp
21950 of the file.

21951 The *fchownat()* function shall be equivalent to the *chown()* and *lchown()* functions except in the
21952 case where *path* specifies a relative path. In this case the file to be changed is determined relative
21953 to the directory associated with the file descriptor *fd* instead of the current working directory. It
21954 is unspecified whether directory searches are permitted based on whether the file was opened
21955 with search permission or on the current permissions of the directory underlying the file
21956 descriptor.

21957 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
21958 in `<fcntl.h>`:21959 `AT_SYMLINK_NOFOLLOW`21960 If *path* names a symbolic link, ownership of the symbolic link is changed.

21961 If *fchownat()* is passed the special value `AT_FDCWD` in the *fd* parameter, the current working
21962 directory is used and the behavior shall be identical to a call to *chown()* or *lchown()* respectively,
21963 depending on whether or not the `AT_SYMLINK_NOFOLLOW` bit is set in the *flag* argument.

chown()21964 **RETURN VALUE**

21965 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 21966 return -1 and set *errno* to indicate the error. If -1 is returned, no changes are made in the user ID
 21967 and group ID of the file.

21968 **ERRORS**

21969 These functions shall fail if:

21970 [EACCES] Search permission is denied on a component of the path prefix.

21971 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 21972 argument.

21973 [ENAMETOOLONG]

21974 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 21975 component is longer than {NAME_MAX}.

21976 [ENOTDIR] A component of the path prefix is not a directory.

21977 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

21978 [EPERM] The effective user ID does not match the owner of the file, or the calling
 21979 process does not have appropriate privileges and
 21980 _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

21981 [EROFS] The named file resides on a read-only file system.

21982 The *fchownat()* function shall fail if:

21983 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 21984 neither AT_FDCWD nor a valid file descriptor open for reading.

21985 These functions may fail if:

21986 [EIO] An I/O error occurred while reading or writing to the file system.

21987 [EINTR] The *chown()* function was interrupted by a signal which was caught.

21988 [EINVAL] The owner or group ID supplied is not a value supported by the
 21989 implementation.

21990 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 21991 resolution of the *path* argument.

21992 [ENAMETOOLONG]

21993 As a result of encountering a symbolic link in resolution of the *path* argument,
 21994 the length of the substituted pathname string exceeded {PATH_MAX}.

21995 The *fchownat()* function may fail if:

21996 [EINVAL] The value of the *flag* argument is not valid.

21997 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
 21998 file descriptor associated with a directory.

21999 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not
 22000 support setting the owner or group of a symbolic link.

22001
22002**EXAMPLES**

None.

22003
22004
22005
22006**APPLICATION USAGE**

Although *chown()* can be used on some implementations by the file owner to change the owner and group to any desired values, the only portable use of this function is to change the group of a file to the effective GID of the calling process or to a member of its group set.

22007
22008
22009
22010
22011
22012
22013**RATIONALE**

System III and System V allow a user to give away files; that is, the owner of a file may change its user ID to anything. This is a serious problem for implementations that are intended to meet government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the user ID of a file. Some government agencies (usually not ones concerned directly with security) find this limitation too confining. This volume of POSIX.1-200x uses *may* to permit secure implementations while not disallowing System V.

22014
22015
22016
22017

System III and System V allow the owner of a file to change the group ID to anything. Version 7 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to change the group ID of a file to its effective group ID or to any of the groups in the list of supplementary group IDs, but to no others.

22018
22019
22020
22021
22022
22023
22024

The POSIX.1-1990 standard requires that the *chown()* function invoked by a non-appropriate privileged process clear the *S_ISGID* and the *S_ISUID* bits for regular files, and permits them to be cleared for other types of files. This is so that changes in accessibility do not accidentally cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-executable data files also clears the mandatory file locking bit (shared with *S_ISGID*), which is an extension on many implementations (it first appeared in System V). These bits should only be required to be cleared on regular files that have one or more of their execute bits set.

22025
22026
22027
22028
22029
22030

The purpose of the *fchownat()* function is to enable changing ownership of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chown()* or *lchown()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchownat()* function it can be guaranteed that the changed file is located relative to the desired directory.

22031
22032**FUTURE DIRECTIONS**

None.

22033
22034**SEE ALSO***chmod*, *fpathconf()*, *lchown()*

22035

XBD [<fcntl.h>](#), [<sys/types.h>](#), [<unistd.h>](#)22036
22037**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

22038
22039**Issue 6**

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

22040
22041
22042
22043
22044
22045

- The wording describing the optional dependency on `_POSIX_CHOWN_RESTRICTED` is restored.
- The `[EPERM]` error is restored as an error dependent on `_POSIX_CHOWN_RESTRICTED`. This is since its operand is a pathname and applications should be aware that the error may not occur for that pathname if the file system does not support `_POSIX_CHOWN_RESTRICTED`.

22046
22047

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

22048
22049
22050
22051
22052
22053
22054
22055
22056
22057
22058
22059
22060
22061
22062
22063

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The value for *owner* of `(uid_t)-1` allows the use of `-1` by the owner of a file to change the group ID only. A corresponding change is made for group.
- The [ELOOP] mandatory error condition is added.
- The [EIO] and [EINTR] optional error conditions are added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

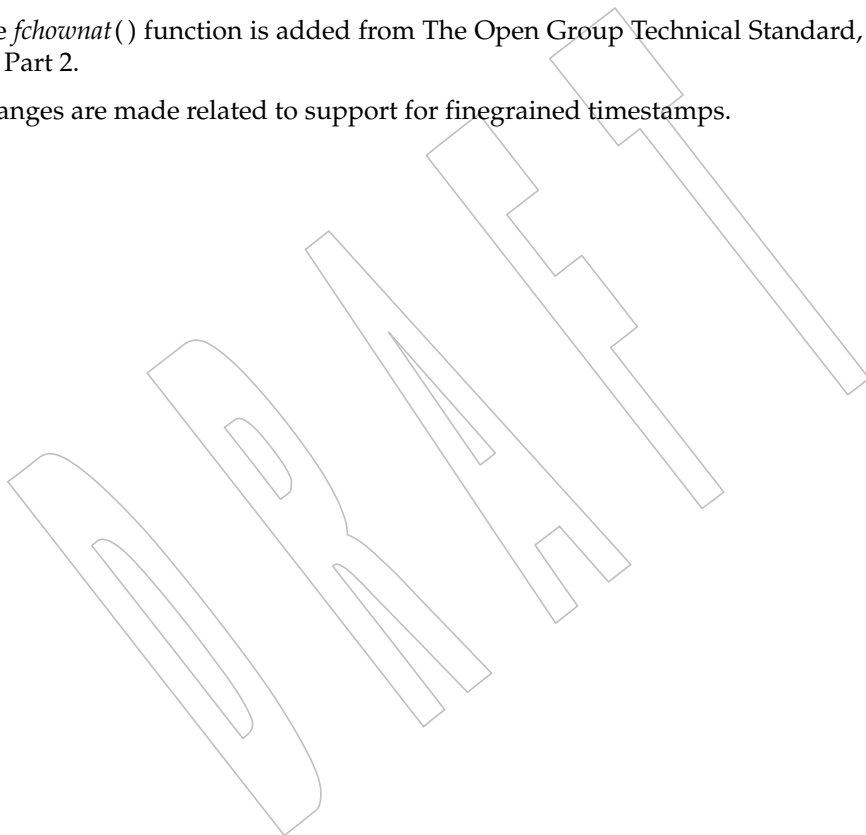
- Clarification is added that the `S_ISUID` and `S_ISGID` bits do not need to be cleared when the process has appropriate privileges.
- The [ELOOP] optional error condition is added.

Issue 7

The `fchownat()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

Changes are made related to support for finegrained timestamps.

+



22064 **NAME**

22065 `cimag, cimagf, cimagl` — complex imaginary functions

22066 **SYNOPSIS**

22067 `#include <complex.h>`

22068 `double cimag(double complex z);`

22069 `float cimagf(float complex z);`

22070 `long double cimagl(long double complex z);`

22071 **DESCRIPTION**

22072 CX The functionality described on this reference page is aligned with the ISO C standard. Any

22073 conflict between the requirements described here and the ISO C standard is unintentional. This

22074 volume of POSIX.1-200x defers to the ISO C standard.

22075 These functions shall compute the imaginary part of `z`.

22076 **RETURN VALUE**

22077 These functions shall return the imaginary part value (as a real).

22078 **ERRORS**

22079 No errors are defined.

22080 **EXAMPLES**

22081 None.

22082 **APPLICATION USAGE**

22083 For a variable `z` of complex type:

22084 `z == creal(z) + cimag(z)*I`

22085 **RATIONALE**

22086 None.

22087 **FUTURE DIRECTIONS**

22088 None.

22089 **SEE ALSO**

22090 [*carg\(\)*](#), [*conj\(\)*](#), [*cproj\(\)*](#), [*creal\(\)*](#)

22091 XBD [**<complex.h>**](#)

22092 **CHANGE HISTORY**

22093 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

clearerr()

22094 **NAME**
 22095 clearerr — clear indicators on a stream

22096 **SYNOPSIS**
 22097 #include <stdio.h>

22098 void clearerr(FILE *stream);

DESCRIPTION

22100 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22101 conflict between the requirements described here and the ISO C standard is unintentional. This
 22102 volume of POSIX.1-200x defers to the ISO C standard.

22103 The *clearerr()* function shall clear the end-of-file and error indicators for the stream to which
 22104 *stream* points.

RETURN VALUE

22105 The *clearerr()* function shall not return a value.
 22106

ERRORS

22107 No errors are defined.
 22108

EXAMPLES

22109 None.
 22110

APPLICATION USAGE

22111 None.
 22112

RATIONALE

22113 None.
 22114

FUTURE DIRECTIONS

22115 None.
 22116

SEE ALSO

22117 XBD <stdio.h>
 22118

CHANGE HISTORY

22119 First released in Issue 1. Derived from Issue 1 of the SVID.
 22120

22121 **NAME**

22122 clock — report CPU time used

22123 **SYNOPSIS**

22124 #include <time.h>

22125 clock_t clock(void);

22126 **DESCRIPTION**

22127 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22128 conflict between the requirements described here and the ISO C standard is unintentional. This
 22129 volume of POSIX.1-200x defers to the ISO C standard.

22130 The *clock()* function shall return the implementation's best approximation to the processor time
 22131 used by the process since the beginning of an implementation-defined era related only to the
 22132 process invocation.

22133 **RETURN VALUE**

22134 To determine the time in seconds, the value returned by *clock()* should be divided by the value
 22135 XSI of the macro `CLOCKS_PER_SEC`. `CLOCKS_PER_SEC` is defined to be one million in `<time.h>`.
 22136 If the processor time used is not available or its value cannot be represented, the function shall
 22137 return the value `(clock_t)-1`.

22138 **ERRORS**

22139 No errors are defined.

22140 **EXAMPLES**

22141 None.

22142 **APPLICATION USAGE**

22143 In order to measure the time spent in a program, *clock()* should be called at the start of the
 22144 program and its return value subtracted from the value returned by subsequent calls. The value
 22145 returned by *clock()* is defined for compatibility across systems that have clocks with different
 22146 resolutions. The resolution on any particular system need not be to microsecond accuracy.

22147 The value returned by *clock()* may wrap around on some implementations. For example, on a
 22148 machine with 32-bit values for `clock_t`, it wraps after 2 147 seconds or 36 minutes.

22149 **RATIONALE**

22150 None.

22151 **FUTURE DIRECTIONS**

22152 None.

22153 **SEE ALSO**22154 *asctime()*, *ctime()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time*, *utime()*22155 XBD `<time.h>`22156 **CHANGE HISTORY**

22157 First released in Issue 1. Derived from Issue 1 of the SVID.

22158 **NAME**22159 clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)22160 **SYNOPSIS**

```
22161 CPT #include <time.h>
22162 int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

22163 **DESCRIPTION**

22164 The *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of the process
 22165 specified by *pid*. If the process described by *pid* exists and the calling process has permission, the
 22166 clock ID of this clock shall be returned in *clock_id*.

22167 If *pid* is zero, the *clock_getcpuclockid()* function shall return the clock ID of the CPU-time clock of
 22168 the process making the call, in *clock_id*.

22169 The conditions under which one process has permission to obtain the CPU-time clock ID of
 22170 other processes are implementation-defined.

22171 **RETURN VALUE**

22172 Upon successful completion, *clock_getcpuclockid()* shall return zero; otherwise, an error number
 22173 shall be returned to indicate the error.

22174 **ERRORS**

22175 The *clock_getcpuclockid()* function shall fail if:

22176 [EPERM] The requesting process does not have permission to access the CPU-time clock
 22177 for the process.

22178 The *clock_getcpuclockid()* function may fail if:

22179 [ESRCH] No process can be found corresponding to the process specified by *pid*.

22180 **EXAMPLES**

22181 None.

22182 **APPLICATION USAGE**

22183 The *clock_getcpuclockid()* function is part of the Process CPU-Time Clocks option and need not be
 22184 provided on all implementations.

22185 **RATIONALE**

22186 None.

22187 **FUTURE DIRECTIONS**

22188 None.

22189 **SEE ALSO**

22190 *clock_getres()*, *timer_create()*

22191 XBD **<time.h>**

22192 **CHANGE HISTORY**

22193 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

22194 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

22195 **NAME**

22196 clock_getres, clock_gettime, clock_settime — clock and timer functions

22197 **SYNOPSIS**

```

22198 CX      #include <time.h>
22199
22199      int clock_getres(clockid_t clock_id, struct timespec *res);
22200      int clock_gettime(clockid_t clock_id, struct timespec *tp);
22201      int clock_settime(clockid_t clock_id, const struct timespec *tp);

```

22202 **DESCRIPTION**

22203 The *clock_getres()* function shall return the resolution of any clock. Clock resolutions are
 22204 implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the
 22205 resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL,
 22206 the clock resolution is not returned. If the *time* argument of *clock_settime()* is not a multiple of *res*,
 22207 then the value is truncated to a multiple of *res*.

22208 The *clock_gettime()* function shall return the current value *tp* for the specified clock, *clock_id*.

22209 The *clock_settime()* function shall set the specified clock, *clock_id*, to the value specified by *tp*.
 22210 Time values that are between two consecutive non-negative integer multiples of the resolution of
 22211 the specified clock shall be truncated down to the smaller multiple of the resolution.

22212 A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that
 22213 is meaningful only within a process). All implementations shall support a *clock_id* of
 22214 CLOCK_REALTIME as defined in **<time.h>**. This clock represents the realtime clock for the
 22215 system. For this clock, the values returned by *clock_gettime()* and specified by *clock_settime()*
 22216 represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation
 22217 may also support additional clocks. The interpretation of time values for these clocks is
 22218 unspecified.

22219 If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock
 22220 shall be used to determine the time of expiration for absolute time services based upon the
 22221 CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the
 22222 absolute time requested at the invocation of such a time service is before the new value of the
 22223 clock, the time service shall expire immediately as if the clock had reached the requested time
 22224 normally.

22225 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on
 22226 threads that are blocked waiting for a relative time service based upon this clock, including the
 22227 *nanosleep()* function; nor on the expiration of relative timers based upon this clock.
 22228 Consequently, these time services shall expire when the requested relative interval elapses,
 22229 independently of the new or old value of the clock.

22230 MON If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of
 22231 CLOCK_MONOTONIC defined in **<time.h>**. This clock represents the monotonic clock for the
 22232 system. For this clock, the value returned by *clock_gettime()* represents the amount of time (in
 22233 seconds and nanoseconds) since an unspecified point in the past (for example, system start-up
 22234 time, or the Epoch). This point does not change after system start-up time. The value of the
 22235 CLOCK_MONOTONIC clock cannot be set via *clock_settime()*. This function shall fail if it is
 22236 invoked with a *clock_id* argument of CLOCK_MONOTONIC.

22237 The effect of setting a clock via *clock_settime()* on armed per-process timers associated with a
 22238 clock other than CLOCK_REALTIME is implementation-defined.

22239 If the value of the CLOCK_REALTIME clock is set via *clock_settime()*, the new value of the clock

clock_getres()

22240 shall be used to determine the time at which the system shall awaken a thread blocked on an
 22241 absolute *clock_nanosleep()* call based upon the CLOCK_REALTIME clock. If the absolute time
 22242 requested at the invocation of such a time service is before the new value of the clock, the call
 22243 shall return immediately as if the clock had reached the requested time normally.

22244 Setting the value of the CLOCK_REALTIME clock via *clock_settime()* shall have no effect on any
 22245 thread that is blocked on a relative *clock_nanosleep()* call. Consequently, the call shall return
 22246 when the requested relative interval elapses, independently of the new or old value of the clock.

22247 The appropriate privilege to set a particular clock is implementation-defined.

22248 CPT If `_POSIX_CPUTIME` is defined, implementations shall support clock ID values obtained by
 22249 invoking *clock_getcpuclockid()*, which represent the CPU-time clock of a given process.
 22250 Implementations shall also support the special `clockid_t` value
 22251 `CLOCK_PROCESS_CPUTIME_ID`, which represents the CPU-time clock of the calling process
 22252 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 22253 returned by *clock_gettime()* and specified by *clock_settime()* represent the amount of execution
 22254 time of the process associated with the clock. Changing the value of a CPU-time clock via
 22255 *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy (see
 22256 [Scheduling Policies](#), on page 479).

22257 TCT If `_POSIX_THREAD_CPUTIME` is defined, implementations shall support clock ID values
 22258 obtained by invoking *pthread_getcpuclockid()*, which represent the CPU-time clock of a given
 22259 thread. Implementations shall also support the special `clockid_t` value
 22260 `CLOCK_THREAD_CPUTIME_ID`, which represents the CPU-time clock of the calling thread
 22261 when invoking one of the *clock_**() or *timer_**() functions. For these clock IDs, the values
 22262 returned by *clock_gettime()* and specified by *clock_settime()* shall represent the amount of
 22263 execution time of the thread associated with the clock. Changing the value of a CPU-time clock
 22264 via *clock_settime()* shall have no effect on the behavior of the sporadic server scheduling policy
 22265 (see [Scheduling Policies](#), on page 479).

RETURN VALUE

22266 A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that
 22267 an error occurred, and *errno* shall be set to indicate the error.

ERRORS

22269 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions shall fail if:

22270 [EINVAL] The *clock_id* argument does not specify a known clock.

22271 The *clock_settime()* function shall fail if:

22272 [EINVAL] The *tp* argument to *clock_settime()* is outside the range for the given clock ID.

22273 [EINVAL] The *tp* argument specified a nanosecond value less than zero or greater than or
 22274 equal to 1 000 million.

22275 [EINVAL] The value of the *clock_id* argument is `CLOCK_MONOTONIC`.

22276 The *clock_settime()* function may fail if:

22277 [EPERM] The requesting process does not have the appropriate privilege to set the
 22278 specified clock.

EXAMPLES

22280
22281 None.

APPLICATION USAGE

22282 Note that the absolute value of the monotonic clock is meaningless (because its origin is
22283 arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the
22284 fact that the value of this clock is never set and, therefore, that time intervals measured with this
22285 clock will not be affected by calls to *clock_settime()*.
22286

RATIONALE

22287
22288 None.

FUTURE DIRECTIONS

22289
22290 None.

SEE ALSO

22291 *clock_getcpuclockid()*, *clock_nanosleep()*, *ctime()*, *mq_receive()*, *mq_send()*, *nanosleep()*,
22292 *pthread_mutex_timedlock()*, *sem_timedwait()*, *time*, *timer_create()*, *timer_getoverrun()*
22293

22294 XBD [<time.h>](#) +

CHANGE HISTORY

22295 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
22296

Issue 6

22297 The [ENOSYS] error condition has been removed as stubs need not be provided if an
22298 implementation does not support the Timers option.
22299

22300 The APPLICATION USAGE section is added.

22301 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22302 • Clarification is added of the effect of resetting the clock resolution.

22303 CPU-time clocks and the *clock_getcpuclockid()* function are added for alignment with IEEE Std
22304 1003.1d-1999.

22305 The following changes are added for alignment with IEEE Std 1003.1j-2000:

- 22306 • The DESCRIPTION is updated as follows:
 - 22307 — The value returned by *clock_gettime()* for CLOCK_MONOTONIC is specified.
 - 22308 — The *clock_settime()* function failing for CLOCK_MONOTONIC is specified.
 - 22309 — The effects of *clock_settime()* on the *clock_nanosleep()* function with respect to
22310 CLOCK_REALTIME are specified.
- 22311 • An [EINVAL] error is added to the ERRORS section, indicating that *clock_settime()* fails for
22312 CLOCK_MONOTONIC.
- 22313 • The APPLICATION USAGE section notes that the CLOCK_MONOTONIC clock need not
22314 and shall not be set by *clock_settime()* since the absolute value of the
22315 CLOCK_MONOTONIC clock is meaningless.
- 22316 • The *clock_nanosleep()*, *mq_timedreceive()*, *mq_timedsend()*, *pthread_mutex_timedlock()*,
22317 *sem_timedwait()*, *timer_create()*, and *timer_settime()* functions are added to the SEE ALSO
22318 section.

Issue 7

22319 Functionality relating to the Clock Selection option is moved to the Base.
22320

22321 The *clock_getres()*, *clock_gettime()*, and *clock_settime()* functions are moved from the Timers
22322 option to the Base.

22323 **NAME**

22324 clock_nanosleep — high resolution sleep with specifiable clock

22325 **SYNOPSIS**

```
22326 CX #include <time.h>
22327
22327 int clock_nanosleep(clockid_t clock_id, int flags,
22328 const struct timespec *rqtp, struct timespec *rmtp);
```

22329 **DESCRIPTION**

22330 If the flag `TIMER_ABSTIME` is not set in the *flags* argument, the `clock_nanosleep()` function shall
 22331 cause the current thread to be suspended from execution until either the time interval specified
 22332 by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to
 22333 invoke a signal-catching function, or the process is terminated. The clock used to measure the
 22334 time shall be the clock specified by *clock_id*.

22335 If the flag `TIMER_ABSTIME` is set in the *flags* argument, the `clock_nanosleep()` function shall
 22336 cause the current thread to be suspended from execution until either the time value of the clock
 22337 specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is
 22338 delivered to the calling thread and its action is to invoke a signal-catching function, or the
 22339 process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or
 22340 equal to the time value of the specified clock, then `clock_nanosleep()` shall return immediately
 22341 and the calling process shall not be suspended.

22342 The suspension time caused by this function may be longer than requested because the
 22343 argument value is rounded up to an integer multiple of the sleep resolution, or because of the
 22344 scheduling of other activity by the system. But, except for the case of being interrupted by a
 22345 signal, the suspension time for the relative `clock_nanosleep()` function (that is, with the
 22346 `TIMER_ABSTIME` flag not set) shall not be less than the time interval specified by *rqtp*, as
 22347 measured by the corresponding clock. The suspension for the absolute `clock_nanosleep()` function
 22348 (that is, with the `TIMER_ABSTIME` flag set) shall be in effect at least until the value of the
 22349 corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being
 22350 interrupted by a signal.

22351 The use of the `clock_nanosleep()` function shall have no effect on the action or blockage of any
 22352 signal.

22353 The `clock_nanosleep()` function shall fail if the *clock_id* argument refers to the CPU-time clock of
 22354 the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

22355 **RETURN VALUE**

22356 If the `clock_nanosleep()` function returns because the requested time has elapsed, its return value
 22357 shall be zero.

22358 If the `clock_nanosleep()` function returns because it has been interrupted by a signal, it shall
 22359 return the corresponding error value. For the relative `clock_nanosleep()` function, if the *rmtp*
 22360 argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the
 22361 amount of time remaining in the interval (the requested time minus the time actually slept). If
 22362 the *rmtp* argument is NULL, the remaining time is not returned. The absolute `clock_nanosleep()`
 22363 function has no effect on the structure referenced by *rmtp*.

22364 If `clock_nanosleep()` fails, it shall return the corresponding error value.

ERRORS

- 22365
22366 The *clock_nanosleep()* function shall fail if:
- 22367 [EINTR] The *clock_nanosleep()* function was interrupted by a signal.
- 22368 [EINVAL] The *rqtpt* argument specified a nanosecond value less than zero or greater than
22369 or equal to 1 000 million; or the *TIMER_ABSTIME* flag was specified in *flags*
22370 and the *rqtpt* argument is outside the range for the clock specified by *clock_id*;
22371 or the *clock_id* argument does not specify a known clock, or specifies the CPU-
22372 time clock of the calling thread.
- 22373 [ENOTSUP] The *clock_id* argument specifies a clock for which *clock_nanosleep()* is not
22374 supported, such as a CPU-time clock.

EXAMPLES

- 22375
22376 None.

APPLICATION USAGE

- 22377
22378 Calling *clock_nanosleep()* with the value *TIMER_ABSTIME* not set in the *flags* argument and with
22379 a *clock_id* of *CLOCK_REALTIME* is equivalent to calling *nanosleep()* with the same *rqtpt* and *rmtpt*
22380 arguments.

RATIONALE

- 22381
22382 The *nanosleep()* function specifies that the system-wide clock *CLOCK_REALTIME* is used to
22383 measure the elapsed time for this time service. However, with the introduction of the monotonic
22384 clock *CLOCK_MONOTONIC* a new relative sleep function is needed to allow an application to
22385 take advantage of the special characteristics of this clock.

- 22386
22387 There are many applications in which a process needs to be suspended and then activated
22388 multiple times in a periodic way; for example, to poll the status of a non-interrupting device or
22389 to refresh a display device. For these cases, it is known that precise periodic activation cannot be
22390 achieved with a relative *sleep()* or *nanosleep()* function call. Suppose, for example, a periodic
22391 process that is activated at time *T0*, executes for a while, and then wants to suspend itself until
22392 time *T0+T*, the period being *T*. If this process wants to use the *nanosleep()* function, it must first
22393 call *clock_gettime()* to get the current time, then calculate the difference between the current time
22394 and *T0+T* and, finally, call *nanosleep()* using the computed interval. However, the process could
22395 be preempted by a different process between the two function calls, and in this case the interval
22396 computed would be wrong; the process would wake up later than desired. This problem would
22397 not occur with the absolute *clock_nanosleep()* function, since only one function call would be
22398 necessary to suspend the process until the desired time. In other cases, however, a relative sleep
is needed, and that is why both functionalities are required.

- 22399
22400 Although it is possible to implement periodic processes using the timers interface, this
22401 implementation would require the use of signals, and the reservation of some signal numbers. In
22402 this regard, the reasons for including an absolute version of the *clock_nanosleep()* function in
POSIX.1-200x are the same as for the inclusion of the relative *nanosleep()*.

- 22403
22404 It is also possible to implement precise periodic processes using *pthread_cond_timedwait()*, in
22405 which an absolute timeout is specified that takes effect if the condition variable involved is never
22406 signaled. However, the use of this interface is unnatural, and involves performing other
22407 operations on mutexes and condition variables that imply an unnecessary overhead.
22408 Furthermore, *pthread_cond_timedwait()* is not available in implementations that do not support
threads.

- 22409
22410 Although the interface of the relative and absolute versions of the new high resolution sleep
22411 service is the same *clock_nanosleep()* function, the *rmtpt* argument is only used in the relative
22412 sleep. This argument is needed in the relative *clock_nanosleep()* function to reissue the function
22413 call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep()* function
call; if the call is interrupted by a signal, the absolute *clock_nanosleep()* function can be invoked

clock_nanosleep()

22414 again with the same *rqtp* argument used in the interrupted call.

FUTURE DIRECTIONS

22415 None.

SEE ALSO

22418 [clock_getres\(\)](#), [nanosleep\(\)](#), [pthread_cond_timedwait\(\)](#), [sleep](#)

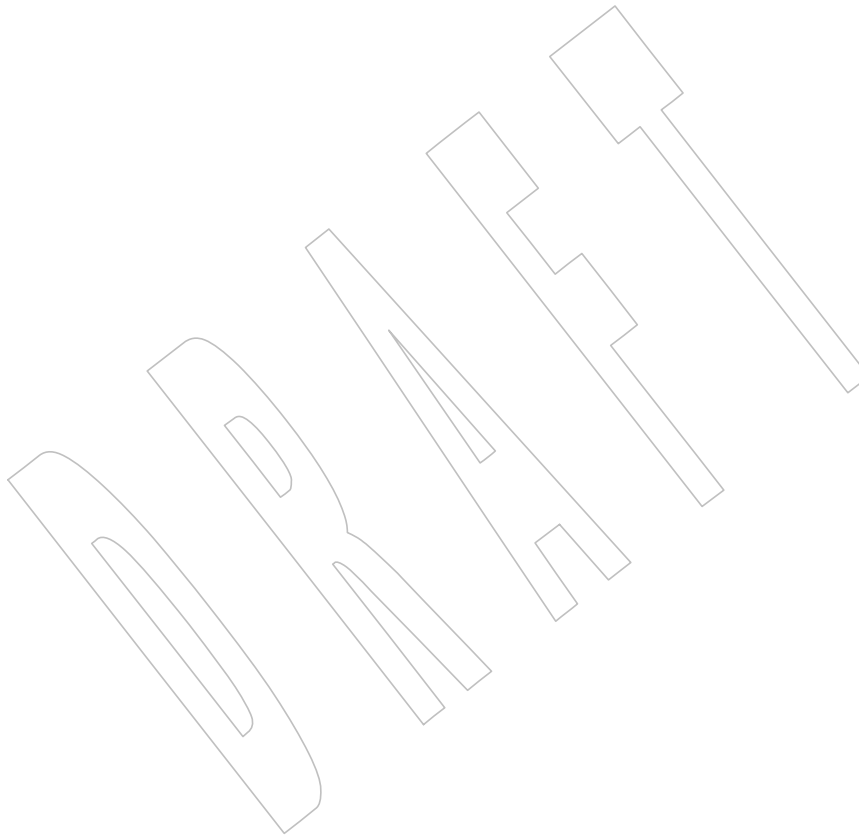
22419 XBD <[time.h](#)>

CHANGE HISTORY

22420 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

Issue 7

22422 The *clock_nanosleep()* function is moved from the Clock Selection option to the Base.



22424 **NAME**
22425 `clock_settime` — clock and timer functions

22426 **SYNOPSIS**

```
22427 CX #include <time.h>  
22428 int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

22429 **DESCRIPTION**

22430 Refer to [clock_getres\(\)](#).

22431 **NAME**
 22432 `clog, clogf, clogl` — complex natural logarithm functions

22433 **SYNOPSIS**
 22434 `#include <complex.h>`
 22435 `double complex clog(double complex z);`
 22436 `float complex clogf(float complex z);`
 22437 `long double complex clogl(long double complex z);`

22438 **DESCRIPTION**
 22439 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22440 conflict between the requirements described here and the ISO C standard is unintentional. This
 22441 volume of POSIX.1-200x defers to the ISO C standard.

22442 These functions shall compute the complex natural (base e) logarithm of z , with a branch cut
 22443 along the negative real axis.

22444 **RETURN VALUE**
 22445 These functions shall return the complex natural logarithm value, in the range of a strip
 22446 mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary
 22447 axis.

22448 **ERRORS**
 22449 No errors are defined.

22450 **EXAMPLES**
 22451 None.

22452 **APPLICATION USAGE**
 22453 None.

22454 **RATIONALE**
 22455 None.

22456 **FUTURE DIRECTIONS**
 22457 None.

22458 **SEE ALSO**
 22459 [`cexp\(\)`](#)
 22460 XBD [<complex.h>](#)

22461 **CHANGE HISTORY**
 22462 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

22463 **NAME**

22464 close — close a file descriptor

22465 **SYNOPSIS**

```
22466 #include <unistd.h>
22467 int close(int fdes);
```

22468 **DESCRIPTION**

22469 The *close()* function shall deallocate the file descriptor indicated by *fdes*. To deallocate means to
 22470 make the file descriptor available for return by subsequent calls to *open()* or other functions that
 22471 allocate file descriptors. All outstanding record locks owned by the process on the file associated
 22472 with the file descriptor shall be removed (that is, unlocked).

22473 If *close()* is interrupted by a signal that is to be caught, it shall return -1 with *errno* set to [EINTR]
 22474 and the state of *fdes* is unspecified. If an I/O error occurred while reading from or writing to
 22475 the file system during *close()*, it may return -1 with *errno* set to [EIO]; if this error is returned, the
 22476 state of *fdes* is unspecified.

22477 When all file descriptors associated with a pipe or FIFO special file are closed, any data
 22478 remaining in the pipe or FIFO shall be discarded.

22479 When all file descriptors associated with an open file description have been closed, the open file
 22480 description shall be freed.

22481 If the link count of the file is 0, when all file descriptors associated with the file are closed, the
 22482 space occupied by the file shall be freed and the file shall no longer be accessible.

22483 OB XSR If a STREAMS-based *fdes* is closed and the calling process was previously registered to receive
 22484 a SIGPOLL signal for events associated with that STREAM, the calling process shall be
 22485 unregistered for events associated with the STREAM. The last *close()* for a STREAM shall cause
 22486 the STREAM associated with *fdes* to be dismantled. If O_NONBLOCK is not set and there have
 22487 been no signals posted for the STREAM, and if there is data on the module's write queue, *close()*
 22488 shall wait for an unspecified time (for each module and driver) for any output to drain before
 22489 dismantling the STREAM. The time delay can be changed via an L_SETCLTIME *ioctl()* request. If
 22490 the O_NONBLOCK flag is set, or if there are any pending signals, *close()* shall not wait for
 22491 output to drain, and shall dismantle the STREAM immediately.

22492 If the implementation supports STREAMS-based pipes, and *fdes* is associated with one end of a
 22493 pipe, the last *close()* shall cause a hangup to occur on the other end of the pipe. In addition, if the
 22494 other end of the pipe has been named by *fattach()*, then the last *close()* shall force the named end
 22495 to be detached by *fdetach()*. If the named end has no open file descriptors associated with it and
 22496 gets detached, the STREAM associated with that end shall also be dismantled.

22497 XSI If *fdes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal
 22498 shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is
 22499 the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal
 22500 flushes all queued input and output.

22501 OB XSR If *fdes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message
 22502 may be sent to the master.

22503 When there is an outstanding cancelable asynchronous I/O operation against *fdes* when *close()*
 22504 is called, that I/O operation may be canceled. An I/O operation that is not canceled completes
 22505 as if the *close()* operation had not yet occurred. All operations that are not canceled shall
 22506 complete as if the *close()* blocked until the operations completed. The *close()* operation itself
 22507 need not block awaiting such I/O completion. Whether any I/O operation is canceled, and
 22508 which I/O operation may be canceled upon *close()*, is implementation-defined.

close()

22509 SHM If a memory mapped file or a shared memory object remains referenced at the last close (that is,
 22510 a process has it mapped), then the entire contents of the memory object shall persist until the
 22511 SHM memory object becomes unreferenced. If this is the last close of a memory mapped file or a
 22512 shared memory object and the close results in the memory object becoming unreferenced, and
 22513 the memory object has been unlinked, then the memory object shall be removed.

22514 If *fdes* refers to a socket, *close()* shall cause the socket to be destroyed. If the socket is in
 22515 connection-mode, and the `SO_LINGER` option is set for the socket with non-zero linger time,
 22516 and the socket has untransmitted data, then *close()* shall block for up to the current linger
 22517 interval until all data is transmitted.

RETURN VALUE

22518 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 22519 indicate the error.
 22520

ERRORS

22521 The *close()* function shall fail if:

22522 [EBADF] The *fdes* argument is not a valid file descriptor.

22523 [EINTR] The *close()* function was interrupted by a signal.

22524 The *close()* function may fail if:

22525 [EIO] An I/O error occurred while reading from or writing to the file system.
 22526

EXAMPLES**Reassigning a File Descriptor**

22527
 22528 The following example closes the file descriptor associated with standard output for the current
 22529 process, re-assigns standard output to a new file descriptor, and closes the original file descriptor
 22530 to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard
 22531 input) is not closed.
 22532

```
22533 #include <unistd.h>
22534 ...
22535 int pfd;
22536 ...
22537 close(1);
22538 dup(pfd);
22539 close(pfd);
22540 ...
```

22541 Incidentally, this is exactly what could be achieved using:

```
22542 dup2(pfd, 1);
22543 close(pfd);
```

Closing a File Descriptor

22544 In the following example, *close()* is used to close a file descriptor after an unsuccessful attempt is
 22545 made to associate that file descriptor with a stream.
 22546

```
22547 #include <stdio.h>
22548 #include <unistd.h>
22549 #include <stdlib.h>
22550 #define LOCKFILE "/etc/ptmp"
22551 ...
22552 int pfd;
```

```

22553     FILE *fpfd;
22554     ...
22555     if ((fpfd = fdopen (pfd, "w")) == NULL) {
22556         close(pfd);
22557         unlink(LOCKFILE);
22558         exit(1);
22559     }
22560     ...

```

APPLICATION USAGE

An application that had used the *stdio* routine *fopen()* to open a file should use the corresponding *fclose()* routine rather than *close()*. Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors by *exec* and *exit()*. This volume of POSIX.1-200x only intends to permit such behavior by specifying the [EINTR] error condition.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 473), *exec*, *fattach()*, *fclose()*, *fdetach()*, *fopen()*, *ioctl()*, *open()*, *unlink*

XBD <[unistd.h](#)>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

Issue 6

The DESCRIPTION related to a STREAMS-based file or pseudo-terminal is marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error condition is added as an optional error.
- The DESCRIPTION is updated to describe the state of the *fildev* file descriptor as unspecified if an I/O error occurs and an [EIO] error condition is returned.

Text referring to sockets is added to the DESCRIPTION.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that shared memory objects and memory mapped files (and not typed memory objects) are the types of memory objects to which the paragraph on last closes applies.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/12 is applied, correcting the XSH shaded text relating to the master side of a pseudo-terminal. The reason for the change is that the behavior of pseudo-terminals and regular terminals should be as much alike as possible in this case; the change achieves that and matches historical behavior.

Issue 7

Functionality relating to the XSI STREAMS option is marked obsolescent.

Functionality relating to the Asynchronous Input and Output and Memory Mapped Files options is moved to the Base.

22598 **NAME**
 22599 `closedir` — close a directory stream

22600 **SYNOPSIS**
 22601 `#include <dirent.h>`
 22602 `int closedir(DIR *dirp);`

22603 **DESCRIPTION**
 22604 The `closedir()` function shall close the directory stream referred to by the argument `dirp`. Upon
 22605 return, the value of `dirp` may no longer point to an accessible object of the type **DIR**. If a file
 22606 descriptor is used to implement type **DIR**, that file descriptor shall be closed.

22607 **RETURN VALUE**
 22608 Upon successful completion, `closedir()` shall return 0; otherwise, `-1` shall be returned and `errno`
 22609 set to indicate the error.

22610 **ERRORS**
 22611 The `closedir()` function may fail if:
 22612 [EBADF] The `dirp` argument does not refer to an open directory stream.
 22613 [EINTR] The `closedir()` function was interrupted by a signal.

22614 **EXAMPLES**
 22615 **Closing a Directory Stream**
 22616 The following program fragment demonstrates how the `closedir()` function is used.

```
22617 ...
22618     DIR *dir;
22619     struct dirent *dp;
22620 ...
22621     if ((dir = opendir(".")) == NULL) {
22622     ...
22623     }
22624     while ((dp = readdir(dir)) != NULL) {
22625     ...
22626     }
22627     closedir(dir);
22628     ...
```

22629 **APPLICATION USAGE**
 22630 None.

22631 **RATIONALE**
 22632 None.

22633 **FUTURE DIRECTIONS**
 22634 None.

22635 **SEE ALSO**
 22636 [`dirfd\(\)`](#), [`fdopendir\(\)`](#)
 22637 XBD [<dirent.h>](#)

22638
22639

22640
22641

22642
22643

22644
22645
22646

22647

CHANGE HISTORY

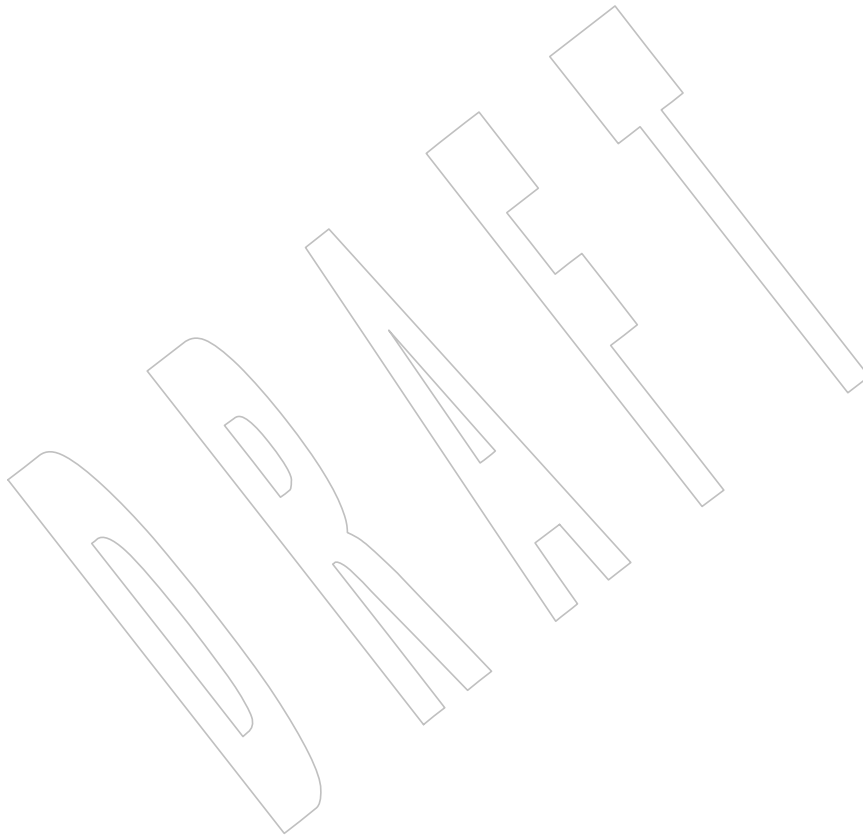
First released in Issue 2.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [EINTR] error condition is added as an optional error condition.



22648 **NAME**

22649 closelog, openlog, setlogmask, syslog — control system log

22650 **SYNOPSIS**

```

22651 XSI #include <syslog.h>
22652
22652 void closelog(void);
22653 void openlog(const char *ident, int logopt, int facility);
22654 int setlogmask(int maskpri);
22655 void syslog(int priority, const char *message, ... /* arguments */);

```

22656 **DESCRIPTION**

22657 The *syslog()* function shall send a message to an implementation-defined logging facility, which
 22658 may log it in an implementation-defined system log, write it to the system console, forward it to
 22659 a list of users, or forward it to the logging facility on another host over the network. The logged
 22660 message shall include a message header and a message body. The message header contains at
 22661 least a timestamp and a tag string.

22662 The message body is generated from the *message* and following arguments in the same manner
 22663 as if these were arguments to *printf()*, except that the additional conversion specification *%m*
 22664 shall be recognized; it shall convert no arguments, shall cause the output of the error message
 22665 string associated with the value of *errno* on entry to *syslog()*, and may be mixed with argument
 22666 specifications of the "*%n\$*" form. If a complete conversion specification with the *m* conversion
 22667 specifier character is not just *%m*, the behavior is undefined. A trailing <newline> may be added
 22668 if needed.

22669 Values of the *priority* argument are formed by OR'ing together a severity-level value and an
 22670 optional facility value. If no facility value is specified, the current default facility value is used.

22671 Possible values of severity level include:

22672	LOG_EMERG	A panic condition.
22673	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system 22674 database.
22675	LOG_CRIT	Critical conditions, such as hard device errors.
22676	LOG_ERR	Errors.
22677	LOG_WARNING	Warning messages. 22678
22679	LOG_NOTICE	Conditions that are not error conditions, but that may require special 22680 handling.
22681	LOG_INFO	Informational messages.
22682	LOG_DEBUG	Messages that contain information normally of use only when debugging a 22683 program.

22684 The facility indicates the application or system component generating the message. Possible
 22685 facility values include:

22686	LOG_USER	Messages generated by arbitrary processes. This is the default facility 22687 identifier if none is specified.
-------	----------	---

22688	LOG_LOCAL0	Reserved for local use.
22689	LOG_LOCAL1	Reserved for local use.
22690	LOG_LOCAL2	Reserved for local use.
22691	LOG_LOCAL3	Reserved for local use.
22692	LOG_LOCAL4	Reserved for local use.
22693	LOG_LOCAL5	Reserved for local use.
22694	LOG_LOCAL6	Reserved for local use.
22695	LOG_LOCAL7	Reserved for local use.

22696 The *openlog()* function shall set process attributes that affect subsequent calls to *syslog()*. The
 22697 *ident* argument is a string that is prepended to every message. The *logopt* argument indicates
 22698 logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of
 22699 the following:

22700	LOG_PID	Log the process ID with each message. This is useful for identifying specific 22701 processes.
22702	LOG_CONS	Write messages to the system console if they cannot be sent to the logging 22703 facility. The <i>syslog()</i> function ensures that the process does not acquire the 22704 console as a controlling terminal in the process of writing the message.
22705	LOG_NDELAY	Open the connection to the logging facility immediately. Normally the open is 22706 delayed until the first message is logged. This is useful for programs that need 22707 to manage the order in which file descriptors are allocated.
22708	LOG_ODELAY	Delay open until <i>syslog()</i> is called.
22709	LOG_NOWAIT	Do not wait for child processes that may have been created during the course 22710 of logging the message. This option should be used by processes that enable 22711 notification of child termination using SIGCHLD, since <i>syslog()</i> may 22712 otherwise block waiting for a child whose exit status has already been 22713 collected.

22714 The *facility* argument encodes a default facility to be assigned to all messages that do not have an
 22715 explicit facility already encoded. The initial default facility is LOG_USER.

22716 The *openlog()* and *syslog()* functions may allocate a file descriptor. It is not necessary to call
 22717 *openlog()* prior to calling *syslog()*.

22718 The *closelog()* function shall close any open file descriptors allocated by previous calls to
 22719 *openlog()* or *syslog()*.

22720 The *setlogmask()* function shall set the log priority mask for the current process to *maskpri* and
 22721 return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified.
 22722 Calls by the current process to *syslog()* with a priority not set in *maskpri* shall be rejected. The
 22723 default log mask allows all priorities to be logged. A call to *openlog()* is not required prior to
 22724 calling *setlogmask()*.

22725 Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are
 22726 defined in the **<syslog.h>** header.

22727 RETURN VALUE

22728 The *setlogmask()* function shall return the previous log priority mask. The *closelog()*, *openlog()*,
 22729 and *syslog()* functions shall not return a value.

closelog()22730 **ERRORS**

22731 No errors are defined.

22732 **EXAMPLES**22733 **Using openlog()**

22734 The following example causes subsequent calls to *syslog()* to log the process ID with each
 22735 message, and to write messages to the system console if they cannot be sent to the logging
 22736 facility.

```
22737 #include <syslog.h>
22738 char *ident = "Process demo";
22739 int logopt = LOG_PID | LOG_CONS;
22740 int facility = LOG_USER;
22741 ...
22742 openlog(ident, logopt, facility);
```

22743 **Using setlogmask()**

22744 The following example causes subsequent calls to *syslog()* to accept error messages, and to reject
 22745 all other messages.

```
22746 #include <syslog.h>
22747 int result;
22748 int mask = LOG_MASK (LOG_ERR);
22749 ...
22750 result = setlogmask(mask);
```

22751 **Using syslog**

22752 The following example sends the message "This is a message" to the default logging
 22753 facility, marking the message as an error message generated by random processes.

```
22754 #include <syslog.h>
22755 char *message = "This is a message";
22756 int priority = LOG_ERR | LOG_USER;
22757 ...
22758 syslog(priority, message);
```

22759 **APPLICATION USAGE**

22760 None.

22761 **RATIONALE**

22762 None.

22763 **FUTURE DIRECTIONS**

22764 None.

22765 **SEE ALSO**22766 *fprintf()*22767 XBD *<syslog.h>*

22768

CHANGE HISTORY

22769

First released in Issue 4, Version 2.

22770

Issue 5

22771

Moved from X/OPEN UNIX extension to BASE.

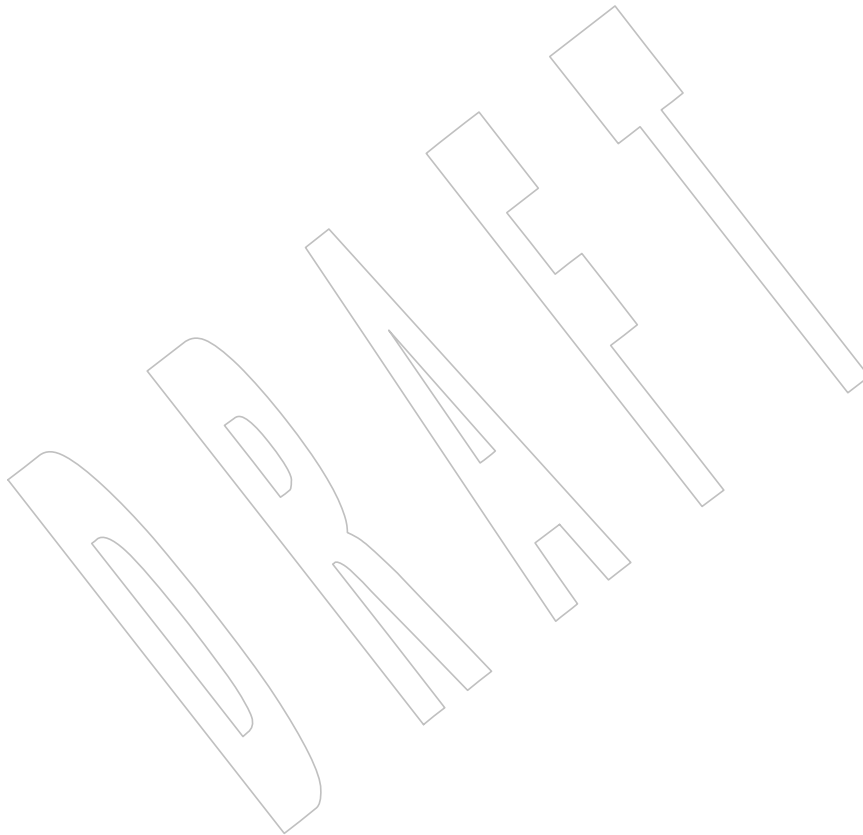
22772

Issue 6

22773

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/13 is applied, correcting the EXAMPLES section.

22774



22775 **NAME**
 22776 confstr — get configurable variables

22777 **SYNOPSIS**
 22778 #include <unistd.h>

22779 size_t confstr(int name, char *buf, size_t len);

22780 **DESCRIPTION**

22781 The *confstr()* function shall return configuration-defined string values. Its use and purpose are
 22782 similar to *sysconf()*, but it is used where string values rather than numeric values are returned.

22783 The *name* argument represents the system variable to be queried. The implementation shall
 22784 support the following name values, defined in <unistd.h>. It may support others:

22785 _CS_PATH
 22786 _CS_POSIX_V7_ILP32_OFF32_CFLAGS
 22787 _CS_POSIX_V7_ILP32_OFF32_LDFLAGS
 22788 _CS_POSIX_V7_ILP32_OFF32_LIBS
 22789 _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
 22790 _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
 22791 _CS_POSIX_V7_ILP32_OFFBIG_LIBS
 22792 _CS_POSIX_V7_LP64_OFF64_CFLAGS
 22793 _CS_POSIX_V7_LP64_OFF64_LDFLAGS
 22794 _CS_POSIX_V7_LP64_OFF64_LIBS
 22795 _CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS
 22796 _CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS
 22797 _CS_POSIX_V7_LPBIG_OFFBIG_LIBS
 22798 _CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
 22799 _CS_V7_ENV

22800 OB _CS_POSIX_V6_ILP32_OFF32_CFLAGS
 22801 _CS_POSIX_V6_ILP32_OFF32_LDFLAGS
 22802 _CS_POSIX_V6_ILP32_OFF32_LIBS
 22803 _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
 22804 _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
 22805 _CS_POSIX_V6_ILP32_OFFBIG_LIBS
 22806 _CS_POSIX_V6_LP64_OFF64_CFLAGS
 22807 _CS_POSIX_V6_LP64_OFF64_LDFLAGS
 22808 _CS_POSIX_V6_LP64_OFF64_LIBS
 22809 _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
 22810 _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
 22811 _CS_POSIX_V6_LPBIG_OFFBIG_LIBS
 22812 _CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
 22813 _CS_V6_ENV

22814 If *len* is not 0, and if *name* has a configuration-defined value, *confstr()* shall copy that value into
 22815 the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes,
 22816 including the terminating null, then *confstr()* shall truncate the string to *len*–1 bytes and null-
 22817 terminate the result. The application can detect that the string was truncated by comparing the
 22818 value returned by *confstr()* with *len*.

22819 If *len* is 0 and *buf* is a null pointer, then *confstr()* shall still return the integer value as defined
 22820 below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is
 22821 unspecified.

22822 After a call to:

22823 `confstr(_CS_V7_ENV, buf, sizeof(buf))`

22824 the string stored in *buf* will contain the space-separated list of variable=value environment
22825 variable pairs required by the implementation to create a conforming environment, as described
22826 in the implementations' conformance documentation.

22827 If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

22828 `confstr(_CS_PATH, buf, sizeof(buf))`

22829 can be used as a value of the *PATH* environment variable that accesses all of the standard
22830 utilities of POSIX.1-200x, if the return value is less than or equal to *sizeof(buf)*.

22831 RETURN VALUE

22832 If *name* has a configuration-defined value, *confstr()* shall return the size of buffer that would be
22833 needed to hold the entire configuration-defined value including the terminating null. If this
22834 return value is greater than *len*, the string returned in *buf* is truncated.

22835 If *name* is invalid, *confstr()* shall return 0 and set *errno* to indicate the error.

22836 If *name* does not have a configuration-defined value, *confstr()* shall return 0 and leave *errno*
22837 unchanged.

22838 ERRORS

22839 The *confstr()* function shall fail if:

22840 [EINVAL] The value of the *name* argument is invalid.

22841 EXAMPLES

22842 None.

22843 APPLICATION USAGE

22844 An application can distinguish between an invalid *name* parameter value and one that
22845 corresponds to a configurable variable that has no configuration-defined value by checking if
22846 *errno* is modified. This mirrors the behavior of *sysconf()*.

22847 The original need for this function was to provide a way of finding the configuration-defined
22848 default value for the environment variable *PATH*. Since *PATH* can be modified by the user to
22849 include directories that could contain utilities replacing the standard utilities in the Shell and
22850 Utilities volume of POSIX.1-200x, applications need a way to determine the system-supplied
22851 *PATH* environment variable value that contains the correct search path for the standard utilities.

22852 An application could use:

22853 `confstr(name, (char *)NULL, (size_t)0)`

22854 to find out how big a buffer is needed for the string value; use *malloc()* to allocate a buffer to
22855 hold the string; and call *confstr()* again to get the string. Alternately, it could allocate a fixed,
22856 static buffer that is big enough to hold most answers (perhaps 512 or 1024 bytes), but then use
22857 *malloc()* to allocate a larger buffer if it finds that this is too small.

22858 RATIONALE

22859 Application developers can normally determine any configuration variable by means of reading
22860 from the stream opened by a call to:

22861 `popen("command -p getconf variable", "r");`

22862 The *confstr()* function with a *name* argument of *_CS_PATH* returns a string that can be used as a
22863 *PATH* environment variable setting that will reference the standard shell and utilities as
22864 described in the Shell and Utilities volume of POSIX.1-200x.

22865 The *confstr()* function copies the returned string into a buffer supplied by the application instead

22866 of returning a pointer to a string. This allows a cleaner function in some implementations (such
 22867 as those with lightweight threads) and resolves questions about when the application must copy
 22868 the string returned.

22869 FUTURE DIRECTIONS

22870 None.

22871 SEE ALSO

22872 *exec*, *fpathconf()*, *sysconf()*

22873 XBD <*unistd.h*>

22874 XCU *c99*

22875 CHANGE HISTORY

22876 First released in Issue 4. Derived from the ISO POSIX-2 standard.

22877 Issue 5

22878 A table indicating the permissible values of *name* is added to the DESCRIPTION. All those
 22879 marked EX are new in this version.

22880 Issue 6

22881 The Open Group Corrigendum U033/7 is applied. The return value for the case returning the
 22882 size of the buffer now explicitly states that this includes the terminating null.

22883 The following new requirements on POSIX implementations derive from alignment with the
 22884 Single UNIX Specification:

- 22885 • The DESCRIPTION is updated with new arguments which can be used to determine
 22886 configuration strings for C compiler flags, linker/loader flags, and libraries for each
 22887 different supported programming environment. This is a change to support data size
 22888 neutrality.

22889 The following changes were made to align with the IEEE P1003.1a draft standard:

- 22890 • The DESCRIPTION is updated to include text describing how *_CS_PATH* can be used to
 22891 obtain a *PATH* to access the standard utilities.

22892 The macros associated with the *c89* programming models are marked LEGACY and new
 22893 equivalent macros associated with *c99* are introduced.

22894 Issue 7

22895 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the *_CS_V7_ENV* variable.

22896 The V6 variables for the supported programming environments are marked obsolescent.

22897 The variables for the supported programming environments are updated to be V7.

22898 The LEGACY variables and obsolescent values are removed.

22899 **NAME**
 22900 conj, conjf, conjl — complex conjugate functions

22901 **SYNOPSIS**
 22902 #include <complex.h>
 22903 double complex conj(double complex z);
 22904 float complex conjf(float complex z);
 22905 long double complex conjl(long double complex z);

22906 **DESCRIPTION**
 22907 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 22908 conflict between the requirements described here and the ISO C standard is unintentional. This
 22909 volume of POSIX.1-200x defers to the ISO C standard.

22910 These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary
 22911 part.

22912 **RETURN VALUE**
 22913 These functions return the complex conjugate value.

22914 **ERRORS**
 22915 No errors are defined.

22916 **EXAMPLES**
 22917 None.

22918 **APPLICATION USAGE**
 22919 None.

22920 **RATIONALE**
 22921 None.

22922 **FUTURE DIRECTIONS**
 22923 None.

22924 **SEE ALSO**
 22925 *carg()*, *cimag()*, *cproj()*, *creal()*

22926 XBD <complex.h>

22927 **CHANGE HISTORY**
 22928 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

22929 **NAME**

22930 connect — connect a socket

22931 **SYNOPSIS**

22932 #include <sys/socket.h>

22933 int connect(int *socket*, const struct sockaddr **address*,
22934 socklen_t *address_len*);22935 **DESCRIPTION**22936 The *connect()* function shall attempt to make a connection on a connection-mode socket or to set
22937 or reset the peer address of a connectionless-mode socket. The function takes the following
22938 arguments:

22939	<i>socket</i>	Specifies the file descriptor associated with the socket.
22940	<i>address</i>	Points to a sockaddr structure containing the peer address. The length and 22941 format of the address depend on the address family of the socket.
22942	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 22943 argument.

22944 If the socket has not already been bound to a local address, *connect()* shall bind it to an address
22945 which, unless the socket's address family is AF_UNIX, is an unused local address.22946 If the initiating socket is not connection-mode, then *connect()* shall set the socket's peer address,
22947 and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all
22948 datagrams are sent on subsequent *send()* functions, and limits the remote sender for subsequent
22949 *recv()* functions. If *address* is a null address for the protocol, the socket's peer address shall be
22950 reset. Note that despite no connection being made, the term "connected" is used to describe a
22951 connectionless-mode socket for which a peer address has been set.22952 If the initiating socket is connection-mode, then *connect()* shall attempt to establish a connection
22953 to the address specified by the *address* argument. If the connection cannot be established
22954 immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect()* shall
22955 block for up to an unspecified timeout interval until the connection is established. If the timeout
22956 interval expires before the connection is established, *connect()* shall fail and the connection
22957 attempt shall be aborted. If *connect()* is interrupted by a signal that is caught while blocked
22958 waiting to establish a connection, *connect()* shall fail and set *errno* to [EINTR], but the connection
22959 request shall not be aborted, and the connection shall be established asynchronously.22960 If the connection cannot be established immediately and O_NONBLOCK is set for the file
22961 descriptor for the socket, *connect()* shall fail and set *errno* to [EINPROGRESS], but the connection
22962 request shall not be aborted, and the connection shall be established asynchronously. Subsequent
22963 calls to *connect()* for the same socket, before the connection is established, shall fail and set *errno*
22964 to [EALREADY].22965 When the connection has been established asynchronously, *pselect()*, *select()*, and *poll()* shall
22966 indicate that the file descriptor for the socket is ready for writing.22967 The socket in use may require the process to have appropriate privileges to use the *connect()*
22968 function.22969 **RETURN VALUE**22970 Upon successful completion, *connect()* shall return 0; otherwise, -1 shall be returned and *errno*
22971 set to indicate the error.

ERRORS

- 22972
22973 The *connect()* function shall fail if:
- 22974 [EADDRNOTAVAIL]
22975 The specified address is not available from the local machine.
- 22976 [EAFNOSUPPORT]
22977 The specified address is not a valid address for the address family of the
22978 specified socket.
- 22979 [EALREADY] A connection request is already in progress for the specified socket.
- 22980 [EBADF] The *socket* argument is not a valid file descriptor.
- 22981 [ECONNREFUSED]
22982 The target address was not listening for connections or refused the connection
22983 request.
- 22984 [EINPROGRESS] O_NONBLOCK is set for the file descriptor for the socket and the connection
22985 cannot be immediately established; the connection shall be established
22986 asynchronously.
- 22987 [EINTR] The attempt to establish a connection was interrupted by delivery of a signal
22988 that was caught; the connection shall be established asynchronously.
- 22989 [EISCONN] The specified socket is connection-mode and is already connected.
- 22990 [ENETUNREACH]
22991 No route to the network is present.
- 22992 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 22993 [EPROTOTYPE] The specified address has a different type than the socket bound to the
22994 specified peer address.
- 22995 [ETIMEDOUT] The attempt to connect timed out before a connection was made.
- 22996 If the address family of the socket is AF_UNIX, then *connect()* shall fail if:
- 22997 [EIO] An I/O error occurred while reading from or writing to the file system.
- 22998 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
22999 in *address*.
- 23000 [ENAMETOOLONG]
23001 A component of a pathname exceeded {NAME_MAX} characters, or an entire
23002 pathname exceeded {PATH_MAX} characters.
- 23003 [ENOENT] A component of the pathname does not name an existing file or the pathname
23004 is an empty string.
- 23005 [ENOTDIR] A component of the path prefix of the pathname in *address* is not a directory.
- 23006 The *connect()* function may fail if:
- 23007 [EACCES] Search permission is denied for a component of the path prefix; or write access
23008 to the named socket is denied.
- 23009 [EADDRINUSE] Attempt to establish a connection that uses addresses that are already in use.
- 23010 [ECONNRESET] Remote host reset the connection request.
- 23011 [EHOSTUNREACH]
23012 The destination host cannot be reached (probably because the host is down or
23013 a remote router cannot reach it).

connect()*System Interfaces*

- 23014 [EINVAL] The *address_len* argument is not a valid length for the address family; or
23015 invalid address family in the **sockaddr** structure.
- 23016 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
23017 resolution of the pathname in *address*.
- 23018 [ENAMETOOLONG]
23019 Pathname resolution of a symbolic link produced an intermediate result
23020 whose length exceeds {PATH_MAX}.
- 23021 [ENETDOWN] The local network interface used to reach the destination is down.
- 23022 [ENOBUFS] No buffer space is available.
- 23023 [EOPNOTSUPP] The socket is listening and cannot be connected.

EXAMPLES

None.

APPLICATION USAGE

If *connect()* fails, the state of the socket is unspecified. Conforming applications should close the file descriptor and create a new socket before attempting to reconnect.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept(), *bind()*, *close()*, *getsockname()*, *poll()*, *pselect()*, *send()*, *shutdown()*, *socket()*

XBD <[sys/socket.h](#)>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #035 is applied, clarifying the description of connected sockets.

SD5-XSH-ERN-185 is applied.

23044 **NAME**
 23045 copysign, copysignf, copysignl — number manipulation function

23046 **SYNOPSIS**
 23047 #include <math.h>
 23048 double copysign(double x, double y);
 23049 float copysignf(float x, float y);
 23050 long double copysignl(long double x, long double y);

23051 **DESCRIPTION**
 23052 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23053 conflict between the requirements described here and the ISO C standard is unintentional. This
 23054 volume of POSIX.1-200x defers to the ISO C standard.

23055 These functions shall produce a value with the magnitude of x and the sign of y . On
 23056 implementations that represent a signed zero but do not treat negative zero consistently in
 23057 arithmetic operations, these functions regard the sign of zero as positive.

23058 **RETURN VALUE**
 23059 Upon successful completion, these functions shall return a value with the magnitude of x and
 23060 the sign of y .

23061 **ERRORS**
 23062 No errors are defined.

23063 **EXAMPLES**
 23064 None.

23065 **APPLICATION USAGE**
 23066 None.

23067 **RATIONALE**
 23068 None.

23069 **FUTURE DIRECTIONS**
 23070 None.

23071 **SEE ALSO**
 23072 [signbit\(\)](#)
 23073 XBD [<math.h>](#)

23074 **CHANGE HISTORY**
 23075 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23076 **NAME**

23077 cos, cosf, cosl — cosine function

23078 **SYNOPSIS**

```
23079 #include <math.h>
23080
23080 double cos(double x);
23081 float cosf(float x);
23082 long double cosl(long double x);
```

23083 **DESCRIPTION**

23084 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23085 conflict between the requirements described here and the ISO C standard is unintentional. This
 23086 volume of POSIX.1-200x defers to the ISO C standard.

23087 These functions shall compute the cosine of their argument x , measured in radians.

23088 An application wishing to check for error situations should set *errno* to zero and call
 23089 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23090 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23091 zero, an error has occurred.

23092 **RETURN VALUE**

23093 Upon successful completion, these functions shall return the cosine of x .

23094 MX If x is NaN, a NaN shall be returned.

23095 If x is ± 0 , the value 1.0 shall be returned.

23096 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 23097 defined value shall be returned.

23098 **ERRORS**

23099 These functions shall fail if:

23100 MX **Domain Error** The x argument is $\pm\text{Inf}$.

23101 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 23102 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 23103 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 23104 shall be raised.

23105 **EXAMPLES**23106 **Taking the Cosine of a 45-Degree Angle**

```
23107 #include <math.h>
23108 ...
23109 double radians = 45 * M_PI / 180;
23110 double result;
23111 ...
23112 result = cos(radians);
```

23113 **APPLICATION USAGE**

23114 These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far
 23115 from 0.

23116 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 23117 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23118
23119
23120
23121
23122
23123
23124
23125
23126
23127
23128
23129
23130
23131
23132
23133
23134
23135

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acos(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *sin()*, *tan()*

XBD Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *cosf()* and *cosl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

23136 **NAME**
 23137 cosh, coshf, coshl — hyperbolic cosine functions

23138 **SYNOPSIS**
 23139 #include <math.h>
 23140 double cosh(double x);
 23141 float coshf(float x);
 23142 long double coshl(long double x);

23143 **DESCRIPTION**
 23144 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23145 conflict between the requirements described here and the ISO C standard is unintentional. This
 23146 volume of POSIX.1-200x defers to the ISO C standard.

23147 These functions shall compute the hyperbolic cosine of their argument x .

23148 An application wishing to check for error situations should set *errno* to zero and call
 23149 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 23150 *fetetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 23151 zero, an error has occurred.

23152 **RETURN VALUE**
 23153 Upon successful completion, these functions shall return the hyperbolic cosine of x .
 23154 If the correct value would cause overflow, a range error shall occur and *cosh*(x), *coshf*(x), and
 23155 *coshl*(x) shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 23156 respectively.

23157 MX If x is NaN, a NaN shall be returned.
 23158 If x is ± 0 , the value 1.0 shall be returned.
 23159 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

23160 **ERRORS**
 23161 These functions shall fail if:
 23162 Range Error The result would cause an overflow.
 23163 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 23164 then *errno* shall be set to [ERANGE]. If the integer expression
 23165 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 23166 floating-point exception shall be raised.

23167 **EXAMPLES**
 23168 None.

23169 **APPLICATION USAGE**
 23170 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 23171 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

23172 For IEEE Std 754-1985 **double**, $710.5 < |x|$ implies that *cosh*(x) has overflowed.

23173 **RATIONALE**
 23174 None.

23175
23176

23177
23178
23179

23180
23181

23182
23183
23184

23185
23186

23187
23188

23189
23190

FUTURE DIRECTIONS

None.

SEE ALSO

acosh(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *sinh()*, *tanh()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *coshf()* and *coshl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

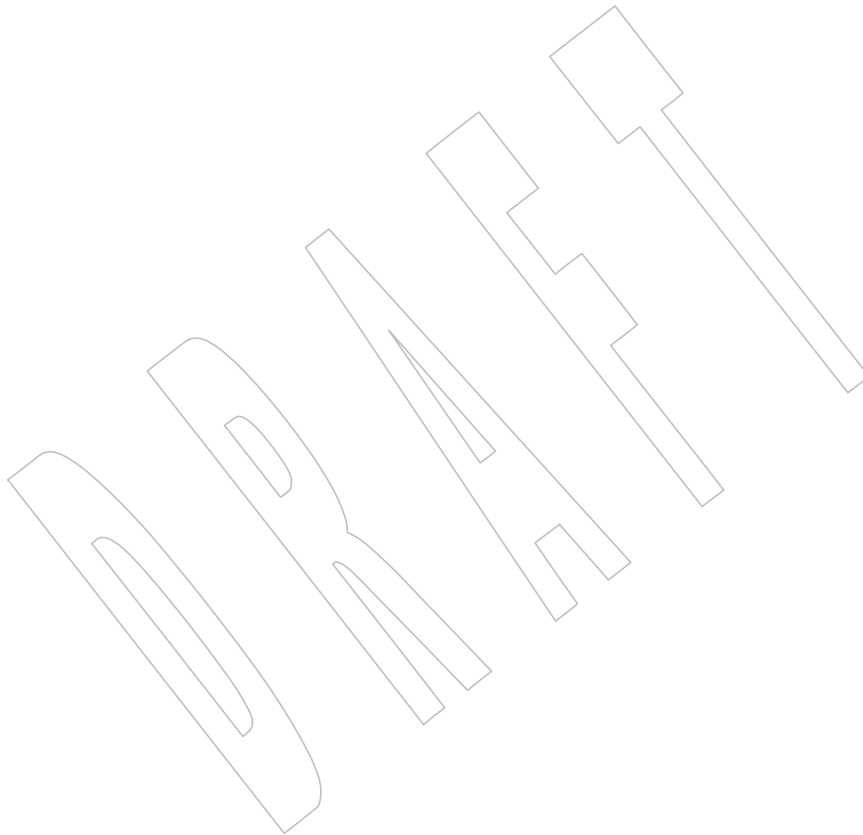
IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

DRAFT

23191 **NAME**
23192 cosl — cosine function

23193 **SYNOPSIS**
23194 #include <math.h>
23195 long double cosl(long double x);

23196 **DESCRIPTION**
23197 Refer to *cos()*.



23198

23199 **NAME**

23200 cpow, cpowf, cpowl — complex power functions

23201 **SYNOPSIS**

23202 #include <complex.h>

23203 double complex cpow(double complex x, double complex y);

23204 float complex cpowf(float complex x, float complex y);

23205 long double complex cpowl(long double complex x,

23206 long double complex y);

23207 **DESCRIPTION**

23208 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23209 conflict between the requirements described here and the ISO C standard is unintentional. This
 23210 volume of POSIX.1-200x defers to the ISO C standard.

23211 These functions shall compute the complex power function x^y , with a branch cut for the first
 23212 parameter along the negative real axis.

23213 **RETURN VALUE**

23214 These functions shall return the complex power function value.

23215 **ERRORS**

23216 No errors are defined.

23217 **EXAMPLES**

23218 None.

23219 **APPLICATION USAGE**

23220 None.

23221 **RATIONALE**

23222 None.

23223 **FUTURE DIRECTIONS**

23224 None.

23225 **SEE ALSO**23226 *cabs()*, *csqrt()*

23227 XBD <complex.h>

23228 **CHANGE HISTORY**

23229 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23230 **NAME**
 23231 cproj, cprojf, cprojl — complex projection functions

23232 **SYNOPSIS**
 23233 #include <complex.h>
 23234 double complex cproj(double complex z);
 23235 float complex cprojf(float complex z);
 23236 long double complex cprojl(long double complex z);

23237 **DESCRIPTION**
 23238 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23239 conflict between the requirements described here and the ISO C standard is unintentional. This
 23240 volume of POSIX.1-200x defers to the ISO C standard.

23241 These functions shall compute a projection of z onto the Riemann sphere: z projects to z , except
 23242 that all complex infinities (even those with one infinite part and one NaN part) project to
 23243 positive infinity on the real axis. If z has an infinite part, then $cproj(z)$ shall be equivalent to:

23244 $\text{INFINITY} + \text{I} * \text{copysign}(0.0, \text{cimag}(z))$

23245 **RETURN VALUE**
 23246 These functions shall return the value of the projection onto the Riemann sphere.

23247 **ERRORS**
 23248 No errors are defined.

23249 **EXAMPLES**
 23250 None.

23251 **APPLICATION USAGE**
 23252 None.

23253 **RATIONALE**
 23254 Two topologies are commonly used in complex mathematics: the complex plane with its
 23255 continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is
 23256 better suited for transcendental functions, the Riemann sphere for algebraic functions. The
 23257 complex types with their multiplicity of infinities provide a useful (though imperfect) model for
 23258 the complex plane. The $cproj()$ function helps model the Riemann sphere by mapping all
 23259 infinities to one, and should be used just before any operation, especially comparisons, that
 23260 might give spurious results for any of the other infinities. Note that a complex value with one
 23261 infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is
 23262 infinite, the complex value is infinite independent of the value of the other part. For the same
 23263 reason, $cabs()$ returns an infinity if its argument has an infinite part and a NaN part.

23264 **FUTURE DIRECTIONS**
 23265 None.

23266 **SEE ALSO**
 23267 [carg\(\)](#), [cimag\(\)](#), [conj\(\)](#), [creal\(\)](#)
 23268 XBD [<complex.h>](#)

23269 **CHANGE HISTORY**
 23270 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23271
23272
23273
23274
23275
23276
23277
23278
23279
23280
23281
23282
23283
23284
23285
23286
23287
23288
23289
23290
23291
23292
23293
23294
23295
23296
23297
23298
23299
23300

NAME

creal, crealf, creall — complex real functions

SYNOPSIS

```
#include <complex.h>

double creal(double complex z);
float crealf(float complex z);
long double creall(long double complex z);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

These functions shall compute the real part of *z*.

RETURN VALUE

These functions shall return the real part value.

ERRORS

No errors are defined.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of type **complex**:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg(), *cimag()*, *conj()*, *cproj()*

XBD **<complex.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

creat()

23301 **NAME**
 23302 `creat` — create a new file or rewrite an existing one

23303 **SYNOPSIS**

23304 OH `#include <sys/stat.h>`
 23305 `#include <fcntl.h>`
 23306 `int creat(const char *path, mode_t mode);`

23307 **DESCRIPTION**

23308 The `creat()` function shall behave as if it is implemented as follows:

```
23309 int creat(const char *path, mode_t mode)
23310 {
23311     return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
23312 }
```

23313 **RETURN VALUE**

23314 Refer to `open()`.

23315 **ERRORS**

23316 Refer to `open()`.

23317 **EXAMPLES**23318 **Creating a File**

23319 The following example creates the file `/tmp/file` with read and write permissions for the file
 23320 owner and read permission for group and others. The resulting file descriptor is assigned to the
 23321 `fd` variable.

```
23322 #include <fcntl.h>
23323 ...
23324 int fd;
23325 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
23326 char *filename = "/tmp/file";
23327 ...
23328 fd = creat(filename, mode);
23329 ...
```

23330 **APPLICATION USAGE**

23331 None.

23332 **RATIONALE**

23333 The `creat()` function is redundant. Its services are also provided by the `open()` function. It has
 23334 been included primarily for historical purposes since many existing applications depend on it. It
 23335 is best considered a part of the C binding rather than a function that should be provided in other
 23336 languages.

23337 **FUTURE DIRECTIONS**

23338 None.

23339 **SEE ALSO**

23340 `mknod()`, `open()`

23341 XBD `<fcntl.h>`, `<sys/stat.h>`, `<sys/types.h>`

23342

CHANGE HISTORY

23343

First released in Issue 1. Derived from Issue 1 of the SVID.

23344

Issue 6

23345

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

23346

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

23347

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

23348

23349

23350

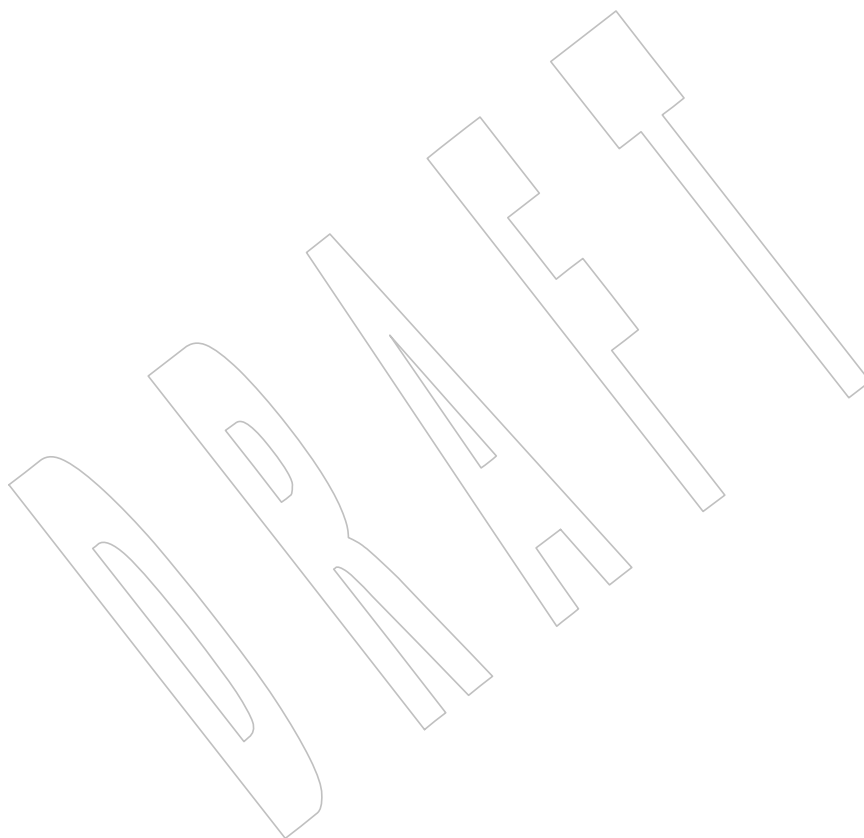
23351

Issue 7

23352

SD5-XSH-ERN-186 is applied.

+



23353 **NAME**23354 `crypt` — string encoding function (**CRYPT**)23355 **SYNOPSIS**

```
23356 XSI #include <unistd.h>
23357 char *crypt(const char *key, const char *salt);
```

23358 **DESCRIPTION**23359 The `crypt()` function is a string encoding function. The algorithm is implementation-defined.

23360 The `key` argument points to a string to be encoded. The `salt` argument shall be a string of at least
 23361 two bytes in length not including the null character chosen from the set:

```
23362 a b c d e f g h i j k l m n o p q r s t u v w x y z
23363 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
23364 0 1 2 3 4 5 6 7 8 9 . /
```

23365 The first two bytes of this string may be used to perturb the encoding algorithm.

23366 The return value of `crypt()` points to static data that is overwritten by each call.

23367 The `crypt()` function need not be thread-safe. A function that is not required to be thread-safe is
 23368 not required to be reentrant.

23369 **RETURN VALUE**

23370 Upon successful completion, `crypt()` shall return a pointer to the encoded string. The first two
 23371 bytes of the returned value shall be those of the `salt` argument. Otherwise, it shall return a null
 23372 pointer and set `errno` to indicate the error.

23373 **ERRORS**23374 The `crypt()` function shall fail if:

23375 [ENOSYS] The functionality is not supported on this implementation.

23376 **EXAMPLES**23377 **Encoding Passwords**

23378 The following example finds a user database entry matching a particular user name and changes
 23379 the current password to a new password. The `crypt()` function generates an encoded version of
 23380 each password. The first call to `crypt()` produces an encoded version of the old password; that
 23381 encoded password is then compared to the password stored in the user database. The second
 23382 call to `crypt()` encodes the new password before it is stored.

23383 The `putpwent()` function, used in the following example, is not part of POSIX.1-200x.

```
23384 #include <unistd.h>
23385 #include <pwd.h>
23386 #include <string.h>
23387 #include <stdio.h>
23388 ...
23389 int valid_change;
23390 int pfd; /* Integer for file descriptor returned by open(). */
23391 FILE *fpfd; /* File pointer for use in putpwent(). */
23392 struct passwd *p;
23393 char user[100];
23394 char oldpasswd[100];
```

```

23395     char newpasswd[100];
23396     char savepasswd[100];
23397     ...
23398     valid_change = 0;
23399     while ((p = getpwent()) != NULL) {
23400         /* Change entry if found. */
23401         if (strcmp(p->pw_name, user) == 0) {
23402             if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
23403                 strcpy(savepasswd, crypt(newpasswd, user));
23404                 p->pw_passwd = savepasswd;
23405                 valid_change = 1;
23406             }
23407             else {
23408                 fprintf(stderr, "Old password is not valid\n");
23409             }
23410         }
23411         /* Put passwd entry into tmp. */
23412         putpwent(p, fpfd);
23413     }

```

APPLICATION USAGE

The values returned by this function need not be portable among XSI-conformant systems.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

encrypt(), *setkey()*

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 7

SD5-XSH-ERN-178 is applied.

23430 **NAME**

23431 csin, csinf, csinl — complex sine functions

23432 **SYNOPSIS**

23433 #include <complex.h>

23434 double complex csin(double complex z);

23435 float complex csinf(float complex z);

23436 long double complex csinl(long double complex z);

23437 **DESCRIPTION**

23438 CX The functionality described on this reference page is aligned with the ISO C standard. Any

23439 conflict between the requirements described here and the ISO C standard is unintentional. This

23440 volume of POSIX.1-200x defers to the ISO C standard.

23441 These functions shall compute the complex sine of z.

23442 **RETURN VALUE**

23443 These functions shall return the complex sine value.

23444 **ERRORS**

23445 No errors are defined.

23446 **EXAMPLES**

23447 None.

23448 **APPLICATION USAGE**

23449 None.

23450 **RATIONALE**

23451 None.

23452 **FUTURE DIRECTIONS**

23453 None.

23454 **SEE ALSO**

23455 *casin()*

23456 XBD <complex.h>

23457 **CHANGE HISTORY**

23458 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23459 **NAME**
 23460 csinh, csinhf, csinhl — complex hyperbolic sine functions

23461 **SYNOPSIS**
 23462 #include <complex.h>

23463 double complex csinh(double complex z);
 23464 float complex csinhf(float complex z);
 23465 long double complex csinhl(long double complex z);

23466 **DESCRIPTION**
 23467 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23468 conflict between the requirements described here and the ISO C standard is unintentional. This
 23469 volume of POSIX.1-200x defers to the ISO C standard.

23470 These functions shall compute the complex hyperbolic sine of *z*.

23471 **RETURN VALUE**
 23472 These functions shall return the complex hyperbolic sine value.

23473 **ERRORS**
 23474 No errors are defined.

23475 **EXAMPLES**
 23476 None.

23477 **APPLICATION USAGE**
 23478 None.

23479 **RATIONALE**
 23480 None.

23481 **FUTURE DIRECTIONS**
 23482 None.

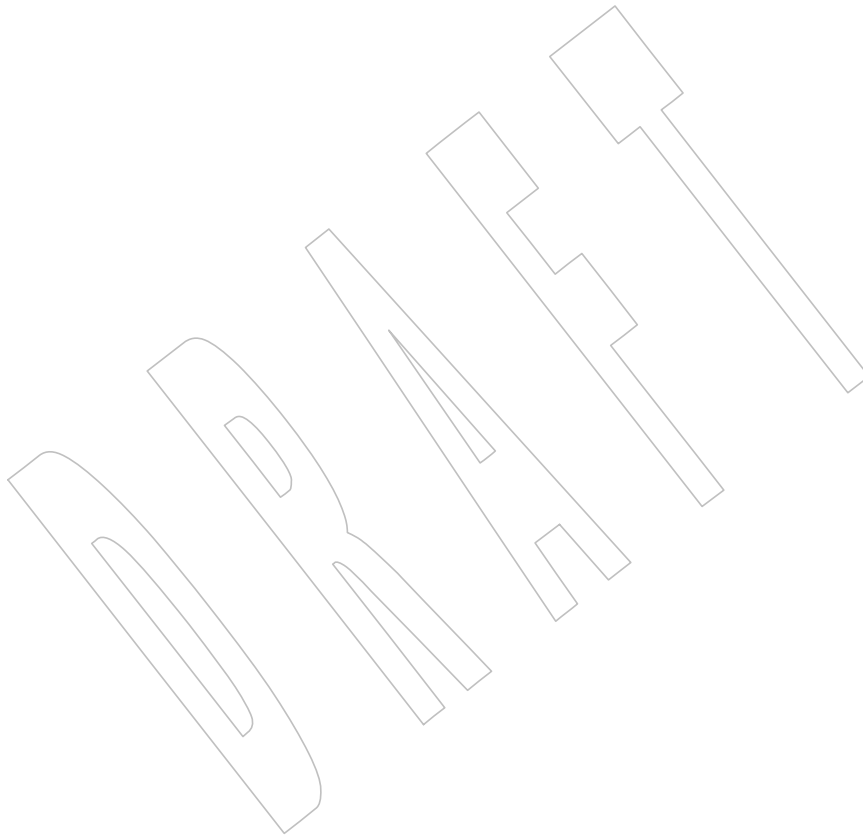
23483 **SEE ALSO**
 23484 *casinh()*
 23485 XBD <complex.h>

23486 **CHANGE HISTORY**
 23487 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23488 **NAME**
23489 `csinl` — complex sine functions

23490 **SYNOPSIS**
23491 `#include <complex.h>`
23492 `long double complex csinl(long double complex z);`

23493 **DESCRIPTION**
23494 Refer to *csin()*.



23495 **NAME**
 23496 csqrt, csqrtf, csqrtl — complex square root functions

23497 **SYNOPSIS**
 23498 #include <complex.h>
 23499 double complex csqrt(double complex z);
 23500 float complex csqrtf(float complex z);
 23501 long double complex csqrtl(long double complex z);

23502 **DESCRIPTION**
 23503 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23504 conflict between the requirements described here and the ISO C standard is unintentional. This
 23505 volume of POSIX.1-200x defers to the ISO C standard.

23506 These functions shall compute the complex square root of z , with a branch cut along the negative
 23507 real axis.

23508 **RETURN VALUE**
 23509 These functions shall return the complex square root value, in the range of the right half-plane
 23510 (including the imaginary axis).

23511 **ERRORS**
 23512 No errors are defined.

23513 **EXAMPLES**
 23514 None.

23515 **APPLICATION USAGE**
 23516 None.

23517 **RATIONALE**
 23518 None.

23519 **FUTURE DIRECTIONS**
 23520 None.

23521 **SEE ALSO**
 23522 *cabs()*, *cpow()*
 23523 XBD <complex.h>

23524 **CHANGE HISTORY**
 23525 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23526 **NAME**
 23527 `ctan, ctanf, ctanl` — complex tangent functions

23528 **SYNOPSIS**
 23529 `#include <complex.h>`

23530 `double complex ctan(double complex z);`
 23531 `float complex ctanf(float complex z);`
 23532 `long double complex ctanl(long double complex z);`

23533 **DESCRIPTION**
 23534 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23535 conflict between the requirements described here and the ISO C standard is unintentional. This
 23536 volume of POSIX.1-200x defers to the ISO C standard.

23537 These functions shall compute the complex tangent of z .

23538 **RETURN VALUE**
 23539 These functions shall return the complex tangent value.

23540 **ERRORS**
 23541 No errors are defined.

23542 **EXAMPLES**
 23543 None.

23544 **APPLICATION USAGE**
 23545 None.

23546 **RATIONALE**
 23547 None.

23548 **FUTURE DIRECTIONS**
 23549 None.

23550 **SEE ALSO**
 23551 [*catan\(\)*](#)
 23552 XBD [<complex.h>](#)

23553 **CHANGE HISTORY**
 23554 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23555 **NAME**
 23556 ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

23557 **SYNOPSIS**
 23558 #include <complex.h>
 23559 double complex ctanh(double complex z);
 23560 float complex ctanhf(float complex z);
 23561 long double complex ctanhl(long double complex z);

23562 **DESCRIPTION**
 23563 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 23564 conflict between the requirements described here and the ISO C standard is unintentional. This
 23565 volume of POSIX.1-200x defers to the ISO C standard.

23566 These functions shall compute the complex hyperbolic tangent of z .

23567 **RETURN VALUE**
 23568 These functions shall return the complex hyperbolic tangent value.

23569 **ERRORS**
 23570 No errors are defined.

23571 **EXAMPLES**
 23572 None.

23573 **APPLICATION USAGE**
 23574 None.

23575 **RATIONALE**
 23576 None.

23577 **FUTURE DIRECTIONS**
 23578 None.

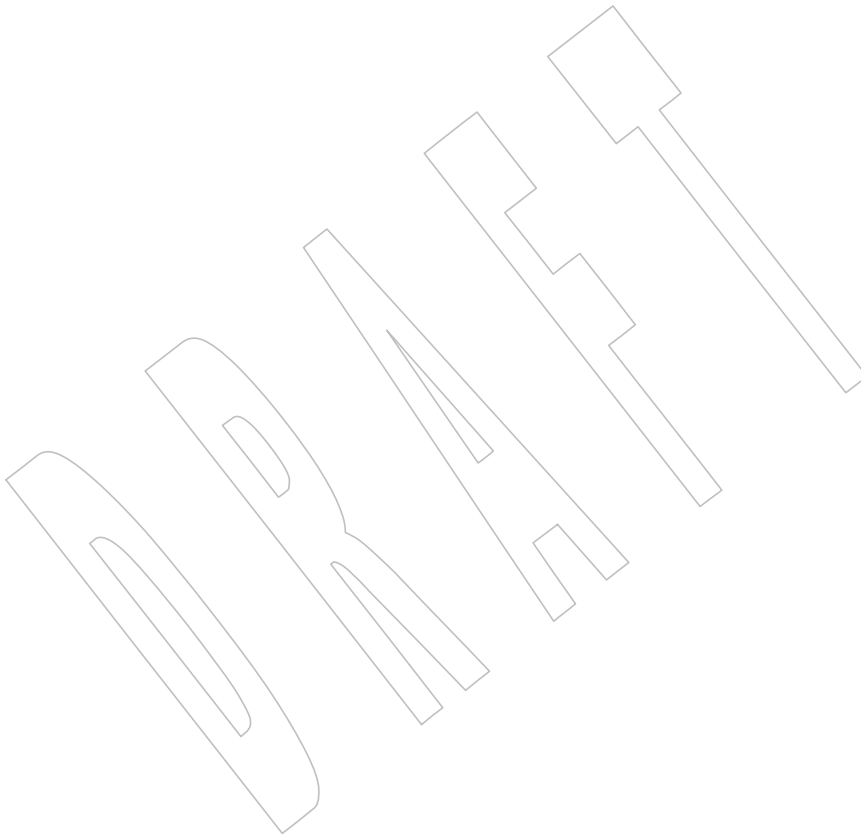
23579 **SEE ALSO**
 23580 [catanh\(\)](#)
 23581 XBD [<complex.h>](#)

23582 **CHANGE HISTORY**
 23583 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

23584 **NAME**
23585 ctanl — complex tangent functions

23586 **SYNOPSIS**
23587 #include <complex.h>
23588 long double complex ctanl(long double complex z);

23589 **DESCRIPTION**
23590 Refer to *ctan()*.



23591 **NAME**
 23592 `ctermid` — generate a pathname for the controlling terminal

23593 **SYNOPSIS**

23594 CX

```
#include <stdio.h>
```


 23595

```
char *ctermid(char *s);
```

23596 **DESCRIPTION**

23597 The `ctermid()` function shall generate a string that, when used as a pathname, refers to the
 23598 current controlling terminal for the current process. If `ctermid()` returns a pathname, access to the
 23599 file is not guaranteed.

23600 If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
 23601 functions, it shall ensure that the `ctermid()` function is called with a non-NULL parameter.

23602 **RETURN VALUE**

23603 If `s` is a null pointer, the string shall be generated in an area that may be static (and therefore may
 23604 be overwritten by each call), the address of which shall be returned. Otherwise, `s` is assumed to
 23605 point to a character array of at least `L_ctermid` bytes; the string is placed in this array and the
 23606 value of `s` shall be returned. The symbolic constant `L_ctermid` is defined in `<stdio.h>`, and shall
 23607 have a value greater than 0.

23608 The `ctermid()` function shall return an empty string if the pathname that would refer to the
 23609 controlling terminal cannot be determined, or if the function is unsuccessful.

23610 **ERRORS**

23611 No errors are defined.

23612 **EXAMPLES**

23613 **Determining the Controlling Terminal for the Current Process**

23614 The following example returns a pointer to a string that identifies the controlling terminal for the
 23615 current process. The pathname for the terminal is stored in the array pointed to by the `ptr`
 23616 argument, which has a size of `L_ctermid` bytes, as indicated by the `term` argument.

23617

```
#include <stdio.h>
```


 23618

```
...
```


 23619

```
char term[L_ctermid];
```


 23620

```
char *ptr;
```


 23621

```
ptr = ctermid(term);
```

23622 **APPLICATION USAGE**

23623 The difference between `ctermid()` and `ttyname()` is that `ttyname()` must be handed a file
 23624 descriptor and return a path of the terminal associated with that file descriptor, while `ctermid()`
 23625 returns a string (such as `"/dev/tty"`) that refers to the current controlling terminal if used as a
 23626 pathname.

23627 **RATIONALE**

23628 `L_ctermid` must be defined appropriately for a given implementation and must be greater than
 23629 zero so that array declarations using it are accepted by the compiler. The value includes the
 23630 terminating null byte.

23631 Conforming applications that use threads cannot call `ctermid()` with NULL as the parameter if
 23632 either `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS` is defined. If `s` is not
 23633 NULL, the `ctermid()` function generates a string that, when used as a pathname, refers to the

23634 current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid()* is
23635 undefined.

23636 There is no additional burden on the programmer—changing to use a hypothetical thread-safe
23637 version of *ctermid()* along with allocating a buffer is more of a burden than merely allocating a
23638 buffer. Application code should not assume that the returned string is short, as some
23639 implementations have more than two pathname components before reaching a logical device
23640 name.

23641 **FUTURE DIRECTIONS**

23642 None.

23643 **SEE ALSO**

23644 *ttyname()*

23645 XBD <stdio.h>

23646 **CHANGE HISTORY**

23647 First released in Issue 1. Derived from Issue 1 of the SVID.

23648 **Issue 5**

23649 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

23650 **Issue 6**

23651 The normative text is updated to avoid use of the term “must” for application requirements.

23652 **NAME**

23653 ctime, ctime_r — convert a time value to a date and time string

23654 **SYNOPSIS**

```
23655 OB #include <time.h>
23656 char *ctime(const time_t *clock);
23657 OB CX char *ctime_r(const time_t *clock, char *buf);
```

23658 **DESCRIPTION**

23659 CX For *ctime()*: The functionality described on this reference page is aligned with the ISO C
 23660 standard. Any conflict between the requirements described here and the ISO C standard is
 23661 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

23662 The *ctime()* function shall convert the time pointed to by *clock*, representing time in seconds
 23663 since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
23664 asctime(localtime(clock))
```

23665 CX The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
 23666 objects: a broken-down time structure and an array of **char**. Execution of any of the functions
 23667 may overwrite the information returned in either of these objects by any of the other functions.

23668 The *ctime()* function need not be thread-safe. A function that is not required to be thread-safe is
 23669 not required to be reentrant.

23670 The *ctime_r()* function shall convert the calendar time pointed to by *clock* to local time in exactly
 23671 the same form as *ctime()* and put the string into the array pointed to by *buf* (which shall be at
 23672 least 26 bytes in size) and return *buf*.

23673 Unlike *ctime()*, the thread-safe version *ctime_r()* is not required to set *tzname*.

23674 **RETURN VALUE**

23675 The *ctime()* function shall return the pointer returned by *asctime()* with that broken-down time
 23676 as an argument.

23677 CX Upon successful completion, *ctime_r()* shall return a pointer to the string pointed to by *buf*.
 23678 When an error is encountered, a null pointer shall be returned.

23679 **ERRORS**

23680 No errors are defined.

23681 **EXAMPLES**

23682 None.

23683 **APPLICATION USAGE**

23684 These functions are included only for compatibility with older implementations. They have
 23685 undefined behavior if the resulting string would be too long, so the use of these functions
 23686 should be discouraged. On implementations that do not detect output string length overflow, it
 23687 is possible to overflow the output buffers in such a way as to cause applications to fail, or
 23688 possible system security violations. Also, these functions do not support localized date and time
 23689 formats. To avoid these problems, applications should use *strftime()* to generate strings from
 23690 broken-down times.

23691 Values for the broken-down time structure can be obtained by calling *gmtime()* or *localtime()*.

23692 The *ctime_r()* function is thread-safe and shall return values in a user-supplied buffer instead of
 23693 possibly using a static data area that may be overwritten by each call.

23694 Attempts to use *ctime()* or *ctime_r()* for times before the Epoch or for times beyond the year 9999
 23695 produce undefined results. Refer to *asctime()* (on page 567).

RATIONALE

23696 The standard developers decided to mark the *ctime()* and *ctime_r()* functions obsolescent even
 23697 though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C
 23698 standard also provides the *strptime()* function which can be used to avoid these problems.
 23699

FUTURE DIRECTIONS

23700 These functions may be removed in a future version.
 23701

SEE ALSO

23702 *asctime()*, *clock()*, *difftime()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *strptime()*, *time*, *utime()*
 23703
 23704 XBD <**time.h**>

CHANGE HISTORY

23705 First released in Issue 1. Derived from Issue 1 of the SVID.
 23706

Issue 5

23707 Normative text previously in the APPLICATION USAGE section is moved to the
 23708 DESCRIPTION.
 23709

23710 The *ctime_r()* function is included for alignment with the POSIX Threads Extension.

23711 A note indicating that the *ctime()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

23712 Extensions beyond the ISO C standard are marked.
 23713

23714 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

23715 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 23716 its avoidance of possibly using a static data area.

Issue 7

23717 SD5-XSH-ERN-25 is applied, updating the APPLICATION USAGE.

23718 Austin Group Interpretation 1003.1-2001 #053 is applied, marking these functions obsolescent.

23719 The *ctime_r()* function is moved from the Thread-Safe Functions option to the Base.
 23720

23721 **NAME**
23722 daylight — daylight savings time flag

23723 **SYNOPSIS**

```
23724 XSI #include <time.h>  
23725 extern int daylight;
```

23726 **DESCRIPTION**

23727 Refer to [tzset\(\)](#).

23728 NAME

23729 dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey,
23730 dbm_open, dbm_store — database functions

23731 SYNOPSIS

```
23732 XSI #include <ndbm.h>
23733
23734 int dbm_clearerr(DBM *db);
23735 void dbm_close(DBM *db);
23736 int dbm_delete(DBM *db, datum key);
23737 int dbm_error(DBM *db);
23738 datum dbm_fetch(DBM *db, datum key);
23739 datum dbm_firstkey(DBM *db);
23740 datum dbm_nextkey(DBM *db);
23741 DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
23742 int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

23742 DESCRIPTION

23743 These functions create, access, and modify a database.

23744 A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object
23745 that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in
23746 the object pointed to by *dptr*.

23747 A database shall be stored in one or two files. When one file is used, the name of the database
23748 file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm_open()*. When
23749 two files are used, the names of the database files shall be formed by appending the suffixes **.dir**
23750 and **.pag** respectively to the *file* argument.

23751 The *dbm_open()* function shall open a database. The *file* argument to the function is the
23752 pathname of the database. The *open_flags* argument has the same meaning as the *flags* argument
23753 of *open()* except that a database opened for write-only access opens the files for read and write
23754 access and the behavior of the **O_APPEND** flag is unspecified. The *file_mode* argument has the
23755 same meaning as the third argument of *open()*.

23756 The *dbm_open()* function need not accept pathnames longer than **{PATH_MAX}-4** bytes
23757 (including the terminating null), or pathnames with a last component longer than
23758 **{NAME_MAX}-4** bytes (excluding the terminating null).

23759 The *dbm_close()* function shall close a database. The application shall ensure that argument *db* is
23760 a pointer to a **dbm** structure that has been returned from a call to *dbm_open()*.

23761 These database functions shall support an internal block size large enough to support
23762 key/content pairs of at least 1 023 bytes.

23763 The *dbm_fetch()* function shall read a record from a database. The argument *db* is a pointer to a
23764 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
23765 **datum** that has been initialized by the application to the value of the key that matches the key of
23766 the record the program is fetching.

23767 The *dbm_store()* function shall write a record to a database. The argument *db* is a pointer to a
23768 database structure that has been returned from a call to *dbm_open()*. The argument *key* is a
23769 **datum** that has been initialized by the application to the value of the key that identifies (for
23770 subsequent reading, writing, or deleting) the record the application is writing. The argument
23771 *content* is a **datum** that has been initialized by the application to the value of the record the
23772 program is writing. The argument *store_mode* controls whether *dbm_store()* replaces any pre-

23773 existing record that has the same key that is specified by the *key* argument. The application shall
 23774 set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that
 23775 matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be
 23776 replaced with the new record. If the database contains a record that matches the *key* argument
 23777 and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record
 23778 ignored. If the database does not contain a record that matches the *key* argument and *store_mode*
 23779 is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

23780 If the sum of a key/content pair exceeds the internal block size, the result is unspecified.
 23781 Moreover, the application shall ensure that all key/content pairs that hash together fit on a
 23782 single block. The *dbm_store()* function shall return an error in the event that a disk block fills
 23783 with inseparable data.

23784 The *dbm_delete()* function shall delete a record and its key from the database. The argument *db* is
 23785 a pointer to a database structure that has been returned from a call to *dbm_open()*. The argument
 23786 *key* is a **datum** that has been initialized by the application to the value of the key that identifies
 23787 the record the program is deleting.

23788 The *dbm_firstkey()* function shall return the first key in the database. The argument *db* is a
 23789 pointer to a database structure that has been returned from a call to *dbm_open()*.

23790 The *dbm_nextkey()* function shall return the next key in the database. The argument *db* is a
 23791 pointer to a database structure that has been returned from a call to *dbm_open()*. The application
 23792 shall ensure that the *dbm_firstkey()* function is called before calling *dbm_nextkey()*. Subsequent
 23793 calls to *dbm_nextkey()* return the next key until all of the keys in the database have been
 23794 returned.

23795 The *dbm_error()* function shall return the error condition of the database. The argument *db* is a
 23796 pointer to a database structure that has been returned from a call to *dbm_open()*.

23797 The *dbm_clearerr()* function shall clear the error condition of the database. The argument *db* is a
 23798 pointer to a database structure that has been returned from a call to *dbm_open()*.

23799 The *dptr* pointers returned by these functions may point into static storage that may be changed
 23800 by subsequent calls.

23801 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 23802 required to be reentrant.

23803 RETURN VALUE

23804 The *dbm_store()* and *dbm_delete()* functions shall return 0 when they succeed and a negative
 23805 value when they fail.

23806 The *dbm_store()* function shall return 1 if it is called with a *flags* value of DBM_INSERT and the
 23807 function finds an existing record with the same key.

23808 The *dbm_error()* function shall return 0 if the error condition is not set and return a non-zero
 23809 value if the error condition is set.

23810 The return value of *dbm_clearerr()* is unspecified.

23811 The *dbm_firstkey()* and *dbm_nextkey()* functions shall return a key **datum**. When the end of the
 23812 database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr*
 23813 member of the key shall be a null pointer and the error condition of the database shall be set.

23814 The *dbm_fetch()* function shall return a content **datum**. If no record in the database matches the
 23815 key or if an error condition has been detected in the database, the *dptr* member of the content
 23816 shall be a null pointer.

23817 The *dbm_open()* function shall return a pointer to a database structure. If an error is detected
 23818 during the operation, *dbm_open()* shall return a (DBM *)0.

23819 **ERRORS**

23820 No errors are defined.

23821 **EXAMPLES**

23822 None.

23823 **APPLICATION USAGE**

23824 The following code can be used to traverse the database:

23825

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

23826 The *dbm_** functions provided in this library should not be confused in any way with those of a
 23827 general-purpose database management system. These functions do not provide for multiple
 23828 search keys per entry, they do not protect against multi-user access (in other words they do not
 23829 lock records or files), and they do not provide the many other useful database functions that are
 23830 found in more robust database management systems. Creating and updating databases by use of
 23831 these functions is relatively slow because of data copies that occur upon hash collisions. These
 23832 functions are useful for applications requiring fast lookup of relatively static information that is
 23833 to be indexed by a single key.

23834 Note that a strictly conforming application is extremely limited by these functions: since there is
 23835 no way to determine that the keys in use do not all hash to the same value (although that would
 23836 be rare), a strictly conforming application cannot be guaranteed that it can store more than one
 23837 block's worth of data in the database. As long as a key collision does not occur, additional data
 23838 may be stored, but because there is no way to determine whether an error is due to a key
 23839 collision or some other error condition (*dbm_error()* being effectively a Boolean), once an error is
 23840 detected, the application is effectively limited to guessing what the error might be if it wishes to
 23841 continue using these functions.

23842 The *dbm_delete()* function need not physically reclaim file space, although it does make it
 23843 available for reuse by the database.

23844 After calling *dbm_store()* or *dbm_delete()* during a pass through the keys by *dbm_firstkey()* and
 23845 *dbm_nextkey()*, the application should reset the database by calling *dbm_firstkey()* before again
 23846 calling *dbm_nextkey()*. The contents of these files are unspecified and may not be portable.

23847 Applications should take care that database pathname arguments specified to *dbm_open()* are
 23848 not prefixes of unrelated files. This might be done, for example, by placing databases in a
 23849 separate directory.

23850 Since some implementations use three characters for a suffix and others use four characters for a
 23851 suffix, applications should ensure that the maximum portable pathname length passed to
 23852 *dbm_open()* is no greater than {PATH_MAX}-4 bytes, with the last component of the pathname
 23853 no greater than {NAME_MAX}-4 bytes.

23854 **RATIONALE**

23855 Previously the standard required the database to be stored in two files, one file being a directory
 23856 containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and
 23857 having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow
 23858 newer implementations of the Berkeley DB interface using a single file that have evolved while
 23859 remaining compatible with the application programming interface. The standard developers
 23860 considered removing the specific suffixes altogether but decided to retain them so as not to
 23861 pollute the application file name space more than necessary and to allow for portable backups of
 23862 the database.

23863 **FUTURE DIRECTIONS**

23864 None.

23865

SEE ALSO

23866

open()

23867

XBD <ndbm.h>

23868

CHANGE HISTORY

23869

First released in Issue 4, Version 2.

23870

Issue 5

23871

Moved from X/OPEN UNIX extension to BASE.

23872

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

23873

23874

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

23875

Issue 6

23876

The normative text is updated to avoid use of the term “must” for application requirements.

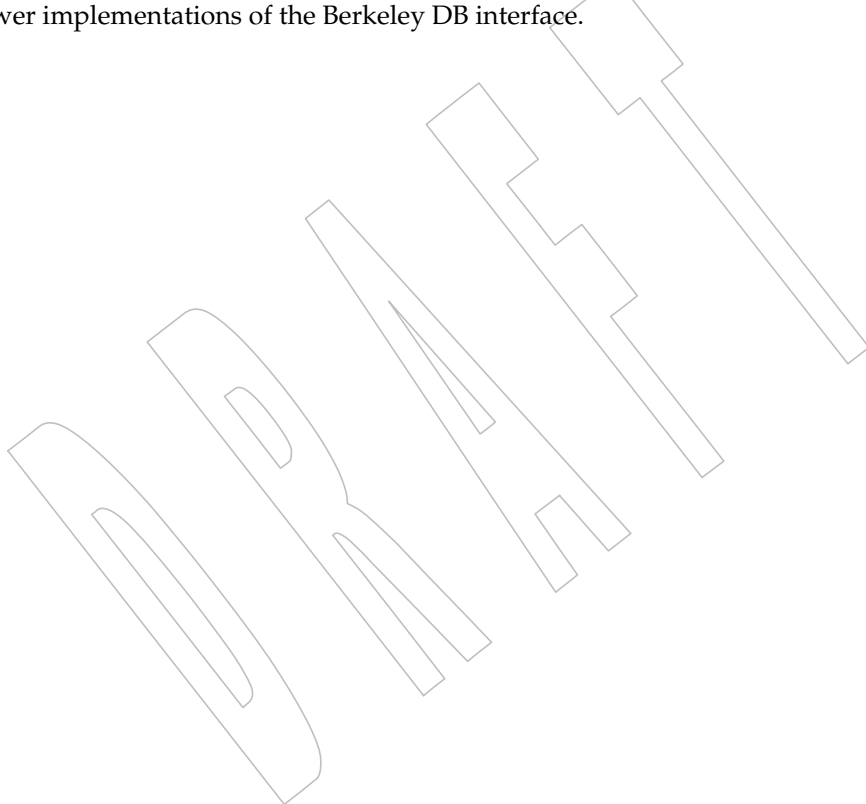
23877

Issue 7

23878

Austin Group Interpretation 1003.1-2001 #042 is applied so that the DESCRIPTION permits newer implementations of the Berkeley DB interface.

23879



difftime()23880 **NAME**23881 `difftime` — compute the difference between two calendar time values23882 **SYNOPSIS**23883 `#include <time.h>`23884 `double difftime(time_t time1, time_t time0);`23885 **DESCRIPTION**23886 CX The functionality described on this reference page is aligned with the ISO C standard. Any
23887 conflict between the requirements described here and the ISO C standard is unintentional. This
23888 volume of POSIX.1-200x defers to the ISO C standard.23889 The *difftime()* function shall compute the difference between two calendar times (as returned by
23890 *time()*): *time1*– *time0*.23891 **RETURN VALUE**23892 The *difftime()* function shall return the difference expressed in seconds as a type **double**.23893 **ERRORS**

23894 No errors are defined.

23895 **EXAMPLES**

23896 None.

23897 **APPLICATION USAGE**

23898 None.

23899 **RATIONALE**

23900 None.

23901 **FUTURE DIRECTIONS**

23902 None.

23903 **SEE ALSO**23904 *asctime()*, *clock()*, *ctime()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time*, *utime()*23905 XBD **<time.h>**23906 **CHANGE HISTORY**

23907 First released in Issue 4. Derived from the ISO C standard.

23908 **NAME**
 23909 dirfd — extract the file descriptor used by a DIR stream

23910 **SYNOPSIS**
 23911 #include <dirent.h>
 23912 int dirfd(DIR *dirp);

23913 **DESCRIPTION**
 23914 The *dirfd()* function shall return a file descriptor referring to the same directory as the *dirp*
 23915 argument. This file descriptor shall be closed by a call to *closedir()*. The behavior of future calls
 23916 to *readdir()* and *readdir_r()* is undefined if the application attempts to alter the file position
 23917 indicator using the returned file descriptor. The behavior of future calls to *closedir()*, *readdir()*,
 23918 and *readdir_r()* is undefined if the application attempts to close the file descriptor.

23919 **RETURN VALUE**
 23920 Upon successful completion, the *dirfd()* function shall return an integer which contains a file
 23921 descriptor for the stream pointed to by *dirp*. Otherwise, it shall return -1 and may set *errno* to
 23922 indicate the error.

23923 **ERRORS**
 23924 The *dirfd()* function may fail if:
 23925 [EINVAL] The *dirp* argument does not refer to a valid directory stream.
 23926 [ENOTSUP] The implementation does not support the association of a file descriptor with
 23927 a directory.

23928 **EXAMPLES**
 23929 None.

23930 **APPLICATION USAGE**
 23931 The *dirfd()* function is intended to be a mechanism by which an application may obtain a file
 23932 descriptor to use for the *fchdir()* function.

23933 **RATIONALE**
 23934 This interface was introduced because the Base Definitions volume of POSIX.1-200x does not
 23935 make public the **DIR** data structure. Applications tend to use the *fchdir()* function on the file
 23936 descriptor returned by this interface, and this has proven useful for security reasons; in
 23937 particular, it is a better technique than others where directory names might change.

23938 The description uses the term “a file descriptor” rather than “the file descriptor”. The
 23939 implication intended is that an implementation that does not use an *fd* for *diropen()* could still
 23940 *open()* the directory to implement the *dirfd()* function. Such a descriptor must be closed later
 23941 during a call to *closedir()*.

23942 An implementation that does not support file descriptors referring to directories may fail with
 23943 [ENOTSUP].

23944 If it is necessary to allocate an *fd* to be returned by *dirfd()*, it should be done at the time of a call
 23945 to *opendir()*.

23946 **FUTURE DIRECTIONS**
 23947 None.

dirfd()

23948

SEE ALSO

23949

closedir(), *fchdir()*, *fdopendir()*, *fileno()*, *open()*, *readdir()*

23950

XBD <**dirent.h**>

23951

CHANGE HISTORY

23952

First released in Issue 7.



23953 **NAME**23954 `dirname` — report the parent directory name of a file pathname23955 **SYNOPSIS**

```
23956 XSI #include <libgen.h>
23957 char *dirname(char *path);
```

23958 **DESCRIPTION**

23959 The `dirname()` function shall take a pointer to a character string that contains a pathname, and
 23960 return a pointer to a string that is a pathname of the parent directory of that file. Trailing '/'
 23961 characters in the path are not counted as part of the path.

23962 If *path* does not contain a '/', then `dirname()` shall return a pointer to the string ".". If *path* is a
 23963 null pointer or points to an empty string, `dirname()` shall return a pointer to the string ".".

23964 The `dirname()` function need not be thread-safe. A function that is not required to be thread-safe
 23965 is not required to be reentrant.

23966 **RETURN VALUE**

23967 The `dirname()` function shall return a pointer to a string that is the parent directory of *path*. If
 23968 *path* is a null pointer or points to an empty string, a pointer to a string "." is returned.

23969 The `dirname()` function may modify the string pointed to by *path*, and may return a pointer to
 23970 static storage that may then be overwritten by subsequent calls to `dirname()`.

23971 **ERRORS**

23972 No errors are defined.

23973 **EXAMPLES**

23974 The following code fragment reads a pathname, changes the current working directory to the
 23975 parent directory, and opens the file.

```
23976 char *path = NULL, *pathcopy;
23977 size_t buflen = 0;
23978 ssize_t linelen = 0;
23979 int fd;
23980
23981 linelen = getline(&path, &buflen, stdin);
23982
23983 path[linelen-1] = 0;
23984 pathcopy = strdup(path);
23985 if (chdir(dirname(pathcopy)) < 0) {
23986     ...
23987 }
23988 if ((fd = open(basename(path), O_RDONLY)) >= 0) {
23989     ...
23990     close (fd);
23991 }
23992 ...
23993 free (pathcopy);
23994 free (path);
```

23993

Sample Input and Output Strings for dirname()

23994

23995

In the following table, the input string is the value pointed to by *path*, and the output string is the return value of the *dirname()* function.

23996

23997

23998

23999

24000

24001

24002

Input String	Output String
"/usr/lib"	"/usr"
"/usr/"	"/"
"usr"	."
"/"	"/"
."	."
".."	."

24003

APPLICATION USAGE

24004

24005

The *dirname()* and *basename()* functions together yield a complete pathname. The expression *dirname(path)* obtains the pathname of the directory where *basename(path)* is found.

24006

24007

Since the meaning of the leading *"/"* is implementation-defined, *dirname("/foo)* may return either *"/"* or *'/'* (but nothing else).

24008

RATIONALE

24009

None.

24010

FUTURE DIRECTIONS

24011

None.

24012

SEE ALSO

24013

basename

24014

XBD <*libgen.h*>

24015

CHANGE HISTORY

24016

First released in Issue 4, Version 2.

24017

Issue 5

24018

Moved from X/OPEN UNIX extension to BASE.

24019

24020

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

24021

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

24022

Issue 7

24023

The EXAMPLES section is revised.

24024 **NAME**
 24025 `div` — compute the quotient and remainder of an integer division

24026 **SYNOPSIS**
 24027 `#include <stdlib.h>`
 24028 `div_t div(int numer, int denom);`

24029 **DESCRIPTION**
 24030 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 24031 conflict between the requirements described here and the ISO C standard is unintentional. This
 24032 volume of POSIX.1-200x defers to the ISO C standard.

24033 The `div()` function shall compute the quotient and remainder of the division of the numerator
 24034 `numer` by the denominator `denom`. If the division is inexact, the resulting quotient is the integer
 24035 of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be
 24036 represented, the behavior is undefined; otherwise, $quot * denom + rem$ shall equal `numer`.

24037 **RETURN VALUE**
 24038 The `div()` function shall return a structure of type `div_t`, comprising both the quotient and the
 24039 remainder. The structure includes the following members, in any order:

24040 `int quot; /* quotient */`
 24041 `int rem; /* remainder */`

24042 **ERRORS**
 24043 No errors are defined.

24044 **EXAMPLES**
 24045 None.

24046 **APPLICATION USAGE**
 24047 None.

24048 **RATIONALE**
 24049 None.

24050 **FUTURE DIRECTIONS**
 24051 None.

24052 **SEE ALSO**
 24053 [ldiv\(\)](#)
 24054 XBD [<stdlib.h>](#)

24055 **CHANGE HISTORY**
 24056 First released in Issue 4. Derived from the ISO C standard.

24057 **NAME**24058 `dlclose` — close a `dlopen()` object24059 **SYNOPSIS**24060 `#include <dlfcn.h>`24061 `int dlclose(void *handle);`24062 **DESCRIPTION**24063 The `dlclose()` function shall inform the system that the object referenced by a `handle` returned
24064 from a previous `dlopen()` invocation is no longer needed by the application.24065 The use of `dlclose()` reflects a statement of intent on the part of the process, but does not create
24066 any requirement upon the implementation, such as removal of the code or symbols referenced
24067 by `handle`. Once an object has been closed using `dlclose()` an application should assume that its
24068 symbols are no longer available to `dlsym()`. All objects loaded automatically as a result of
24069 invoking `dlopen()` on the referenced object shall also be closed if this is the last reference to it.24070 Although a `dlclose()` operation is not required to remove structures from an address space,
24071 neither is an implementation prohibited from doing so. The only restriction on such a removal is
24072 that no object shall be removed to which references have been relocated, until or unless all such
24073 references are removed. For instance, an object that had been loaded with a `dlopen()` operation
24074 specifying the `RTLD_GLOBAL` flag might provide a target for dynamic relocations performed in
24075 the processing of other objects—in such environments, an application may assume that no
24076 relocation, once made, shall be undone or remade unless the object requiring the relocation has
24077 itself been removed.24078 **RETURN VALUE**24079 If the referenced object was successfully closed, `dlclose()` shall return 0. If the object could not be
24080 closed, or if `handle` does not refer to an open object, `dlclose()` shall return a non-zero value. More
24081 detailed diagnostic information shall be available through `dLError()`.24082 **ERRORS**

24083 No errors are defined.

24084 **EXAMPLES**24085 The following example illustrates use of `dlopen()` and `dlclose()`:24086

```
...
```

24087

```
/* Open a dynamic library and then close it ... */
```

24088

```
#include <dlfcn.h>
```

24089

```
void *mylib;
```

24090

```
int eret;
```

24091

```
mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
```

24092

```
...
```

24093

```
eret = dlclosure(mylib);
```

24094

```
...
```

24095 **APPLICATION USAGE**24096 A conforming application should employ a `handle` returned from a `dlopen()` invocation only
24097 within a given scope bracketed by the `dlopen()` and `dlclose()` operations. Implementations are
24098 free to use reference counting or other techniques such that multiple calls to `dlopen()` referencing
24099 the same object may return the same object for `handle`. Implementations are also free to reuse a
24100 `handle`. For these reasons, the value of a `handle` must be treated as an opaque object by the
24101 application, used only in calls to `dlsym()` and `dlclose()`.

24102
24103
24104
24105
24106
24107
24108
24109
24110
24111
24112
24113
24114
24115

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlerror(), *dlopen()*, *dlsym()*

XBD <dlfcn.h>

CHANGE HISTORY

First released in Issue 5.

Issue 6

The DESCRIPTION is updated to say that the referenced object is closed “if this is the last reference to it”.

Issue 7

The *dlopen()* function is moved from the XSI option to Base.

DRAFT

dlderror()24116 **NAME**24117 `dlderror` — get diagnostic information24118 **SYNOPSIS**24119 `#include <dldfcn.h>`24120 `char *dlderror(void);`24121 **DESCRIPTION**

24122 The `dlderror()` function shall return a null-terminated character string (with no trailing
 24123 <newline>) that describes the last error that occurred during dynamic linking processing. If no
 24124 dynamic linking errors have occurred since the last invocation of `dlderror()`, `dlderror()` shall return
 24125 NULL. Thus, invoking `dlderror()` a second time, immediately following a prior invocation, shall
 24126 result in NULL being returned.

24127 The `dlderror()` function need not be thread-safe. A function that is not required to be thread-safe is
 24128 not required to be reentrant.

24129 **RETURN VALUE**

24130 If successful, `dlderror()` shall return a null-terminated character string; otherwise, NULL shall be
 24131 returned.

24132 **ERRORS**

24133 No errors are defined.

24134 **EXAMPLES**

24135 The following example prints out the last dynamic linking error:

```
24136 ...
24137 #include <dldfcn.h>
24138 char *errstr;
24139 errstr = dlderror();
24140 if (errstr != NULL)
24141     printf ("A dynamic linking error occurred: (%s)\n", errstr);
24142 ...
```

24143 **APPLICATION USAGE**

24144 The messages returned by `dlderror()` may reside in a static buffer that is overwritten on each call
 24145 to `dlderror()`. Application code should not write to this buffer. Programs wishing to preserve an
 24146 error message should make their own copies of that message. Depending on the application
 24147 environment with respect to asynchronous execution events, such as signals or other
 24148 asynchronous computation sharing the address space, conforming applications should use a
 24149 critical section to retrieve the error pointer and buffer.

24150 **RATIONALE**

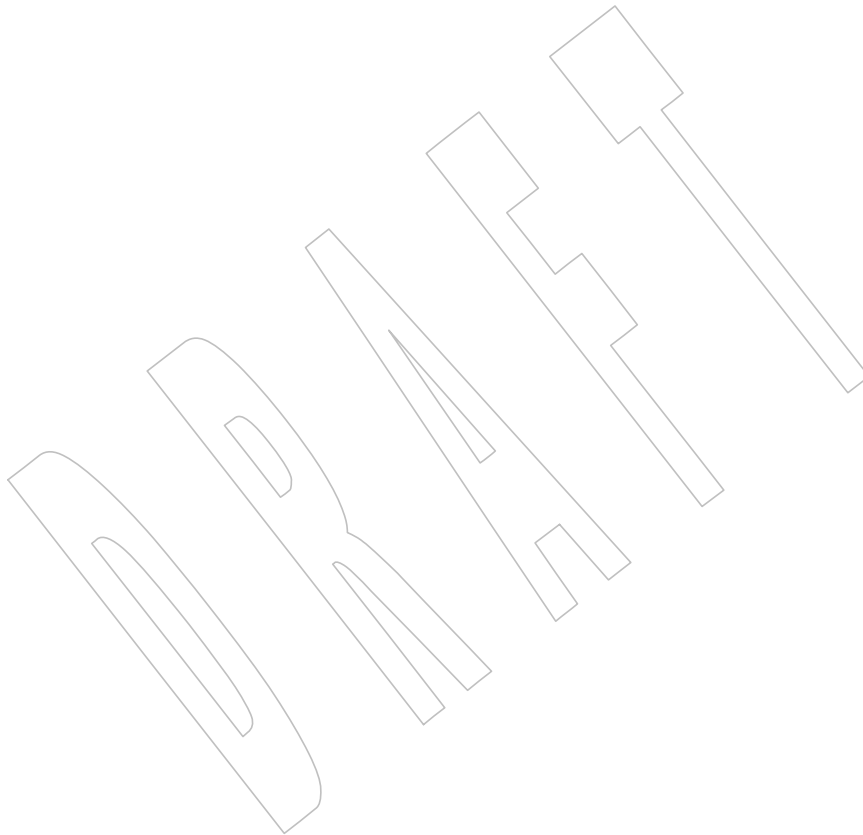
24151 None.

24152 **FUTURE DIRECTIONS**

24153 None.

24154 **SEE ALSO**24155 [*dlclose\(\)*](#), [*dlopen\(\)*](#), [*dlsym\(\)*](#)24156 XBD [*<dldfcn.h>*](#)

24157	CHANGE HISTORY
24158	First released in Issue 5.
24159	Issue 6
24160	In the DESCRIPTION the note about reentrancy and thread-safety is added.
24161	Issue 7
24162	The <i>dlerror()</i> function is moved from the XSI option to the Base.



24163 **NAME**24164 `dlopen` — gain access to an executable object file24165 **SYNOPSIS**24166 `#include <dlfcn.h>`24167 `void *dlopen(const char *file, int mode);`24168 **DESCRIPTION**

24169 The `dlopen()` function shall make an executable object file specified by `file` available to the calling
 24170 program. The class of files eligible for this operation and the manner of their construction are
 24171 implementation-defined, though typically such files are executable objects such as shared
 24172 libraries, relocatable files, or programs. Note that some implementations permit the construction
 24173 of dependencies between such objects that are embedded within files. In such cases, a `dlopen()`
 24174 operation shall load such dependencies in addition to the object referenced by `file`.
 24175 Implementations may also impose specific constraints on the construction of programs that can
 24176 employ `dlopen()` and its related services.

24177 A successful `dlopen()` shall return a *handle* which the caller may use on subsequent calls to
 24178 `dlsym()` and `dlclose()`. The value of this *handle* should not be interpreted in any way by the caller.

24179 The `file` argument is used to construct a pathname to the object file. If `file` contains a slash
 24180 character, the `file` argument is used as the pathname for the file. Otherwise, `file` is used in an
 24181 implementation-defined manner to yield a pathname.

24182 If the value of `file` is 0, `dlopen()` shall provide a *handle* on a global symbol object. This object shall
 24183 provide access to the symbols from an ordered set of objects consisting of the original program
 24184 image file, together with any objects loaded at program start-up as specified by that process
 24185 image file (for example, shared libraries), and the set of objects loaded using a `dlopen()` operation
 24186 together with the `RTLD_GLOBAL` flag. As the latter set of objects can change during execution,
 24187 the set identified by *handle* can also change dynamically.

24188 Only a single copy of an object file is brought into the address space, even if `dlopen()` is invoked
 24189 multiple times in reference to the file, and even if different pathnames are used to reference the
 24190 file.

24191 The `mode` parameter describes how `dlopen()` shall operate upon `file` with respect to the processing
 24192 of relocations and the scope of visibility of the symbols provided within `file`. When an object is
 24193 brought into the address space of a process, it may contain references to symbols whose
 24194 addresses are not known until the object is loaded. These references shall be relocated before the
 24195 symbols can be accessed. The `mode` parameter governs when these relocations take place and
 24196 may have the following values:

24197 **RTLD_LAZY** Relocations shall be performed at an implementation-defined time,
 24198 ranging from the time of the `dlopen()` call until the first reference to a
 24199 given symbol occurs. Specifying `RTLD_LAZY` should improve
 24200 performance on implementations supporting dynamic symbol binding as
 24201 a process may not reference all of the functions in any given object. And,
 24202 for systems supporting dynamic symbol resolution for normal process
 24203 execution, this behavior mimics the normal handling of process
 24204 execution.

24205 **RTLD_NOW** All necessary relocations shall be performed when the object is first
 24206 loaded. This may waste some processing if relocations are performed for
 24207 functions that are never referenced. This behavior may be useful for
 24208 applications that need to know as soon as an object is loaded that all
 24209 symbols referenced during execution are available.

24210 Any object loaded by *dlopen()* that requires relocations against global symbols can reference the
 24211 symbols in the original process image file, any objects loaded at program start-up, from the
 24212 object itself as well as any other object included in the same *dlopen()* invocation, and any objects
 24213 that were loaded in any *dlopen()* invocation and which specified the RTLD_GLOBAL flag. To
 24214 determine the scope of visibility for the symbols loaded with a *dlopen()* invocation, the *mode*
 24215 parameter should be a bitwise-inclusive OR with one of the following values:

24216 RTLD_GLOBAL The object's symbols shall be made available for the relocation processing
 24217 of any other object. In addition, symbol lookup using *dlopen(0, mode)* and
 24218 an associated *dlsym()* allows objects loaded with this *mode* to be searched.

24219 RTLD_LOCAL The object's symbols shall not be made available for the relocation
 24220 processing of any other object.

24221 If neither RTLD_GLOBAL nor RTLD_LOCAL are specified, then the default behavior is
 24222 unspecified.

24223 If a *file* is specified in multiple *dlopen()* invocations, *mode* is interpreted at each invocation. Note,
 24224 however, that once RTLD_NOW has been specified all relocations shall have been completed
 24225 rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations
 24226 irrelevant. Similarly, note that once RTLD_GLOBAL has been specified the object shall maintain
 24227 the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL,
 24228 as long as the object remains in the address space (see *dlclose()*).

24229 Symbols introduced into a program through calls to *dlopen()* may be used in relocation activities.
 24230 Symbols so introduced may duplicate symbols already defined by the program or previous
 24231 *dlopen()* operations. To resolve the ambiguities such a situation might present, the resolution of a
 24232 symbol reference to symbol definition is based on a symbol resolution order. Two such
 24233 resolution orders are defined: *load* or *dependency* ordering. Load order establishes an ordering
 24234 among symbol definitions, such that the definition first loaded (including definitions from the
 24235 image file and any dependent objects loaded with it) has priority over objects added later (via
 24236 *dlopen()*). Load ordering is used in relocation processing. Dependency ordering uses a breadth-
 24237 first order starting with a given object, then all of its dependencies, then any dependents of
 24238 those, iterating until all dependencies are satisfied. With the exception of the global symbol
 24239 object obtained via a *dlopen()* operation on a *file* of 0, dependency ordering is used by the
 24240 *dlsym()* function. Load ordering is used in *dlsym()* operations upon the global symbol object.

24241 When an object is first made accessible via *dlopen()* it and its dependent objects are added in
 24242 dependency order. Once all the objects are added, relocations are performed using load order.
 24243 Note that if an object or its dependencies had been previously loaded, the load and dependency
 24244 orders may yield different resolutions.

24245 The symbols introduced by *dlopen()* operations and available through *dlsym()* are at a minimum
 24246 those which are exported as symbols of global scope by the object. Typically such symbols shall
 24247 be those that were specified in (for example) C source code as having *extern* linkage. The precise
 24248 manner in which an implementation constructs the set of exported symbols for a *dlopen()* object
 24249 is specified by that implementation.

24250 RETURN VALUE

24251 If *file* cannot be found, cannot be opened for reading, is not of an appropriate object format for
 24252 processing by *dlopen()*, or if an error occurs during the process of loading *file* or relocating its
 24253 symbolic references, *dlopen()* shall return NULL. More detailed diagnostic information shall be
 24254 available through *dlerror()*.

24255 ERRORS

24256 No errors are defined.

dlopen()

24257

EXAMPLES

24258

Refer to *dlsym()* (on page 709).

24259

APPLICATION USAGE

24260

None.

24261

RATIONALE

24262

None.

24263

FUTURE DIRECTIONS

24264

None.

24265

SEE ALSO

24266

dlclose(), *dlderror()*, *dlsym()*

24267

XBD `<dlfcn.h>`

24268

CHANGE HISTORY

24269

First released in Issue 5.

24270

Issue 6

24271

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/21 is applied, changing the default behavior in the DESCRIPTION when neither RTLD_GLOBAL nor RTLD_LOCAL are specified from implementation-defined to unspecified.

24272

24273

24274

Issue 7

24275

The *dlopen()* function is moved from the XSI option to the Base.

24276

The EXAMPLES section is updated to refer to *dlsym()*.

24277 **NAME**24278 dlsym — obtain the address of a symbol from a *dlopen()* object24279 **SYNOPSIS**

24280 #include <dlfcn.h>

24281 void *dlsym(void *restrict *handle*, const char *restrict *name*);24282 **DESCRIPTION**

24283 The *dlsym()* function shall obtain the address of a symbol defined within an object made
 24284 accessible through a *dlopen()* call. The *handle* argument is the value returned from a call to
 24285 *dlopen()* (and which has not since been released via a call to *dlclose()*), and *name* is the symbol's
 24286 name as a character string.

24287 The *dlsym()* function shall search for the named symbol in all objects loaded automatically as a
 24288 result of loading the object referenced by *handle* (see *dlopen()*). Load ordering is used in *dlsym()*
 24289 operations upon the global symbol object. The symbol resolution algorithm used shall be
 24290 dependency order as described in *dlopen()*.

24291 The RTLD_DEFAULT and RTLD_NEXT flags are reserved for future use.

24292 **RETURN VALUE**

24293 If *handle* does not refer to a valid object opened by *dlopen()*, or if the named symbol cannot be
 24294 found within any of the objects associated with *handle*, *dlsym()* shall return NULL. More
 24295 detailed diagnostic information shall be available through *dLError()*.

24296 **ERRORS**

24297 No errors are defined.

24298 **EXAMPLES**

24299 The following example shows how *dlopen()* and *dlsym()* can be used to access either function or
 24300 data objects. For simplicity, error checking has been omitted.

```
24301 void    *handle;
24302 int     *iptr, (*fptr)(int);

24303 /* open the needed object */
24304 handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);

24305 /* find the address of function and data objects */
24306 *(void **)(&fptr) = dlsym(handle, "my_function");
24307 iptr = (int *)dlsym(handle, "my_object");

24308 /* invoke function, passing value of integer as a parameter */
24309 (*fptr)(*iptr);
```

24310 **APPLICATION USAGE**

24311 Special purpose values for *handle* are reserved for future use. These values and their meanings
 24312 are:

24313 RTLD_DEFAULT The symbol lookup happens in the normal global scope; that is, a search for a
 24314 symbol using this handle would find the same definition as a direct use of this
 24315 symbol in the program code.

24316 RTLD_NEXT Specifies the next object after this one that defines *name*. *This one* refers to the
 24317 object containing the invocation of *dlsym()*. The *next* object is the one found
 24318 upon the application of a load order symbol resolution algorithm (see
 24319 *dlopen()*). The next object is either one of global scope (because it was
 24320 introduced as part of the original process image or because it was added with

24321
24322
24323
24324
24325
24326
24327
24328

a *dlopen()* operation including the `RTLD_GLOBAL` flag), or is an object that was included in the same *dlopen()* operation that loaded this one.

The `RTLD_NEXT` flag is useful to navigate an intentionally created hierarchy of multiply-defined symbols created through *interposition*. For example, if a program wished to create an implementation of *malloc()* that embedded some statistics gathering about memory allocations, such an implementation could use the real *malloc()* definition to perform the memory allocation—and itself only embed the necessary logic to implement the statistics gathering function.

24329
24330
24331
24332
24333**RATIONALE**

The ISO C standard does not require that pointers to functions can be cast back and forth to pointers to data. However, POSIX-conforming implementations are required to support this, as noted in [Section 2.12.3](#) (on page 519). The result of converting a pointer to a function into a pointer to another data type (except `void *`) is still undefined, however.

24334
24335

Note that compilers conforming to the ISO C standard are required to generate a warning if a conversion from a `void *` pointer to a function pointer is attempted as in:

24336

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

24337
24338**FUTURE DIRECTIONS**

None.

24339
24340**SEE ALSO**

[*dlclose\(\)*](#), [*dlerror\(\)*](#), [*dlopen\(\)*](#)

24341

XBD [<dlfcn.h>](#)

24342
24343**CHANGE HISTORY**

First released in Issue 5.

24344
24345
24346
24347
24348
24349
24350**Issue 6**

The `restrict` keyword is added to the *dlsym()* prototype for alignment with the ISO/IEC 9899:1999 standard.

The `RTLD_DEFAULT` and `RTLD_NEXT` flags are reserved for future use.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/14 is applied, correcting an example, and adding text to the RATIONALE describing issues related to conversion of pointers to functions and back again.

24351
24352**Issue 7**

The *dlsym()* function is moved from the XSI option to the Base.

24353 **NAME**
24354 `dprintf` — print formatted output

24355 **SYNOPSIS**

24356 CX `#include <stdio.h>`
24357 `int dprintf(int files, const char *format, ...);`

24358 **DESCRIPTION**

24359 Refer to *fprintf()*.

24360 **NAME**

24361 `drand48`, `erand48`, `rand48`, `lcong48`, `rand48`, `mrnd48`, `nrnd48`, `seed48`, `srand48` — generate
 24362 uniformly distributed pseudo-random numbers

24363 **SYNOPSIS**

```
24364 XSI #include <stdlib.h>
24365
24365 double drand48(void);
24366 double erand48(unsigned short xsubi[3]);
24367 long jrand48(unsigned short xsubi[3]);
24368 void lcong48(unsigned short param[7]);
24369 long rand48(void);
24370 long mrnd48(void);
24371 long nrnd48(unsigned short xsubi[3]);
24372 unsigned short *seed48(unsigned short seed16v[3]);
24373 void srand48(long seedval);
```

24374 **DESCRIPTION**

24375 This family of functions shall generate pseudo-random numbers using a linear congruential
 24376 algorithm and 48-bit integer arithmetic.

24377 The `drand48()` and `erand48()` functions shall return non-negative, double-precision, floating-
 24378 point values, uniformly distributed over the interval [0.0,1.0).

24379 The `rand48()` and `nrnd48()` functions shall return non-negative, long integers, uniformly
 24380 distributed over the interval $[0,2^{31})$.

24381 The `mrnd48()` and `jrand48()` functions shall return signed long integers uniformly distributed
 24382 over the interval $[-2^{31},2^{31})$.

24383 The `srand48()`, `seed48()`, and `lcong48()` functions are initialization entry points, one of which
 24384 should be invoked before either `drand48()`, `rand48()`, or `mrnd48()` is called. (Although it is not
 24385 recommended practice, constant default initializer values shall be supplied automatically if
 24386 `drand48()`, `rand48()`, or `mrnd48()` is called without a prior call to an initialization entry point.)
 24387 The `erand48()`, `nrnd48()`, and `jrand48()` functions do not require an initialization entry point to
 24388 be called first.

24389 All the routines work by generating a sequence of 48-bit integer values, X_i , according to the
 24390 linear congruential formula:

$$24391 X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n \geq 0$$

24392 The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless `lcong48()` is invoked,
 24393 the multiplier value a and the addend value c are given by:

$$24394 a = 5DEECE66D_{16} = 273673163155_8$$

$$24395 c = B_{16} = 13_8$$

24396 The value returned by any of the `drand48()`, `erand48()`, `jrand48()`, `rand48()`, `mrnd48()`, or
 24397 `nrnd48()` functions is computed by first generating the next 48-bit X_i in the sequence. Then the
 24398 appropriate number of bits, according to the type of data item to be returned, are copied from
 24399 the high-order (leftmost) bits of X_i and transformed into the returned value.

24400 The `drand48()`, `rand48()`, and `mrnd48()` functions store the last 48-bit X_i generated in an
 24401 internal buffer; that is why the application shall ensure that these are initialized prior to being
 24402 invoked. The `erand48()`, `nrnd48()`, and `jrand48()` functions require the calling program to

24403 provide storage for the successive X_i values in the array specified as an argument when the
 24404 functions are invoked. That is why these routines do not have to be initialized; the calling
 24405 program merely has to place the desired initial value of X_i into the array and pass it as an
 24406 argument. By using different arguments, *erand48()*, *nrnd48()*, and *jrnd48()* allow separate
 24407 modules of a large program to generate several *independent* streams of pseudo-random numbers;
 24408 that is, the sequence of numbers in each stream shall *not* depend upon how many times the
 24409 routines are called to generate numbers for the other streams.

24410 The initializer function *srand48()* sets the high-order 32 bits of X_i to the low-order 32 bits
 24411 contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

24412 The initializer function *seed48()* sets the value of X_i to the 48-bit value specified in the argument
 24413 array. The low-order 16 bits of X_i are set to the low-order 16 bits of *seed16v*[0]. The mid-order 16
 24414 bits of X_i are set to the low-order 16 bits of *seed16v*[1]. The high-order 16 bits of X_i are set to the
 24415 low-order 16 bits of *seed16v*[2]. In addition, the previous value of X_i is copied into a 48-bit
 24416 internal buffer, used only by *seed48()*, and a pointer to this buffer is the value returned by
 24417 *seed48()*. This returned pointer, which can just be ignored if not needed, is useful if a program is
 24418 to be restarted from a given point at some future time—use the pointer to get at and store the
 24419 last X_i value, and then use this value to reinitialize via *seed48()* when the program is restarted.

24420 The initializer function *lcong48()* allows the user to specify the initial X_i , the multiplier value a ,
 24421 and the addend value c . Argument array elements *param*[0-2] specify X_i , *param*[3-5] specify the
 24422 multiplier a , and *param*[6] specifies the 16-bit addend c . After *lcong48()* is called, a subsequent
 24423 call to either *srand48()* or *seed48()* shall restore the standard multiplier and addend values, a and
 24424 c , specified above.

24425 The *drand48()*, *lrnd48()*, and *mrnd48()* function need not be thread-safe. A function that is not
 24426 required to be thread-safe is not required to be reentrant.

24427 RETURN VALUE

24428 As described in the DESCRIPTION above.

24429 ERRORS

24430 No errors are defined.

24431 EXAMPLES

24432 None.

24433 APPLICATION USAGE

24434 None.

24435 RATIONALE

24436 None.

24437 FUTURE DIRECTIONS

24438 None.

24439 SEE ALSO

24440 [rand\(\)](#)

24441 XBD [<stdlib.h>](#)

24442 CHANGE HISTORY

24443 First released in Issue 1. Derived from Issue 1 of the SVID.

24444 Issue 5

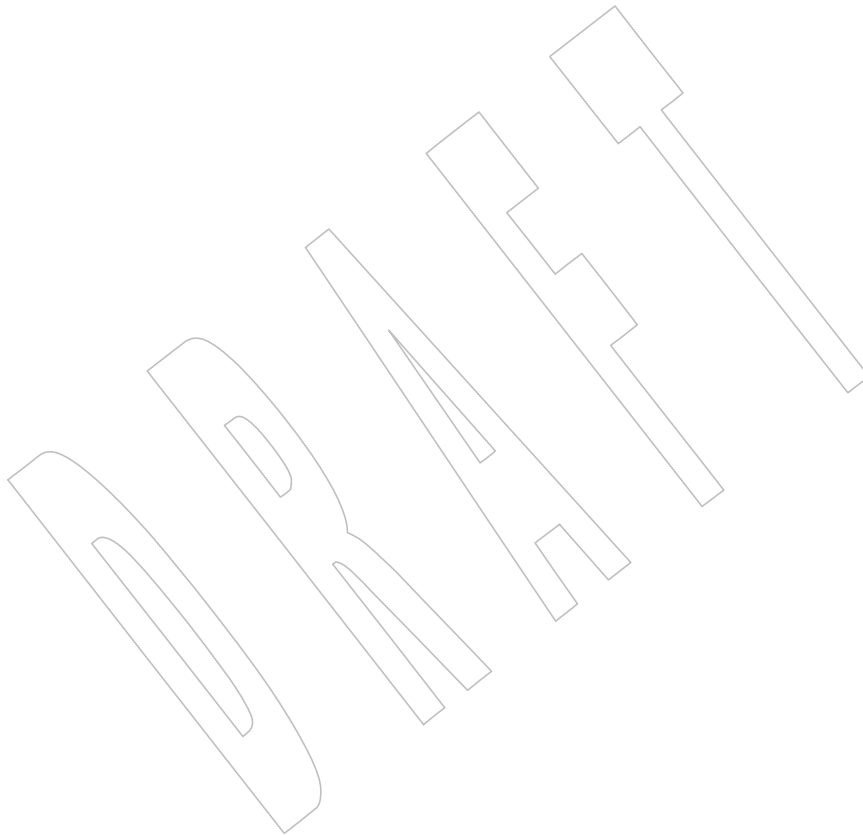
24445 A note indicating that the *drand48()*, *lrnd48()*, and *mrnd48()* functions need not be reentrant is
 24446 added to the DESCRIPTION.

24447

24448

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.



24449 **NAME**

24450 dup, dup2 — duplicate an open file descriptor

24451 **SYNOPSIS**

24452 #include <unistd.h>

24453 int dup(int *fil-des*);24454 int dup2(int *fil-des*, int *fil-des2*);24455 **DESCRIPTION**24456 The *dup()* function provides an alternative interface to the service provided by *fcntl()* using the
24457 F_DUPFD command. The call *dup(fil-des)* shall be equivalent to:24458 *fcntl(fil-des, F_DUPFD, 0);*24459 The *dup2()* function shall cause the file descriptor *fil-des2* to refer to the same open file
24460 description as the file descriptor *fil-des* and to share any locks, and shall return *fil-des2*. If *fil-des2* is
24461 already a valid open file descriptor, it shall be closed first, unless *fil-des* is equal to *fil-des2* in which
24462 case *dup2()* shall return *fil-des2* without closing it. If the close operation fails to close *fil-des2*,
24463 *dup2()* shall return -1 without changing the open file description to which *fil-des2* refers. If *fil-des*
24464 is not a valid file descriptor, *dup2()* shall return -1 and shall not close *fil-des2*. If *fil-des2* is less than
24465 0 or greater than or equal to {OPEN_MAX}, *dup2()* shall return -1 with *errno* set to [EBADF].24466 On successful completion, if *fil-des* is not equal to *fil-des2*, the FD_CLOEXEC flag associated with
24467 *fil-des2* shall be cleared. If *fil-des* is equal to *fil-des2*, the FD_CLOEXEC flag associated with *fil-des2*
24468 shall not be changed.24469 TYM If *fil-des* refers to a typed memory object, the result of the *dup2()* function is unspecified.24470 **RETURN VALUE**24471 Upon successful completion a non-negative integer, namely the file descriptor, shall be returned;
24472 otherwise, -1 shall be returned and *errno* set to indicate the error.24473 **ERRORS**24474 The *dup()* function shall fail if:24475 [EBADF] The *fil-des* argument is not a valid open file descriptor.

24476 [EMFILE] All file descriptors available to the process are currently open.

24477 The *dup2()* function shall fail if:24478 [EBADF] The *fil-des* argument is not a valid open file descriptor or the argument *fil-des2* is
24479 negative or greater than or equal to {OPEN_MAX}.24480 [EINTR] The *dup2()* function was interrupted by a signal.24481 The *dup2()* function may fail if: +24482 [EIO] An I/O error occurred while attempting to close *fil-des2*. +

24483 **EXAMPLES**24484 **Redirecting Standard Output to a File**

24485 The following example closes standard output for the current processes, re-assigns standard
 24486 output to go to the file referenced by *pfid*, and closes the original file descriptor to clean up.

```
24487 #include <unistd.h>
24488 ...
24489 int pfid;
24490 ...
24491 close(1);
24492 dup(pfid);
24493 close(pfid);
24494 ...
```

24495 **Redirecting Error Messages**

24496 The following example redirects messages from *stderr* to *stdout*.

```
24497 #include <unistd.h>
24498 ...
24499 dup2(1, 2);
24500 ...
```

24501 **APPLICATION USAGE**

24502 None.

24503 **RATIONALE**

24504 The *dup()* and *dup2()* functions are redundant. Their services are also provided by the *fcntl()*
 24505 function. They have been included in this volume of POSIX.1-200x primarily for historical
 24506 reasons, since many existing applications use them.

24507 The *dup2()* function is not marked obsolescent because it presents a type-safe version of -
 24508 functionality provided in a type-unsafe version by *fcntl()*. It is used in the POSIX Ada binding.

24509 The *dup2()* function is not intended for use in critical regions as a synchronization mechanism.

24510 In the description of [EBADF], the case of *fildes* being out of range is covered by the given case of
 24511 *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of
 24512 invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter
 24513 whether *fildes2* refers to an open file when the *dup2()* call is made.

24514 **FUTURE DIRECTIONS**

24515 None.

24516 **SEE ALSO**

24517 *close()*, *fcntl()*, *open()*

24518 XBD [<unistd.h>](#) |

24519 **CHANGE HISTORY**

24520 First released in Issue 1. Derived from Issue 1 of the SVID.

24521 **Issue 7**

24522 SD5-XSH-ERN-187 is applied. +

24523 **NAME**

24524 duplocale — duplicate a locale object

24525 **SYNOPSIS**

```
24526 CX #include <locale.h>
24527 locale_t duplocale(locale_t locobj);
```

24528 **DESCRIPTION**

24529 The *duplocale()* function shall create a duplicate copy of the locale object referenced by the *locobj*
 24530 argument.

24531 **RETURN VALUE**

24532 Upon successful completion, the *duplocale()* function shall return a handle for a new locale
 24533 object. Otherwise, *duplocale()* shall return **(locale_t)0** and set *errno* to indicate the error.

24534 **ERRORS**

24535 The *duplocale()* function shall fail if:

24536 [ENOMEM] There is not enough memory available to create the locale object or load the
 24537 locale data.

24538 The *duplocale()* function may fail if:

24539 [EINVAL] *locobj* is not a handle for a locale object.

24540 **EXAMPLES**24541 **Constructing an Altered Version of an Existing Locale Object**

24542 The following example shows a code fragment to create a slightly altered version of an existing
 24543 locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME*
 24544 category data in the locale object with that from the named locale.

```
24545 #include <locale.h>
24546 ...
24547 locale_t
24548 with_changed_lc_time (locale_t obj, const char *name)
24549 {
24550     locale_t retval = duplocale (obj);
24551     if (retval != (locale_t) 0)
24552     {
24553         locale_t changed = newlocale (LC_TIME_MASK, name, retval);
24554         if (changed == (locale_t) 0)
24555             /* An error occurred. Free all allocated resources. */
24556             freelocale (retval);
24557         retval = changed;
24558     }
24559     return retval; }
24560 }
```

24561 **APPLICATION USAGE**

24562 The use of the *duplocale()* function is recommended for situations where a locale object is being
 24563 used in multiple places, and it is possible that the lifetime of the locale object might end before
 24564 all uses are finished. Another reason to duplicate a locale object is if a slightly modified form is
 24565 needed. This can be achieved by a call to *newlocale()* following the *duplocale()* call.

duplocale()

24566 As with the *newlocale()* function, handles for locale objects created by the *duplocale()* function
24567 should be released by a corresponding call to *freelocale()*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

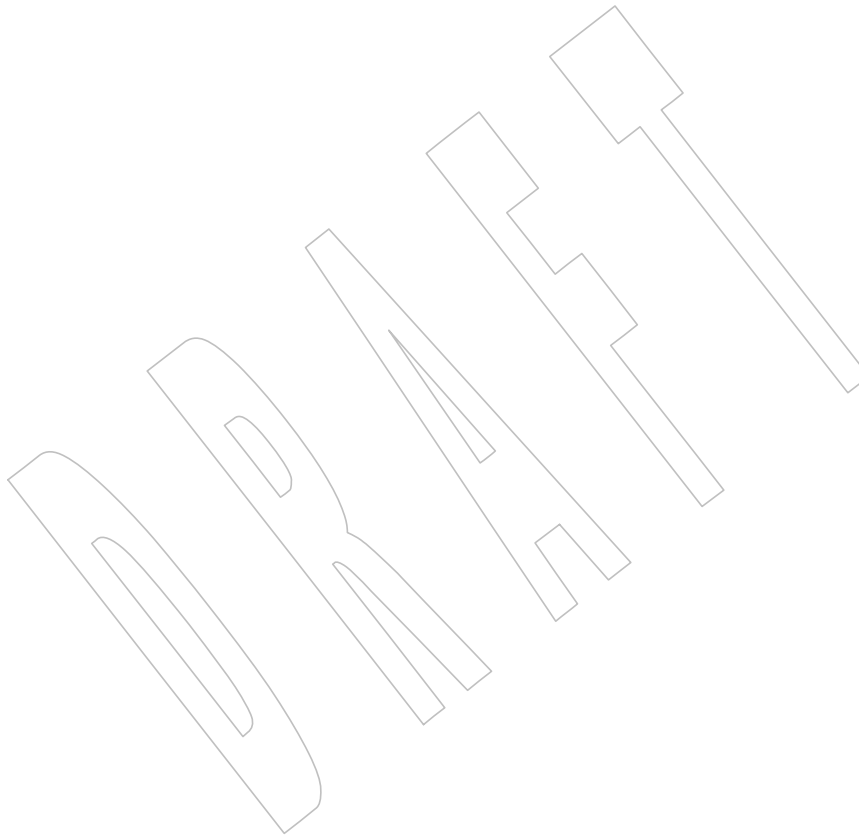
SEE ALSO

freelocale(), *newlocale()*, *uselocale()*

XBD <locale.h>

CHANGE HISTORY

First released in Issue 7.



24577 **NAME**24578 encrypt — encoding function (**CRYPT**)24579 **SYNOPSIS**

```
24580 XSI #include <unistd.h>
24581 void encrypt(char block[64], int edflag);
```

24582 **DESCRIPTION**

24583 The *encrypt()* function shall provide access to an implementation-defined encoding algorithm.
 24584 The key generated by *setkey()* is used to encrypt the string *block* with *encrypt()*.

24585 The *block* argument to *encrypt()* shall be an array of length 64 bytes containing only the bytes
 24586 with values of 0 and 1. The array is modified in place to a similar array using the key set by
 24587 *setkey()*. If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see
 24588 the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to
 24589 [ENOSYS].

24590 The *encrypt()* function shall not change the setting of *errno* if successful. An application wishing
 24591 to check for error situations should set *errno* to 0 before calling *encrypt()*. If *errno* is non-zero on
 24592 return, an error has occurred.

24593 The *encrypt()* function need not be thread-safe. A function that is not required to be thread-safe
 24594 is not required to be reentrant.

24595 **RETURN VALUE**24596 The *encrypt()* function shall not return a value.24597 **ERRORS**24598 The *encrypt()* function shall fail if:

24599 [ENOSYS] The functionality is not supported on this implementation.

24600 **EXAMPLES**

24601 None.

24602 **APPLICATION USAGE**24603 Historical implementations of the *encrypt()* function used a rather primitive encoding algorithm.

24604 In some environments, decoding might not be implemented. This is related to some Government
 24605 restrictions on encryption and decryption routines. Historical practice has been to ship a
 24606 different version of the encryption library without the decryption feature in the routines
 24607 supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

24608 **RATIONALE**

24609 None.

24610 **FUTURE DIRECTIONS**

24611 None.

24612 **SEE ALSO**24613 *crypt()*, *setkey()*

24614 XBD <unistd.h>

encrypt()

24615

CHANGE HISTORY

24616

First released in Issue 1. Derived from Issue 1 of the SVID.

24617

Issue 5

24618

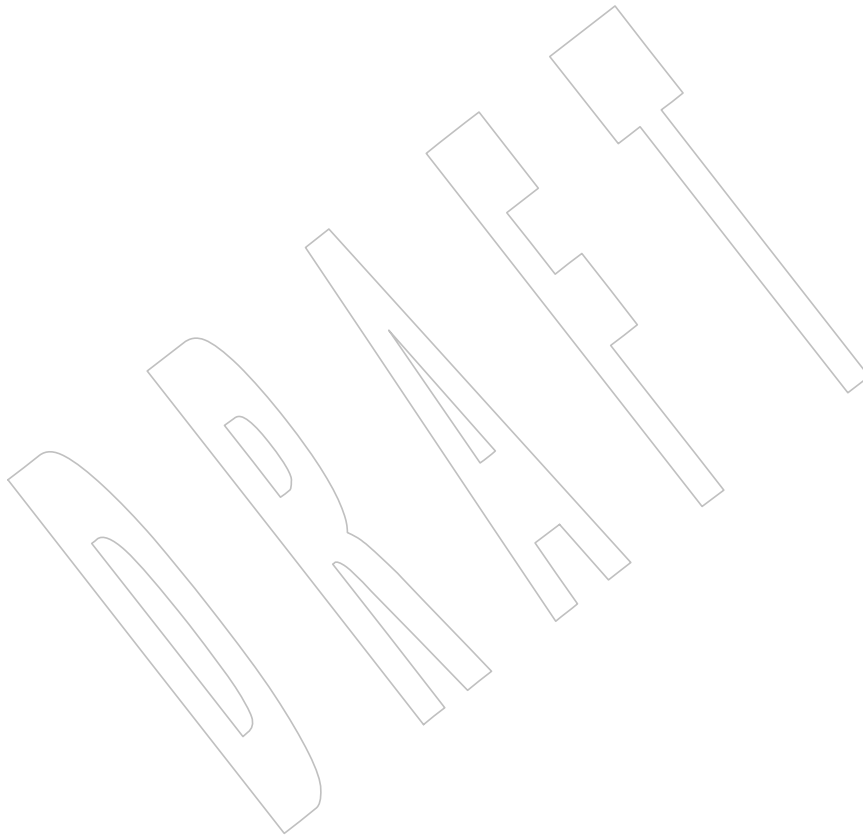
A note indicating that this function need not be reentrant is added to the DESCRIPTION.

24619

Issue 6

24620

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.



24621 **NAME**
 24622 endgrent, getgrent, setgrent — group database entry functions

24623 **SYNOPSIS**

```
24624 XSI #include <grp.h>
24625 void endgrent(void);
24626 struct group *getgrent(void);
24627 void setgrent(void);
```

24628 **DESCRIPTION**

24629 The *getgrent()* function shall return a pointer to a structure containing the broken-out fields of an
 24630 entry in the group database. When first called, *getgrent()* shall return a pointer to a **group**
 24631 structure containing the first entry in the group database. Thereafter, it shall return a pointer to a
 24632 **group** structure containing the next group structure in the group database, so successive calls
 24633 may be used to search the entire database.

24634 An implementation that provides extended security controls may impose further
 24635 implementation-defined restrictions on accessing the group database. In particular, the system
 24636 may deny the existence of some or all of the group database entries associated with groups other
 24637 than those groups associated with the caller and may omit users other than the caller from the
 24638 list of members of groups in database entries that are returned.

24639 The *setgrent()* function shall rewind the group database to allow repeated searches.

24640 The *endgrent()* function may be called to close the group database when processing is complete.

24641 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 24642 required to be reentrant.

24643 **RETURN VALUE**

24644 When first called, *getgrent()* shall return a pointer to the first group structure in the group
 24645 database. Upon subsequent calls it shall return the next group structure in the group database.
 24646 The *getgrent()* function shall return a null pointer on end-of-file or an error and *errno* may be set
 24647 to indicate the error.

24648 The return value may point to a static area which is overwritten by a subsequent call to
 24649 *getgrgid()*, *getgrnam()*, or *getgrent()*.

24650 **ERRORS**

24651 The *getgrent()* function may fail if:

- | | | |
|-------|----------|--|
| 24652 | [EINTR] | A signal was caught during the operation. |
| 24653 | [EIO] | An I/O error has occurred. |
| 24654 | [EMFILE] | All file descriptors available to the process are currently open. |
| 24655 | [ENFILE] | The maximum allowable number of files is currently open in the system. |

endgrent()

24656

EXAMPLES

24657

None.

24658

APPLICATION USAGE

24659

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the group database, whether the database is a single file, or where in the file system name space the database resides. Applications should use *getgrnam()* and *getgrgid()* whenever possible because it avoids these dependencies.

24660

24661

24662

24663

RATIONALE

24664

None.

24665

FUTURE DIRECTIONS

24666

None.

24667

SEE ALSO

24668

endpwent(), *getgrgid()*, *getgrnam()*, *getlogin()*

24669

XBD <grp.h>

24670

CHANGE HISTORY

24671

First released in Issue 4, Version 2.

24672

Issue 5

24673

Moved from X/OPEN UNIX extension to BASE.

24674

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

24675

24676

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

24677

Issue 6

24678

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24679

Issue 7

24680

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

24681 **NAME**
 24682 `endhostent`, `gethostent`, `sethostent` — network host database functions

24683 **SYNOPSIS**
 24684 `#include <netdb.h>`
 24685 `void endhostent(void);`
 24686 `struct hostent *gethostent(void);`
 24687 `void sethostent(int stayopen);`

24688 **DESCRIPTION**
 24689 These functions shall retrieve information about hosts. This information is considered to be
 24690 stored in a database that can be accessed sequentially or randomly. The implementation of this
 24691 database is unspecified.

24692 **Note:** In many cases this database is implemented by the Domain Name System, as documented in
 24693 RFC 1034, RFC 1035, and RFC 1886.

24694 The `sethostent()` function shall open a connection to the database and set the next entry for
 24695 retrieval to the first entry in the database. If the `stayopen` argument is non-zero, the connection
 24696 shall not be closed by a call to `gethostent()`, and the implementation may maintain an open file
 24697 descriptor.

24698 The `gethostent()` function shall read the next entry in the database, opening and closing a
 24699 connection to the database as necessary.

24700 Entries shall be returned in **hostent** structures.

24701 The `endhostent()` function shall close the connection to the database, releasing any open file
 24702 descriptor.

24703 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 24704 required to be reentrant.

24705 **RETURN VALUE**
 24706 Upon successful completion, the `gethostent()` function shall return a pointer to a **hostent**
 24707 structure if the requested entry was found, and a null pointer if the end of the database was
 24708 reached or the requested entry was not found.

24709 **ERRORS**
 24710 No errors are defined for `endhostent()`, `gethostent()`, and `sethostent()`.

24711 **EXAMPLES**
 24712 None.

24713 **APPLICATION USAGE**
 24714 The `gethostent()` function may return pointers to static data, which may be overwritten by
 24715 subsequent calls to any of these functions.

24716 **RATIONALE**
 24717 None.

24718 **FUTURE DIRECTIONS**
 24719 None.

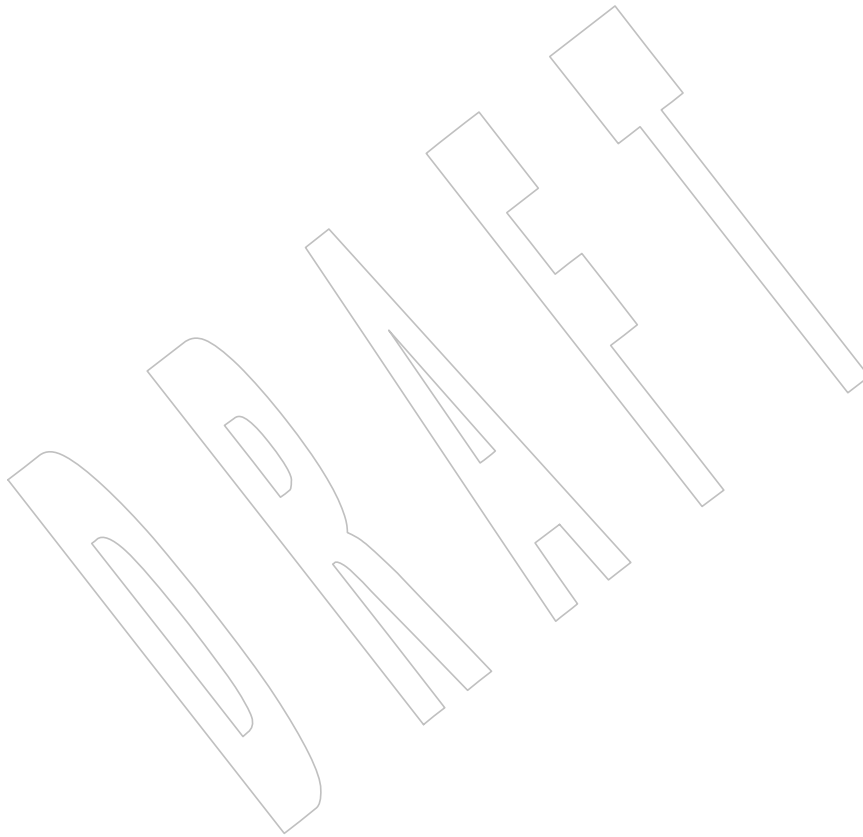
24720 **SEE ALSO**
 24721 [endservent\(\)](#)
 24722 XBD [<netdb.h>](#)

endhostent()

24723
24724

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



24725 **NAME**
 24726 endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

24727 **SYNOPSIS**

24728 #include <netdb.h>

24729 void endnetent(void);
 24730 struct netent *getnetbyaddr(uint32_t net, int type);
 24731 struct netent *getnetbyname(const char *name);
 24732 struct netent *getnetent(void);
 24733 void setnetent(int stayopen);

24734 **DESCRIPTION**

24735 These functions shall retrieve information about networks. This information is considered to be
 24736 stored in a database that can be accessed sequentially or randomly. The implementation of this
 24737 database is unspecified.

24738 The *setnetent()* function shall open and rewind the database. If the *stayopen* argument is non-
 24739 zero, the connection to the *net* database shall not be closed after each call to *getnetent()* (either
 24740 directly, or indirectly through one of the other *getnet**() functions), and the implementation may
 24741 maintain an open file descriptor to the database.

24742 The *getnetent()* function shall read the next entry of the database, opening and closing a
 24743 connection to the database as necessary.

24744 The *getnetbyaddr()* function shall search the database from the beginning, and find the first entry
 24745 for which the address family specified by *type* matches the *n_addrtype* member and the network
 24746 number *net* matches the *n_net* member, opening and closing a connection to the database as
 24747 necessary. The *net* argument shall be the network number in host byte order.

24748 The *getnetbyname()* function shall search the database from the beginning and find the first entry
 24749 for which the network name specified by *name* matches the *n_name* member, opening and
 24750 closing a connection to the database as necessary.

24751 The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions shall each return a pointer to a
 24752 **netent** structure, the members of which shall contain the fields of an entry in the network
 24753 database.

24754 The *endnetent()* function shall close the database, releasing any open file descriptor.

24755 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 24756 required to be reentrant.

24757 **RETURN VALUE**

24758 Upon successful completion, *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* shall return a
 24759 pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the
 24760 database was reached or the requested entry was not found. Otherwise, a null pointer shall be
 24761 returned.

24762 **ERRORS**

24763 No errors are defined.

endnetent()

24764

EXAMPLES

24765

None.

24766

APPLICATION USAGE

24767

The *getnetbyaddr()*, *getnetbyname()*, and *getnetent()* functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

24768

24769

RATIONALE

24770

None.

24771

FUTURE DIRECTIONS

24772

None.

24773

SEE ALSO

24774

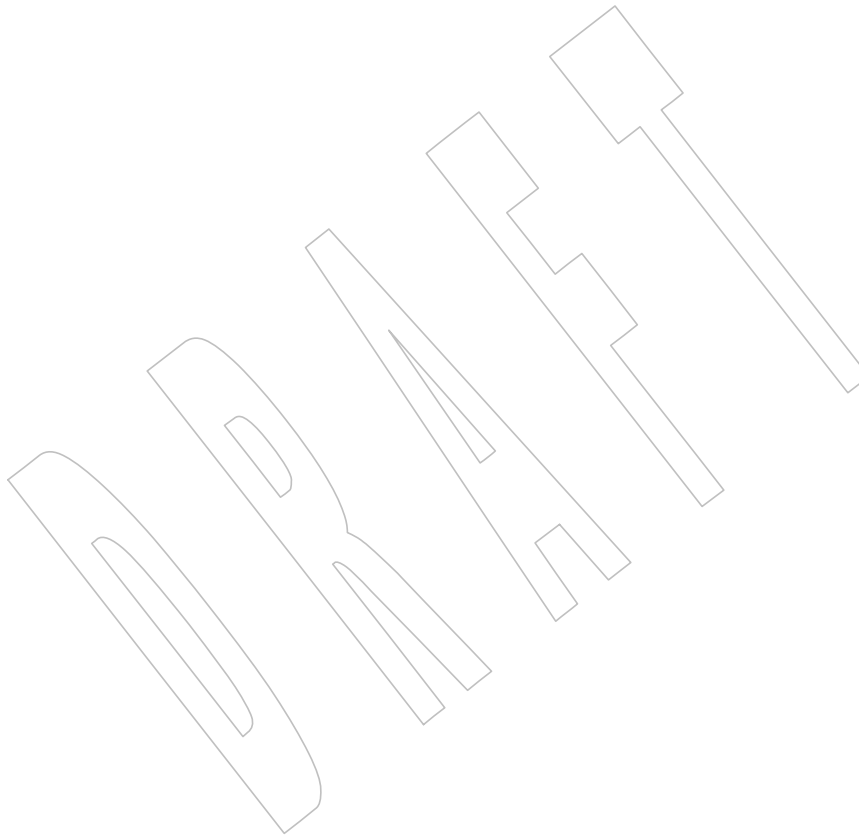
XBD <[netdb.h](#)>

24775

CHANGE HISTORY

24776

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



24777 **NAME**

24778 endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol
24779 database functions

24780 **SYNOPSIS**

```
24781 #include <netdb.h>

24782 void endprotoent(void);
24783 struct protoent *getprotobyname(const char *name);
24784 struct protoent *getprotobynumber(int proto);
24785 struct protoent *getprotoent(void);
24786 void setprotoent(int stayopen);
```

24787 **DESCRIPTION**

24788 These functions shall retrieve information about protocols. This information is considered to be
24789 stored in a database that can be accessed sequentially or randomly. The implementation of this
24790 database is unspecified.

24791 The *setprotoent()* function shall open a connection to the database, and set the next entry to the
24792 first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database
24793 shall not be closed after each call to *getprotoent()* (either directly, or indirectly through one of the
24794 other *getproto**(*)* functions), and the implementation may maintain an open file descriptor for
24795 the database.

24796 The *getprotobyname()* function shall search the database from the beginning and find the first
24797 entry for which the protocol name specified by *name* matches the *p_name* member, opening and
24798 closing a connection to the database as necessary.

24799 The *getprotobynumber()* function shall search the database from the beginning and find the first
24800 entry for which the protocol number specified by *proto* matches the *p_proto* member, opening
24801 and closing a connection to the database as necessary.

24802 The *getprotoent()* function shall read the next entry of the database, opening and closing a
24803 connection to the database as necessary.

24804 The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions shall each return a pointer
24805 to a **protoent** structure, the members of which shall contain the fields of an entry in the network
24806 protocol database.

24807 The *endprotoent()* function shall close the connection to the database, releasing any open file
24808 descriptor.

24809 These functions need not be thread-safe. A function that is not required to be thread-safe is not
24810 required to be reentrant.

24811 **RETURN VALUE**

24812 Upon successful completion, *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* return a
24813 pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of
24814 the database was reached or the requested entry was not found. Otherwise, a null pointer is
24815 returned.

24816 **ERRORS**

24817 No errors are defined.

endprotoent()

24818

EXAMPLES

24819

None.

24820

APPLICATION USAGE

24821

The *getprotobyname()*, *getprotobynumber()*, and *getprotoent()* functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

24822

24823

RATIONALE

24824

None.

24825

FUTURE DIRECTIONS

24826

None.

24827

SEE ALSO

24828

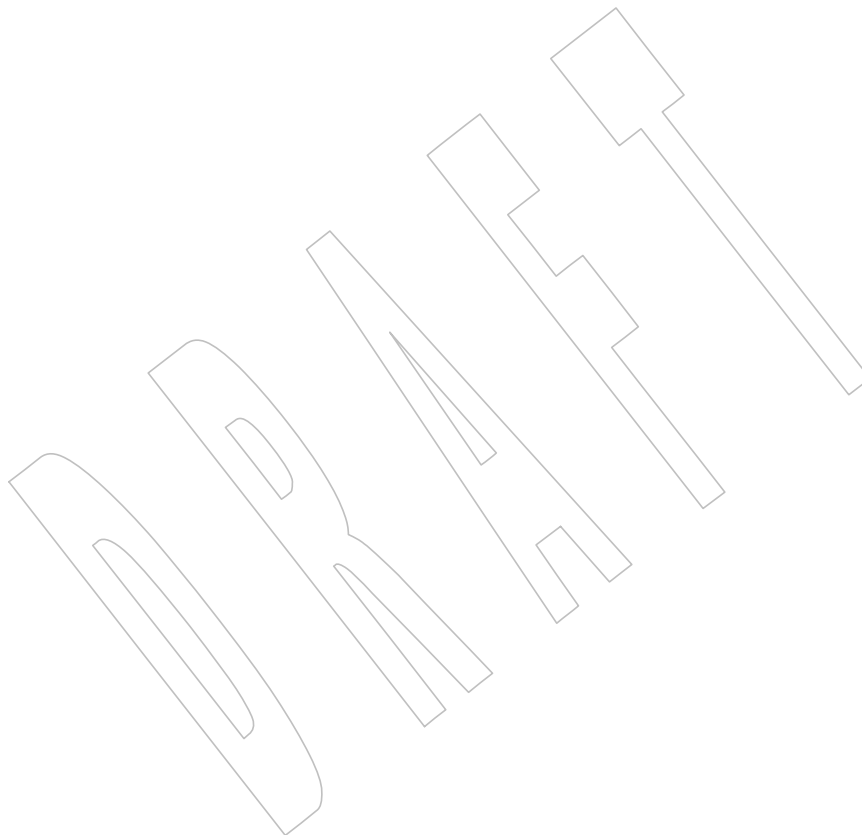
XBD <[netdb.h](#)>

24829

CHANGE HISTORY

24830

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



24831 **NAME**
 24832 endpwent, getpwent, setpwent — user database functions

24833 **SYNOPSIS**

```
24834 XSI #include <pwd.h>
24835 void endpwent(void);
24836 struct passwd *getpwent(void);
24837 void setpwent(void);
```

24838 **DESCRIPTION**

24839 These functions shall retrieve information about users.

24840 The *getpwent()* function shall return a pointer to a structure containing the broken-out fields of
 24841 an entry in the user database. Each entry in the user database contains a **passwd** structure. When
 24842 first called, *getpwent()* shall return a pointer to a **passwd** structure containing the first entry in
 24843 the user database. Thereafter, it shall return a pointer to a **passwd** structure containing the next
 24844 entry in the user database. Successive calls can be used to search the entire user database.

24845 If an end-of-file or an error is encountered on reading, *getpwent()* shall return a null pointer.

24846 An implementation that provides extended security controls may impose further
 24847 implementation-defined restrictions on accessing the user database. In particular, the system
 24848 may deny the existence of some or all of the user database entries associated with users other
 24849 than the caller.

24850 The *setpwent()* function effectively rewinds the user database to allow repeated searches.

24851 The *endpwent()* function may be called to close the user database when processing is complete.

24852 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 24853 required to be reentrant.

24854 **RETURN VALUE**

24855 The *getpwent()* function shall return a null pointer on end-of-file or error.

24856 **ERRORS**

24857 The *getpwent()*, *setpwent()*, and *endpwent()* functions may fail if:

24858 [EIO] An I/O error has occurred.

24859 In addition, *getpwent()* and *setpwent()* may fail if:

24860 [EMFILE] All file descriptors available to the process are currently open.

24861 [ENFILE] The maximum allowable number of files is currently open in the system.

24862 The return value may point to a static area which is overwritten by a subsequent call to
 24863 *getpwuid()*, *getpwnam()*, or *getpwent()*.

24864 **EXAMPLES**24865 **Searching the User Database**

24866 The following example uses the *getpwent()* function to get successive entries in the user
 24867 database, returning a pointer to a **passwd** structure that contains information about each user.
 24868 The call to *endpwent()* closes the user database and cleans up.

```

24869 #include <pwd.h>
24870 #include <stdio.h>

24871 void printname(uid_t uid)
24872 {
24873     struct passwd *pwd;

24874     setpwent();
24875     while((pwd = getpwent()) != NULL) {
24876         if (pwd->pw_uid == uid) {
24877             printf("name=%s\n", pwd->pw_name);
24878             break;
24879         }
24880     }
24881     endpwent();
24882 }

```

24883 **APPLICATION USAGE**

24884 These functions are provided due to their historical usage. Applications should avoid
 24885 dependencies on fields in the password database, whether the database is a single file, or where
 24886 in the file system name space the database resides. Applications should use *getpwuid()*
 24887 whenever possible because it avoids these dependencies.

24888 **RATIONALE**

24889 None.

24890 **FUTURE DIRECTIONS**

24891 None.

24892 **SEE ALSO**

24893 *endgrent()*, *getlogin()*, *getpwnam()*, *getpwuid()*

24894 XBD [<pwd.h>](#)

24895 **CHANGE HISTORY**

24896 First released in Issue 4, Version 2.

24897 **Issue 5**

24898 Moved from X/OPEN UNIX extension to BASE.

24899 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 24900 VALUE section.

24901 A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

24902 **Issue 6**

24903 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

24904 **Issue 7**

24905 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

24906 The EXAMPLES section is revised. +

24907 **NAME**

24908 endservent, getservbyname, getservbyport, getservent, setservent — network services database
24909 functions

24910 **SYNOPSIS**

```
24911 #include <netdb.h>

24912 void endservent(void);
24913 struct servent *getservbyname(const char *name, const char *proto);
24914 struct servent *getservbyport(int port, const char *proto);
24915 struct servent *getservent(void);
24916 void setservent(int stayopen);
```

24917 **DESCRIPTION**

24918 These functions shall retrieve information about network services. This information is
24919 considered to be stored in a database that can be accessed sequentially or randomly. The
24920 implementation of this database is unspecified.

24921 The *setservent()* function shall open a connection to the database, and set the next entry to the
24922 first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each
24923 call to the *getservent()* function (either directly, or indirectly through one of the other *getserv*()*
24924 functions), and the implementation may maintain an open file descriptor for the database.

24925 The *getservent()* function shall read the next entry of the database, opening and closing a
24926 connection to the database as necessary.

24927 The *getservbyname()* function shall search the database from the beginning and find the first
24928 entry for which the service name specified by *name* matches the *s_name* member and the protocol
24929 name specified by *proto* matches the *s_proto* member, opening and closing a connection to the
24930 database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be
24931 matched.

24932 The *getservbyport()* function shall search the database from the beginning and find the first entry
24933 for which the port specified by *port* matches the *s_port* member and the protocol name specified
24934 by *proto* matches the *s_proto* member, opening and closing a connection to the database as
24935 necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port*
24936 argument shall be a value obtained by converting a *uint16_t* in network byte order to *int*.

24937 The *getservbyname()*, *getservbyport()*, and *getservent()* functions shall each return a pointer to a
24938 **servent** structure, the members of which shall contain the fields of an entry in the network
24939 services database.

24940 The *endservent()* function shall close the database, releasing any open file descriptor.

24941 These functions need not be thread-safe. A function that is not required to be thread-safe is not
24942 required to be reentrant.

24943 **RETURN VALUE**

24944 Upon successful completion, *getservbyname()*, *getservbyport()*, and *getservent()* return a pointer to
24945 a **servent** structure if the requested entry was found, and a null pointer if the end of the database
24946 was reached or the requested entry was not found. Otherwise, a null pointer is returned.

24947 **ERRORS**

24948 No errors are defined.

endservent()

24949
24950
24951
24952
24953
24954
24955
24956
24957
24958
24959
24960
24961
24962
24963
24964
24965
24966

EXAMPLES

None.

APPLICATION USAGE

The *port* argument of *getservbyport()* need not be compatible with the port values of all address families.

The *getservbyname()*, *getservbyport()*, and *getservent()* functions may return pointers to static data, which may be overwritten by subsequent calls to any of these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endhostent(), *endprotoent()*, *htonl()*, *inet_addr()*

XBD <[netdb.h](#)>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

SD5-XBD-ERN-14 is applied.

DRAFT

24967 **NAME**
 24968 endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database
 24969 functions

24970 SYNOPSIS

```
24971 XSI #include <utmpx.h>
24972
24973 void endutxent(void);
24974 struct utmpx *getutxent(void);
24975 struct utmpx *getutxid(const struct utmpx *id);
24976 struct utmpx *getutxline(const struct utmpx *line);
24977 struct utmpx *pututxline(const struct utmpx *utmpx);
24978 void setutxent(void);
```

24978 DESCRIPTION

24979 These functions shall provide access to the user accounting database.

24980 The *getutxent()* function shall read the next entry from the user accounting database. If the
 24981 database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

24982 The *getutxid()* function shall search forward from the current point in the database. If the
 24983 *ut_type* value of the **utmpx** structure pointed to by *id* is *BOOT_TIME*, *OLD_TIME*, or
 24984 *NEW_TIME*, then it shall stop when it finds an entry with a matching *ut_type* value. If the
 24985 *ut_type* value is *INIT_PROCESS*, *LOGIN_PROCESS*, *USER_PROCESS*, or *DEAD_PROCESS*,
 24986 then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member
 24987 matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is
 24988 reached without a match, *getutxid()* shall fail.

24989 The *getutxline()* function shall search forward from the current point in the database until it
 24990 finds an entry of the type *LOGIN_PROCESS* or *USER_PROCESS* which also has a *ut_line* value
 24991 matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached
 24992 without a match, *getutxline()* shall fail.

24993 The *getutxid()* or *getutxline()* function may cache data. For this reason, to use *getutxline()* to
 24994 search for multiple occurrences, the application shall zero out the static data after each success,
 24995 or *getutxline()* may return a pointer to the same **utmpx** structure.

24996 There is one exception to the rule about clearing the structure before further reads are done. The
 24997 implicit read done by *pututxline()* (if it finds that it is not already at the correct place in the user
 24998 accounting database) shall not modify the static structure returned by *getutxent()*, *getutxid()*, or
 24999 *getutxline()*, if the application has modified this structure and passed the pointer back to
 25000 *pututxline()*.

25001 For all entries that match a request, the *ut_type* member indicates the type of the entry. Other
 25002 members of the entry shall contain meaningful data based on the value of the *ut_type* member as
 25003 follows:

25004
25005
25006
25007
25008
25009
25010
25011
25012
25013

ut_type Member	Other Members with Meaningful Data
EMPTY	No others
BOOT_TIME	<i>ut_tv</i>
OLD_TIME	<i>ut_tv</i>
NEW_TIME	<i>ut_tv</i>
USER_PROCESS	<i>ut_id, ut_user</i> (login name of the user), <i>ut_line, ut_pid, ut_tv</i>
INIT_PROCESS	<i>ut_id, ut_pid, ut_tv</i>
LOGIN_PROCESS	<i>ut_id, ut_user</i> (implementation-defined name of the login process), <i>ut_pid, ut_tv</i>
DEAD_PROCESS	<i>ut_id, ut_pid, ut_tv</i>

25014
25015
25016
25017

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

25018
25019
25020
25021

If the process has appropriate privileges, the *pututxline()* function shall write out the structure into the user accounting database. It shall use *getutxid()* to search for a record that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

25022

The *endutxent()* function shall close the user accounting database.

25023
25024

The *setutxent()* function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

25025
25026

These functions need not be thread-safe. A function that is not required to be thread-safe is not required to be reentrant.

25027

RETURN VALUE

25028
25029
25030

Upon successful completion, *getutxent()*, *getutxid()*, and *getutxline()* shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

25031
25032

The return value may point to a static area which is overwritten by a subsequent call to *getutxid()* or *getutxline()*.

25033
25034
25035

Upon successful completion, *pututxline()* shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

25036

The *endutxent()* and *setutxent()* functions shall not return a value.

25037

ERRORS

25038
25039

No errors are defined for the *endutxent()*, *getutxent()*, *getutxid()*, *getutxline()*, and *setutxent()* functions.

25040

The *pututxline()* function may fail if:

25041

[EPERM] The process does not have appropriate privileges.

25042
25043
25044
25045
25046
25047
25048
25049
25050
25051
25052
25053
25054
25055
25056
25057
25058
25059
25060

EXAMPLES

None.

APPLICATION USAGE

The sizes of the arrays in the structure can be found using the *sizeof* operator.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<utmpx.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

A note indicating that these functions need not be reentrant is added to the DESCRIPTION.

Issue 6

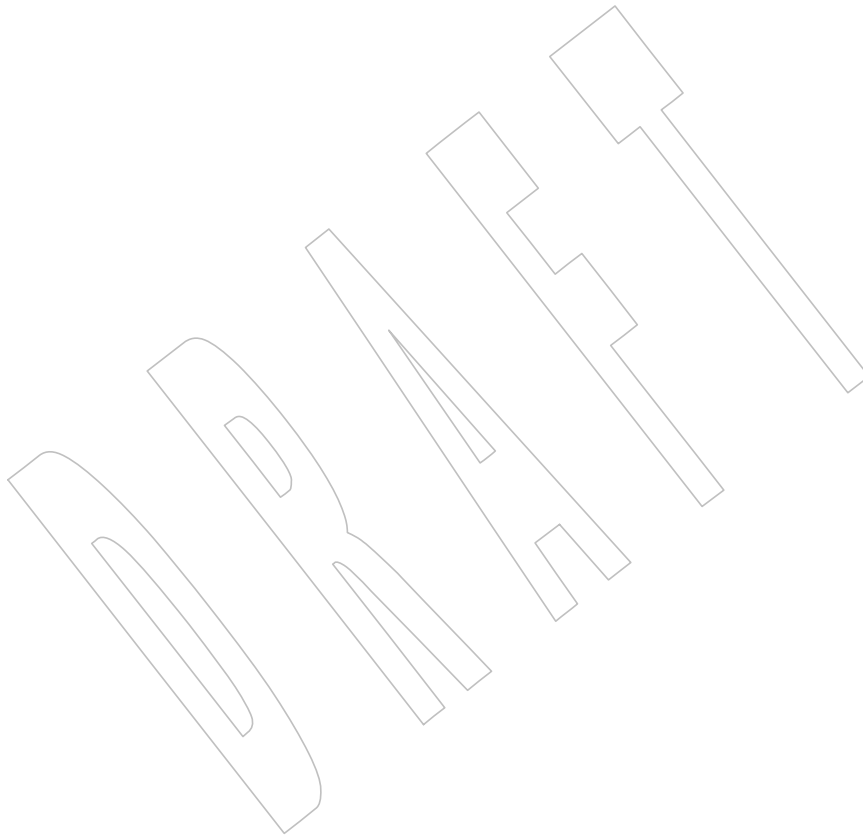
In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

DRAFT

25061 **NAME**
25062 `environ` — array of character pointers to the environment strings

25063 **SYNOPSIS**
25064 `extern char **environ;`

25065 **DESCRIPTION**
25066 Refer to XBD [Chapter 8](#) (on page 159) and *exec*.



25067 **NAME**
25068 erand48 — generate uniformly distributed pseudo-random numbers

25069 **SYNOPSIS**

```
25070 XSI       #include <stdlib.h>  
25071       double erand48(unsigned short xsubi[3]);
```

25072 **DESCRIPTION**

25073 Refer to *drand48()*.

25074 **NAME**
 25075 erf, erff, erfl — error functions

25076 **SYNOPSIS**
 25077 #include <math.h>
 25078 double erf(double x);
 25079 float erff(float x);
 25080 long double erfl(long double x);

25081 DESCRIPTION

25082 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25083 conflict between the requirements described here and the ISO C standard is unintentional. This
 25084 volume of POSIX.1-200x defers to the ISO C standard.

25085 These functions shall compute the error function of their argument x , defined as:

25086
 25087
$$\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

25088 An application wishing to check for error situations should set *errno* to zero and call
 25089 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 25090 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 25091 zero, an error has occurred.

25092 RETURN VALUE

25093 Upon successful completion, these functions shall return the value of the error function.

25094 MX If x is NaN, a NaN shall be returned.
 25095 If x is ± 0 , ± 0 shall be returned.
 25096 If x is $\pm \text{Inf}$, ± 1 shall be returned.
 25097 If x is subnormal, a range error may occur, and $2 * x / \text{sqrt}(\pi)$ should be returned.

25098 ERRORS

25099 These functions may fail if:

25100 MX **Range Error** The result underflows.
 25101 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25102 then *errno* shall be set to [ERANGE]. If the integer expression
 25103 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 25104 floating-point exception shall be raised.

25105 EXAMPLES

25106 Computing the Probability for a Normal Variate

25107 This example shows how to use *erf()* to compute the probability that a normal variate assumes a
 25108 value in the range $[x1, x2]$ with $x1 \leq x2$. This example uses the constant M_SQRT1_2 which is part
 25109 of the XSI option.

```
25110 #include <math.h>
25111 double
25112 Phi(const double x1, const double x2)
25113 {
25114     return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
```

25115 }

25116 APPLICATION USAGE

25117 Underflow occurs when $|x| < \text{DBL_MIN} * (\text{sqrt}(\pi)/2)$.

25118 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
25119 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

25120 RATIONALE

25121 None.

25122 FUTURE DIRECTIONS

25123 None.

25124 SEE ALSO

25125 *erfc()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

25126 XBD Section 4.19 (on page 104), `<math.h>`

25127 CHANGE HISTORY

25128 First released in Issue 1. Derived from Issue 1 of the SVID.

25129 Issue 5

25130 The DESCRIPTION is updated to indicate how an application should check for an error. This
25131 text was previously published in the APPLICATION USAGE section.

25132 Issue 6

25133 The *erf()* function is no longer marked as an extension.

25134 The *erfc()* function is split out onto its own reference page.

25135 The *erff()* and *erfl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

25136 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
25137 revised to align with the ISO/IEC 9899:1999 standard.

25138 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
25139 marked.

25140 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/22 is applied, adding the example to the
25141 EXAMPLES section.

25142 **NAME**25143 `erfc`, `erfcf`, `erfcl` — complementary error functions25144 **SYNOPSIS**

```
25145 #include <math.h>
25146
25146 double erfc(double x);
25147 float erfcf(float x);
25148 long double erfcl(long double x);
```

25149 **DESCRIPTION**

25150 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25151 conflict between the requirements described here and the ISO C standard is unintentional. This
 25152 volume of POSIX.1-200x defers to the ISO C standard.

25153 These functions shall compute the complementary error function $1.0 - \text{erf}(x)$.

25154 An application wishing to check for error situations should set *errno* to zero and call
 25155 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 25156 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 25157 zero, an error has occurred.

25158 **RETURN VALUE**

25159 Upon successful completion, these functions shall return the value of the complementary error
 25160 function.

25161 If the correct value would cause underflow and is not representable, a range error may occur
 25162 MX and either 0.0 (if representable), or an implementation-defined value shall be returned.

25163 MX If *x* is NaN, a NaN shall be returned.

25164 If *x* is ± 0 , +1 shall be returned.

25165 If *x* is $-\text{Inf}$, +2 shall be returned.

25166 If *x* is $+\text{Inf}$, +0 shall be returned.

25167 If the correct value would cause underflow and is representable, a range error may occur and
 25168 the correct value shall be returned.

25169 **ERRORS**

25170 These functions may fail if:

25171 Range Error The result underflows.

25172 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25173 then *errno* shall be set to [ERANGE]. If the integer expression
 25174 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 25175 floating-point exception shall be raised.

25176 **EXAMPLES**

25177 None.

25178 **APPLICATION USAGE**

25179 The *erfc()* function is provided because of the extreme loss of relative accuracy if *erf(x)* is called
 25180 for large *x* and the result subtracted from 1.0.

25181 Note for IEEE Std 754-1985 **double**, $26.55 < x$ implies *erfc(x)* has underflowed.

25182 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 25183 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

25184
25185
25186
25187
25188
25189
25190
25191
25192
25193
25194
25195
25196
25197
25198
25199
25200
25201
25202

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erf(), *feclearexcept()*, *fetetestexcept()*, *isnan()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *erfc()* function is no longer marked as an extension.

These functions are split out from the *erf()* reference page.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

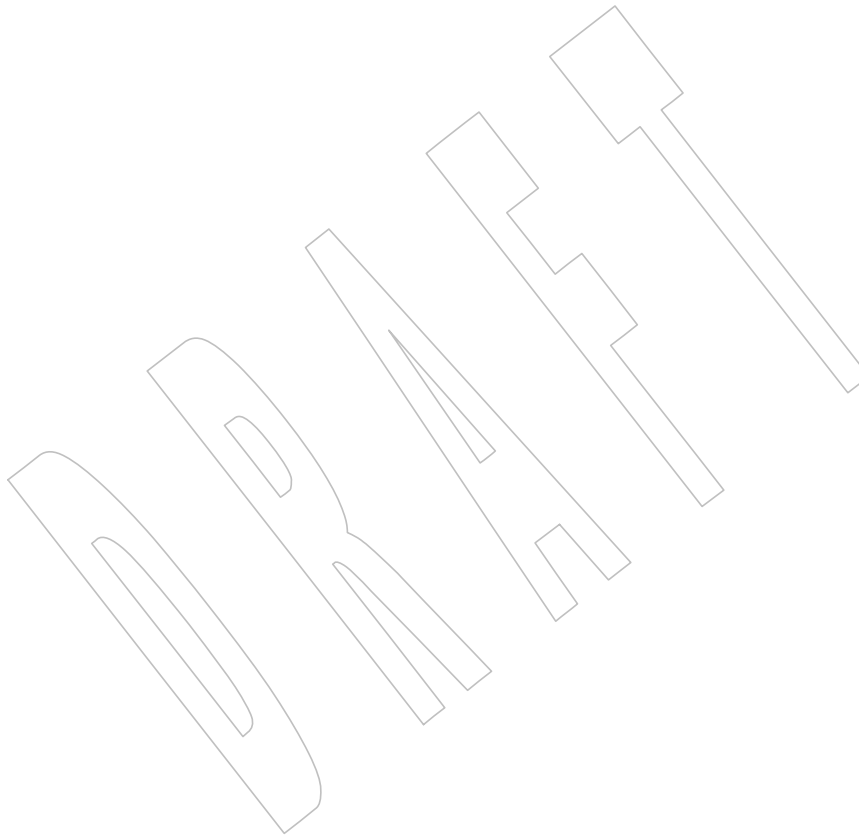
IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

DRAFT

25203 **NAME**
25204 `erff, erfl` — error functions

25205 **SYNOPSIS**
25206 `#include <math.h>`
25207 `float erff(float x);`
25208 `long double erfl(long double x);`

25209 **DESCRIPTION**
25210 Refer to *erf()*.



25211 **NAME**

25212 errno — error return value

25213 **SYNOPSIS**

25214 #include <errno.h>

25215 **DESCRIPTION**25216 The lvalue *errno* is used by many functions to return error values.

25217 Many functions provide an error number in *errno*, which has type **int** and is defined in
 25218 <**errno.h**>. The value of *errno* shall be defined only after a call to a function for which it is
 25219 explicitly stated to be set and until it is changed by the next function call or if the application
 25220 assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by
 25221 a function's return value. Applications shall obtain the definition of *errno* by the inclusion of
 25222 <**errno.h**>. No function in this volume of POSIX.1-200x shall set *errno* to 0. The setting of *errno*
 25223 after a successful call to a function is unspecified unless the description of that function specifies
 25224 that *errno* shall not be modified.

25225 It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a
 25226 macro definition is suppressed in order to access an actual object, or a program defines an
 25227 identifier with the name *errno*, the behavior is undefined.

25228 The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant
 25229 pages.

25230 **RETURN VALUE**

25231 None.

25232 **ERRORS**

25233 None.

25234 **EXAMPLES**

25235 None.

25236 **APPLICATION USAGE**

25237 Previously both POSIX and X/Open documents were more restrictive than the ISO C standard
 25238 in that they required *errno* to be defined as an external variable, whereas the ISO C standard
 25239 required only that *errno* be defined as a modifiable lvalue with type **int**.

25240 An application that needs to examine the value of *errno* to determine the error should set it to 0
 25241 before a function call, then inspect it before a subsequent function call.

25242 **RATIONALE**

25243 None.

25244 **FUTURE DIRECTIONS**

25245 None.

25246 **SEE ALSO**25247 [Section 2.3](#)25248 XBD <**errno.h**>25249 **CHANGE HISTORY**

25250 First released in Issue 1. Derived from Issue 1 of the SVID.

25251

Issue 5

25252

25253

25254

The following sentence is deleted from the DESCRIPTION: “The value of *errno* is 0 at program start-up, but is never set to 0 by any XSI function”. The DESCRIPTION also no longer states that conforming implementations may support the declaration:

25255

```
extern int errno;
```

25256

Issue 6

25257

Obsolescent text regarding defining *errno* as:

25258

```
extern int errno
```

25259

is removed.

25260

Text regarding no function setting *errno* to zero to indicate an error is changed to no function shall set *errno* to zero. This is for alignment with the ISO/IEC 9899:1999 standard.

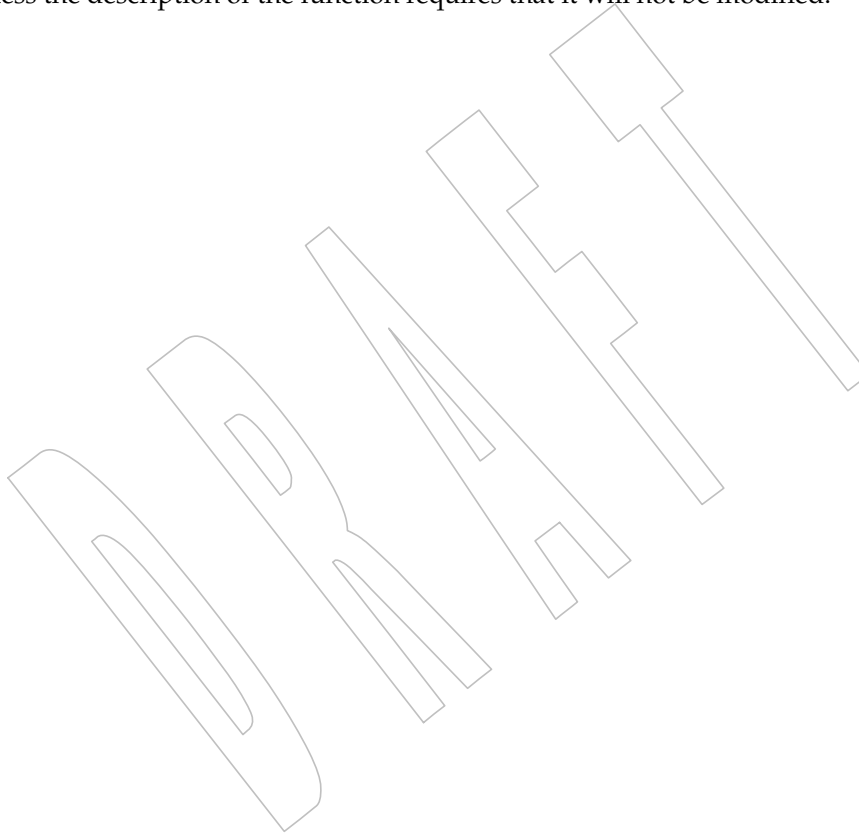
25261

25262

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/23 is applied, adding text to the DESCRIPTION stating that the setting of *errno* after a successful call to a function is unspecified unless the description of the function requires that it will not be modified.

25263

25264



25265 **NAME**

25266 environ, execl, execv, execl, execve, execlp, execvp, fexecve — execute a file

25267 **SYNOPSIS**

```

25268 #include <unistd.h>
25269
25269 extern char **environ;
25270 int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
25271 int execl(const char *path, const char *arg0, ... /*,
25272          (char *)0, char *const envp[]*/);
25273 int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
25274 int execv(const char *path, char *const argv[]);
25275 int execve(const char *path, char *const argv[], char *const envp[]);
25276 int execvp(const char *file, char *const argv[]);
25277 int fexecve(int fd, char *const argv[], char *const envp[]);

```

25278 **DESCRIPTION**

25279 The *exec* family of functions shall replace the current process image with a new process image.
 25280 The new image shall be constructed from a regular, executable file called the *new process image*
 25281 *file*. There shall be no return from a successful *exec*, because the calling process image is overlaid
 25282 by the new process image.

25283 The *fexecve()* function shall be equivalent to the *execve()* function except that the file to be
 25284 executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is
 25285 ignored.

25286 When a C-language program is executed as a result of a call to one of the *exec* family of
 25287 functions, it shall be entered as a C-language function call as follows:

```
25288 int main (int argc, char *argv[]);
```

25289 where *argc* is the argument count and *argv* is an array of character pointers to the arguments
 25290 themselves. In addition, the following variable:

```
25291 extern char **environ;
```

25292 is initialized as a pointer to an array of character pointers to the environment strings. The *argv*
 25293 and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv*
 25294 array is not counted in *argc*.

25295 Conforming multi-threaded applications shall not use the *environ* variable to access or modify
 25296 any environment variable while any other thread is concurrently modifying any environment
 25297 variable. A call to any function dependent on any environment variable shall be considered a
 25298 use of the *environ* variable to access that environment variable.

25299 The arguments specified by a program with one of the *exec* functions shall be passed on to the
 25300 new process image in the corresponding *main()* arguments.

25301 The argument *path* points to a pathname that identifies the new process image file.

25302 The argument *file* is used to construct a pathname that identifies the new process image file. If
 25303 the *file* argument contains a slash character, the *file* argument shall be used as the pathname for
 25304 this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed
 25305 as the environment variable *PATH* (see XBD Chapter 8, on page 159). If this environment
 25306 variable is not present, the results of the search are implementation-defined.

25307 There are two distinct ways in which the contents of the process image file may cause the
 25308 execution to fail, distinguished by the setting of *errno* to either [ENOEXEC] or [EINVAL] (see the
 25309 ERRORS section). In the cases where the other members of the *exec* family of functions would

25310 fail and set *errno* to [ENOEXEC], the *execlp()* and *execvp()* functions shall execute a command
 25311 interpreter and the environment of the executed command shall be as if the process invoked the
 25312 *sh* utility using *execl()* as follows:

```
25313 execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

25314 where *<shell path>* is an unspecified pathname for the *sh* utility, *file* is the process image file, and
 25315 for *execvp()*, where *arg0*, *arg1*, and so on correspond to the values passed to *execvp()* in *argv*[0],
 25316 *argv*[1], and so on.

25317 The arguments represented by *arg0*,... are pointers to null-terminated character strings. These
 25318 strings shall constitute the argument list available to the new process image. The list is
 25319 terminated by a null pointer. The argument *arg0* should point to a filename that is associated
 25320 with the process being started by one of the *exec* functions.

25321 The argument *argv* is an array of character pointers to null-terminated strings. The application
 25322 shall ensure that the last member of this array is a null pointer. These strings shall constitute the
 25323 argument list available to the new process image. The value in *argv*[0] should point to a filename
 25324 that is associated with the process being started by one of the *exec* functions.

25325 The argument *envp* is an array of character pointers to null-terminated strings. These strings
 25326 shall constitute the environment for the new process image. The *envp* array is terminated by a
 25327 null pointer.

25328 For those forms not containing an *envp* pointer (*execl()*, *execv()*, *execlp()*, and *execvp()*), the
 25329 environment for the new process image shall be taken from the external variable *environ* in the
 25330 calling process.

25331 The number of bytes available for the new process' combined argument and environment lists is
 25332 {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any
 25333 alignment bytes are included in this total.

25334 File descriptors open in the calling process image shall remain open in the new process image,
 25335 except for those whose close-on-exec flag FD_CLOEXEC is set. For those file descriptors that
 25336 remain open, all attributes of the open file description remain unchanged. For any file descriptor
 25337 that is closed for this reason, file locks are removed as a result of the close as described in *close()*.
 25338 Locks that are not removed by closing of file descriptors remain unchanged.

25339 If file descriptors 0, 1, and 2 would otherwise be closed after a successful call to one of the *exec*
 25340 family of functions, and the new process image file has the set-user-ID or set-group-ID file mode
 25341 XSI bits set, and the ST_NOSUID bit is not set for the file system containing the new process image
 25342 file, implementations may open an unspecified file for each of these file descriptors in the new
 25343 process image.

25344 Directory streams open in the calling process image shall be closed in the new process image.

25345 The state of the floating-point environment in the initial thread of the new process image shall
 25346 be set to the default.

25347 The state of conversion descriptors and message catalog descriptors in the new process image is
 25348 undefined.

25349 For the new process image, the equivalent of:

```
25350 setlocale(LC_ALL, "C")
```

25351 shall be executed at start-up.

25352 Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default
 25353 action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by
 25354 the calling process image shall be set to be ignored by the new process image. Signals set to be
 25355 caught by the calling process image shall be set to the default action in the new process image

- 25356 (see <signal.h>).
- 25357 If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether
25358 the SIGCHLD signal is set to be ignored or to the default action in the new process image.
- 25359 XSI After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and
25360 the SA_ONSTACK flag shall be cleared for all signals.
- 25361 After a successful call to any of the *exec* functions, any functions previously registered by the
25362 *atexit()* or *pthread_atfork()* functions are no longer registered.
- 25363 XSI If the ST_NOSUID bit is set for the file system containing the new process image file, then the
25364 effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged
25365 in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is
25366 set, the effective user ID of the new process image shall be set to the user ID of the new process
25367 image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the
25368 effective group ID of the new process image shall be set to the group ID of the new process
25369 image file. The real user ID, real group ID, and supplementary group IDs of the new process
25370 image shall remain the same as those of the calling process image. The effective user ID and
25371 effective group ID of the new process image shall be saved (as the saved set-user-ID and the
25372 saved set-group-ID) for use by *setuid()*.
- 25373 XSI Any shared memory segments attached to the calling process image shall not be attached to the
25374 new process image.
- 25375 Any named semaphores open in the calling process shall be closed as if by appropriate calls to
25376 *sem_close()*.
- 25377 TYM Any blocks of typed memory that were mapped in the calling process are unmapped, as if
25378 *munmap()* was implicitly called to unmap them.
- 25379 ML Memory locks established by the calling process via calls to *mlockall()* or *mlock()* shall be
25380 removed. If locked pages in the address space of the calling process are also mapped into the
25381 address spaces of other processes and are locked by those processes, the locks established by the
25382 other processes shall be unaffected by the call by this process to the *exec* function. If the *exec*
25383 function fails, the effect on memory locks is unspecified.
- 25384 Memory mappings created in the process are unmapped before the address space is rebuilt for
25385 the new process image.
- 25386 SS When the calling process image does not use the SCHED_FIFO, SCHED_RR, or
25387 SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new
25388 process image and the initial thread in that new process image are implementation-defined.
- 25389 PS When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC
25390 scheduling policies, the process policy and scheduling parameter settings shall not be changed
25391 by a call to an *exec* function. The initial thread in the new process image shall inherit the process
25392 scheduling policy and parameters. It shall have the default system contention scope, but shall
25393 inherit its allocation domain from the calling process image.
- 25394 Per-process timers created by the calling process shall be deleted before replacing the current
25395 process image with the new process image.
- 25396 MSG All open message queue descriptors in the calling process shall be closed, as described in
25397 *mq_close()*.
- 25398 Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O
25399 operations that are not canceled shall complete as if the *exec* function had not yet occurred, but
25400 any associated signal notifications shall be suppressed. It is unspecified whether the *exec*
25401 function itself blocks awaiting such I/O completion. In no event, however, shall the new process
25402 image created by the *exec* function be affected by the presence of outstanding asynchronous I/O

25403		operations at the time the <i>exec</i> function is called. Whether any I/O is canceled, and which I/O
25404		may be canceled upon <i>exec</i> , is implementation-defined.
25405	CPT	The new process image shall inherit the CPU-time clock of the calling process image. This
25406		inheritance means that the process CPU-time clock of the process being <i>exec</i> -ed shall not be
25407		reinitialized or altered as a result of the <i>exec</i> function other than to reflect the time spent by the
25408		process executing the <i>exec</i> function itself.
25409	TCT	The initial value of the CPU-time clock of the initial thread of the new process image shall be set
25410		to zero.
25411	OB TRC	If the calling process is being traced, the new process image shall continue to be traced into the
25412		same trace stream as the original process image, but the new process image shall not inherit the
25413		mapping of trace event names to trace event type identifiers that was defined by calls to the
25414		<i>posix_trace_eventid_open()</i> or the <i>posix_trace_trid_eventid_open()</i> functions in the calling process
25415		image.
25416		If the calling process is a trace controller process, any trace streams that were created by the
25417		calling process shall be shut down as described in the <i>posix_trace_shutdown()</i> function.
25418		The thread ID of the initial thread in the new process image is unspecified.
25419		The size and location of the stack on which the initial thread in the new process image runs is
25420		unspecified.
25421		The initial thread in the new process image shall have its cancellation type set to
25422		PTHREAD_CANCEL_DEFERRED and its cancellation state set to
25423		PTHREAD_CANCEL_ENABLED.
25424		The initial thread in the new process image shall have all thread-specific data values set to
25425		NULL and all thread-specific data keys shall be removed by the call to <i>exec</i> without running
25426		destructors.
25427		The initial thread in the new process image shall be joinable, as if created with the <i>detachstate</i>
25428		attribute set to PTHREAD_CREATE_JOINABLE.
25429		The new process shall inherit at least the following attributes from the calling process image:
25430	XSI	• Nice value (see <i>nice()</i>)
25431	XSI	• <i>semadj</i> values (see <i>semop()</i>)
25432		• Process ID
25433		• Parent process ID
25434		• Process group ID
25435		• Session membership
25436		• Real user ID
25437		• Real group ID
25438		• Supplementary group IDs
25439		• Time left until an alarm clock signal (see <i>alarm()</i>)
25440		• Current working directory
25441		• Root directory
25442		• File mode creation mask (see <i>umask()</i>)

- 25443 XSI • File size limit (see `getrlimit()` and `setrlimit()`)
- 25444 • Process signal mask (see `pthread_sigmask()`)
- 25445 • Pending signal (see `sigpending()`)
- 25446 • `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times()`)
- 25447 XSI • Resource limits
- 25448 • Controlling terminal
- 25449 XSI • Interval timers
- 25450 The initial thread of the new process shall inherit at least the following attributes from the
- 25451 calling thread:
- 25452 • Signal mask (see `sigprocmask()` and `pthread_sigmask()`)
- 25453 • Pending signals (see `sigpending()`)
- 25454 All other process attributes defined in this volume of POSIX.1-200x shall be inherited in the new
- 25455 process image from the old process image. All other thread attributes defined in this volume of
- 25456 POSIX.1-200x shall be inherited in the initial thread in the new process image from the calling
- 25457 thread in the old process image. The inheritance of process or thread attributes not defined by
- 25458 this volume of POSIX.1-200x is implementation-defined.
- 25459 A call to any *exec* function from a process with more than one thread shall result in all threads
- 25460 being terminated and the new executable image being loaded and executed. No destructor
- 25461 functions or cleanup handlers shall be called.
- 25462 Upon successful completion, the *exec* functions shall mark for update the last data access
- 25463 timestamp of the file. If an *exec* function failed but was able to locate the process image file,
- 25464 whether the last data access timestamp is marked for update is unspecified. Should the *exec*
- 25465 function succeed, the process image file shall be considered to have been opened with `open()`.
- 25466 The corresponding `close()` shall be considered to occur at a time after this open, but before
- 25467 process termination or successful completion of a subsequent call to one of the *exec* functions,
- 25468 `posix_spawn()`, or `posix_spawnnp()`. The `argv[]` and `envp[]` arrays of pointers and the strings to
- 25469 which those arrays point shall not be modified by a call to one of the *exec* functions, except as a
- 25470 consequence of replacing the process image.
- 25471 XSI The saved resource limits in the new process image are set to be a copy of the process'
- 25472 corresponding hard and soft limits.

RETURN VALUE

- 25473 If one of the *exec* functions returns to the calling process image, an error has occurred; the return
- 25474 value shall be `-1`, and `errno` shall be set to indicate the error.

ERRORS

- 25476 The *exec* functions shall fail if:
- 25477
- 25478 [E2BIG] The number of bytes used by the new process image's argument list and
- 25479 environment list is greater than the system-imposed limit of `{ARG_MAX}`
- 25480 bytes.
- 25481 [EACCES] Search permission is denied for a directory listed in the new process image
- 25482 file's path prefix, or the new process image file denies execution permission,
- 25483 or the new process image file is not a regular file and the implementation does
- 25484 not support execution of files of its type.
- 25485 [EINVAL] The new process image file has the appropriate permission and has a
- 25486 recognized executable binary format, but the system does not support
- 25487 execution of a file with this format.

- 25488 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path* or *file*
25489 argument.
- 25490 [ENAMETOOLONG]
25491 The length of the *path* or *file* arguments exceeds {PATH_MAX} or a pathname
25492 component is longer than {NAME_MAX}.
- 25493 [ENOENT] A component of *path* or *file* does not name an existing file or *path* or *file* is an
25494 empty string.
- 25495 [ENOTDIR] A component of the new process image file's path prefix is not a directory.
25496 The *exec* functions, except for *execlp()* and *execvp()*, shall fail if:
- 25497 [ENOEXEC] The new process image file has the appropriate access permission but has an
25498 unrecognized format.
- 25499 The *fexecve()* function shall fail if:
- 25500 [EBADF] The *fd* argument is not a valid file descriptor open for executing.
25501 The *exec* functions may fail if:
- 25502 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
25503 resolution of the *path* or *file* argument.
- 25504 [ENAMETOOLONG]
25505 As a result of encountering a symbolic link in resolution of the *path* argument,
25506 the length of the substituted pathname string exceeded {PATH_MAX}.
- 25507 [ENOMEM] The new process image requires more memory than is allowed by the
25508 hardware or system-imposed memory management constraints.
- 25509 [ETXTBSY] The new process image file is a pure procedure (shared text) file that is
25510 currently open for writing by some process.

EXAMPLES

Using *execl()*

The following example executes the *ls* command, specifying the pathname of the executable (*/bin/ls*) and using arguments supplied directly to the command to produce single-column output.

```
#include <unistd.h>
int ret;
...
ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

Using *execle()*

The following example is similar to [Using *execl\(\)*](#). In addition, it specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>
int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

25528

Using execlp()

25529

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable.

25530

25531

```
#include <unistd.h>
```

25532

```
int ret;
```

25533

```
...
```

25534

```
ret = execlp ("ls", "ls", "-l", (char *)0);
```

25535

Using execv()

25536

The following example passes arguments to the *ls* command in the *cmd* array.

25537

```
#include <unistd.h>
```

25538

```
int ret;
```

25539

```
char *cmd[] = { "ls", "-l", (char *)0 };
```

25540

```
...
```

25541

```
ret = execv ("/bin/ls", cmd);
```

25542

Using execve()

25543

The following example passes arguments to the *ls* command in the *cmd* array, and specifies the environment for the new process image using the *env* argument.

25544

25545

```
#include <unistd.h>
```

25546

```
int ret;
```

25547

```
char *cmd[] = { "ls", "-l", (char *)0 };
```

25548

```
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
```

25549

```
...
```

25550

```
ret = execve ("/bin/ls", cmd, env);
```

25551

Using execvp()

25552

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable, and passes arguments to the *ls* command in the *cmd* array.

25553

25554

25555

```
#include <unistd.h>
```

25556

```
int ret;
```

25557

```
char *cmd[] = { "ls", "-l", (char *)0 };
```

25558

```
...
```

25559

```
ret = execvp ("ls", cmd);
```

25560

APPLICATION USAGE

25561

As the state of conversion descriptors and message catalog descriptors in the new process image is undefined, conforming applications should not rely on their use and should close them prior to calling one of the *exec* functions.

25562

25563

25564

Applications that require other than the default POSIX locale should call *setlocale()* with the appropriate parameters to establish the locale of the new process.

25565

25566

The *environ* array should not be accessed directly by the application.

25567

The new process might be invoked in a non-conforming environment if the *envp* array does not contain implementation-defined variables required by the implementation to provide a conforming environment. See the *_CS_V7_ENV* entry in **<unistd.h>** and *confstr()* for details.

25568

25569

25570 Applications should not depend on file descriptors 0, 1, and 2 being closed after an *exec*. A
25571 future version may allow these file descriptors to be automatically opened for any process.

25572 If an application wants to perform a checksum test of the file being executed before executing it,
25573 the file will need to be opened with read permission to perform the checksum test.

25574 Since execute permission is checked by *fexecve()*, the file description *fd* need not have been
25575 opened with the O_EXEC flag. However, if the file to be executed denies read and write
25576 permission for the process preparing to do the *exec*, the only way to provide the *fd* to *fexecve()*
25577 will be to use the O_EXEC flag when opening *fd*. In this case, the application will not be able to
25578 perform a checksum test since it will not be able to read the contents of the file.

25579 Note that when a file descriptor is opened with O_RDONLY, O_RDWR, or O_WRONLY mode,
25580 the file descriptor can be used to read, read and write, or write the file, respectively, even if the
25581 mode of the file changes after the file was opened. Using the O_EXEC open mode is different;
25582 *fexecve()* will ignore the mode that was used when the file descriptor was opened and the *exec*
25583 will fail if the mode of the file associated with *fd* does not grant execute permission to the calling
25584 process at the time *fexecve()* is called.

25585 RATIONALE

25586 Early proposals required that the value of *argc* passed to *main()* be “one or greater”. This was
25587 driven by the same requirement in drafts of the ISO C standard. In fact, historical
25588 implementations have passed a value of zero when no arguments are supplied to the caller of
25589 the *exec* functions. This requirement was removed from the ISO C standard and subsequently
25590 removed from this volume of POSIX.1-200x as well. The wording, in particular the use of the
25591 word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to
25592 the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an
25593 application. In fact, this is good practice, since many existing applications reference *argv*[0]
25594 without first checking the value of *argc*.

25595 The requirement on a Strictly Conforming POSIX Application also states that the value passed as
25596 the first argument be a filename associated with the process being started. Although some
25597 existing applications pass a pathname rather than a filename in some circumstances, a filename
25598 is more generally useful, since the common usage of *argv*[0] is in printing diagnostics. In some
25599 cases the filename passed is not the actual filename of the file; for example, many
25600 implementations of the *login* utility use a convention of prefixing a hyphen ('-') to the actual
25601 filename, which indicates to the command interpreter being invoked that it is a “login shell”.

25602 Historically there have been two ways that implementations can *exec* shell scripts.

25603 One common historical implementation is that the *execl()*, *execv()*, *execle()*, and *execve()*
25604 functions return an [ENOEXEC] error for any file not recognizable as executable, including a
25605 shell script. When the *execlp()* and *execvp()* functions encounter such a file, they assume the file
25606 to be a shell script and invoke a known command interpreter to interpret such files. This is now
25607 required by POSIX.1-200x. These implementations of *execvp()* and *execlp()* only give the
25608 [ENOEXEC] error in the rare case of a problem with the command interpreter’s executable file.
25609 Because of these implementations, the [ENOEXEC] error is not mentioned for *execlp()* or
25610 *execvp()*, although implementations can still give it.

25611 Another way that some historical implementations handle shell scripts is by recognizing the first
25612 two bytes of the file as the character string “#!” and using the remainder of the first line of the
25613 file as the name of the command interpreter to execute.

25614 One potential source of confusion noted by the standard developers is over how the contents of
25615 a process image file affect the behavior of the *exec* family of functions. The following is a
25616 description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate permission) for this system, then the system executes the file.
2. If the process image file has appropriate permission and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to [EINVAL] (see later RATIONALE on [EINVAL]).
3. If the process image file has appropriate permission but is not otherwise recognized:
 - a. If this is a call to *execlp()* or *execvp()*, then they invoke a command interpreter assuming that the process image file is a shell script.
 - b. If this is not a call to *execlp()* or *execvp()*, then an error occurs and *errno* is set to [ENOEXEC].

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main()* called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main()* with two arguments, so implementations must support applications written this way. Since this volume of POSIX.1-200x defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv()* function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of POSIX.1-200x specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the new process image inherits the signal mask of the thread that called *exec* in the old process image. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *execs* without explicit reason to do so, and especially not to block signals across *execs* of arbitrary (not closely co-operating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv[]* and *envp[]* being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const-qualification* for the *argv[]* and *envp[]* parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst=src* derived from the ISO C standard summarizes the compatibility:

25661
25662
25663
25664
25665
25666

<i>dst:</i>	char *[]	const char *[]	char *const[]	const char *const[]
<i>src:</i>				
char *[]	VALID	—	VALID	—
const char *[]	—	VALID	—	VALID
char * const []	—	—	VALID	—
const char *const[]	—	—	—	VALID

25667
25668
25669
25670
25671

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

25672
25673
25674
25675
25676
25677
25678

The ISO C standard and this volume of POSIX.1-200x do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork()*), it conforms to both standards. A library can contain *fork()*, but if there is a user-provided *fork()*, that *fork()* is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of POSIX.1-200x is to be used if there is no user-provided *environ* to take precedence. At least three implementations are known to exist that solve this problem.

25679
25680

[E2BIG] The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.

25681
25682

[EFAULT] Some historical systems return [EFAULT] rather than [ENOEXEC] when the new process image file is corrupted. They are non-conforming.

25683
25684
25685
25686
25687
25688
25689
25690
25691
25692
25693

[EINVAL] This error condition was added to POSIX.1-200x to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, *execvp()*, and *execlp()* that encounter an [ENOEXEC] error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning [EINVAL] when a valid executable for a different architecture is encountered. Some historical implementations return [EINVAL] to indicate that the *path* argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:

25694
25695
25696
25697

1. The new utilization of [EINVAL] will provide some measure of utility to the user community.
2. Historical use of [EINVAL] is not acceptable in an internationalized operating environment.

25698
25699
25700
25701

[ENAMETOOLONG]

Since the file pathname may be constructed by taking elements in the *PATH* variable and putting them together with the filename, the [ENAMETOOLONG] error condition could also be reached this way.

25702
25703
25704

[ETXTBSY]

System V returns this error when the executable file is currently open for writing by some process. This volume of POSIX.1-200x neither requires nor prohibits this behavior.

25705
25706
25707

Other systems (such as System V) may return [INTR] from *exec*. This is not addressed by this volume of POSIX.1-200x, but implementations may have a window between the call to *exec* and the time that a signal could cause one of the *exec* calls to return with [INTR].

25708

An explicit statement regarding the floating-point environment (as defined in the **<fenv.h>**

header) was added to make it clear that the floating-point environment is set to its default when a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:

25713	<i>posix_spawn()</i>	Set to default.
25714	<i>fork()</i>	Inherit.
25715	<i>pthread_create()</i>	Inherit.

The purpose of the *fexecve()* function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, an function like *openat()* can be used to open a file which has been found by reading the content of a directory using *readdir()*.

25721 FUTURE DIRECTIONS

25722 None.

25723 SEE ALSO

25724 *alarm()*, *atexit()*, *chmod*, *close()*, *confstr()*, *exit()*, *fcntl()*, *fork()*, *fstatvfs()*, *getenv()*, *getitimer()*,
 25725 *getrlimit()*, *mknod()*, *mmap()*, *nice*, *open()*, *posix_spawn()*, *posix_trace_create()*, *posix_trace_event()*,
 25726 *posix_trace_eventid_equal()*, *pthread_atfork()*, *pthread_sigmask()*, *putenv()*, *readdir()*, *semop()*,
 25727 *setlocale()*, *shmat()*, *sigaction()*, *sigaltstack()*, *sigpending()*, *system()*, *times()*, *ulimit*, *umask*

25728 XBD Chapter 8 (on page 159), [<unistd.h>](#)

25729 CHANGE HISTORY

25730 First released in Issue 1. Derived from Issue 1 of the SVID.

25731 Issue 5

25732 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 25733 Threads Extension.

25734 Large File Summit extensions are added.

25735 Issue 6

25736 The following new requirements on POSIX implementations derive from alignment with the
 25737 Single UNIX Specification:

- 25738 • In the DESCRIPTION, behavior is defined for when the process image file is not a valid
 25739 executable.
- 25740 • In this version, `_POSIX_SAVED_IDS` is mandated, thus the effective user ID and effective
 25741 group ID of the new process image shall be saved (as the saved set-user-ID and the saved
 25742 set-group-ID) for use by the *setuid()* function.
- 25743 • The [ELOOP] mandatory error condition is added.
- 25744 • A second [ENAMETOOLONG] is added as an optional error condition.
- 25745 • The [ETXTBSY] optional error condition is added.

25746 The following changes were made to align with the IEEE P1003.1a draft standard:

- 25747 • The [EINVAL] mandatory error condition is added.
- 25748 • The [ELOOP] optional error condition is added.

25749 The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

25750 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics
 25751 for typed memory.

- 25752 The normative text is updated to avoid use of the term “must” for application requirements.
- 25753 The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.
- 25754 IEEE PASC Interpretation 1003.1 #132 is applied.
- 25755 The DESCRIPTION is updated to make it explicit that the floating-point environment in the new
25756 process image is set to the default.
- 25757 The DESCRIPTION and RATIONALE are updated to include clarifications of how the contents
25758 of a process image file affect the behavior of the *exec* functions.
- 25759 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/15 is applied, adding a new paragraph to
25760 the DESCRIPTION and text to the end of the APPLICATION USAGE section. This change
25761 addresses a security concern, where implementations may want to reopen file descriptors 0, 1,
25762 and 2 for programs with the set-user-id or set-group-id file mode bits calling the *exec* family of
25763 functions.
- 25764 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/24 is applied, applying changes to the
25765 DESCRIPTION, addressing which attributes are inherited by threads, and behavioral
25766 requirements for threads attributes.
- 25767 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/25 is applied, updating text in the
25768 RATIONALE from “the process signal mask be unchanged across an *exec*” to “the new process
25769 image inherits the signal mask of the thread that called *exec* in the old process image”.
- 25770 **Issue 7**
- 25771 Austin Group Interpretation 1003.1-2001 #047 is applied, adding the description of `_CS_V7_ENV`
25772 to the APPLICATION USAGE.
- 25773 The *fexecve()* function is added from The Open Group Technical Standard, 2006, Extended API
25774 Set Part 2.
- 25775 Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Threads,
25776 and Timers options is moved to the Base.
- 25777 Changes are made related to support for finegrained timestamps. +

25778 **NAME**

25779 exit — terminate a process

25780 **SYNOPSIS**

25781 #include <stdlib.h>

25782 void exit(int status);

25783 **DESCRIPTION**

25784 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25785 conflict between the requirements described here and the ISO C standard is unintentional. This
 25786 volume of POSIX.1-200x defers to the ISO C standard.

25787 CX The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only
 25788 the least significant 8 bits (that is, *status* & 0377) shall be available to a waiting parent process.

25789 The *exit()* function shall first call all functions registered by *atexit()*, in the reverse order of their
 25790 registration, except that a function is called after any previously registered functions that had
 25791 already been called at the time it was registered. Each function is called as many times as it was
 25792 registered. If, during the call to any such function, a call to the *longjmp()* function is made that
 25793 would terminate the call to the registered function, the behavior is undefined.

25794 If a function registered by a call to *atexit()* fails to return, the remaining registered functions shall
 25795 not be called and the rest of the *exit()* processing shall not be completed. If *exit()* is called more
 25796 than once, the behavior is undefined.

25797 CX The *exit()* function shall then flush all open streams with unwritten buffered data and close all
 25798 open streams. Finally, the process shall be terminated with the same consequences as described
 25799 in [Consequences of Process Termination](#) (on page 523).

25800 **RETURN VALUE**25801 The *exit()* function does not return.25802 **ERRORS**

25803 No errors are defined.

25804 **EXAMPLES**

25805 None.

25806 **APPLICATION USAGE**

25807 None.

25808 **RATIONALE**25809 See *_Exit()*.25810 **FUTURE DIRECTIONS**

25811 None.

25812 **SEE ALSO**25813 [_Exit\(\)](#), [atexit\(\)](#), [exec](#), [longjmp\(\)](#), [tmpfile\(\)](#)25814 XBD [<stdlib.h>](#)25815 **CHANGE HISTORY**25816 **Issue 7**25817 Austin Group Interpretation 1003.1-2001 #031 is applied, separating the *_Exit()* and *_exit()*
 25818 functions from the *exit()* function.

25819 Austin Group Interpretation 1003.1-2001 #085 is applied.

25820 **NAME**
 25821 `exp, expf, expl` — exponential function

25822 **SYNOPSIS**
 25823 `#include <math.h>`
 25824 `double exp(double x);`
 25825 `float expf(float x);`
 25826 `long double expl(long double x);`

25827 **DESCRIPTION**

25828 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25829 conflict between the requirements described here and the ISO C standard is unintentional. This
 25830 volume of POSIX.1-200x defers to the ISO C standard.

25831 These functions shall compute the base-*e* exponential of *x*.

25832 An application wishing to check for error situations should set *errno* to zero and call
 25833 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 25834 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 25835 zero, an error has occurred.

25836 **RETURN VALUE**

25837 Upon successful completion, these functions shall return the exponential value of *x*.

25838 If the correct value would cause overflow, a range error shall occur and *exp()*, *expf()*, and *expl()*
 25839 shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

25840 If the correct value would cause underflow, and is not representable, a range error may occur,
 25841 and either 0.0 (if supported), or an implementation-defined value shall be returned.

25842 MX If *x* is NaN, a NaN shall be returned.

25843 If *x* is ± 0 , 1 shall be returned.

25844 If *x* is $-\text{Inf}$, +0 shall be returned.

25845 If *x* is $+\text{Inf}$, *x* shall be returned.

25846 If the correct value would cause underflow, and is representable, a range error may occur and
 25847 the correct value shall be returned.

25848 **ERRORS**

25849 These functions shall fail if:

25850 Range Error The result overflows.

25851 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25852 then *errno* shall be set to [ERANGE]. If the integer expression
 25853 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 25854 floating-point exception shall be raised.

25855 These functions may fail if:

25856 Range Error The result underflows.

25857 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25858 then *errno* shall be set to [ERANGE]. If the integer expression
 25859 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 25860 floating-point exception shall be raised.

25861 **EXAMPLES**25862 **Computing the Density of the Standard Normal Distribution**

25863 This function shows an implementation for the density of the standard normal distribution
 25864 using `exp()`. This example uses the constant `M_PI` which is part of the XSI option.

```
25865 #include <math.h>
25866
25867 double
25868 normal_density (double x)
25869 {
25870     return exp(-x*x/2) / sqrt (2*M_PI);
25871 }
```

25871 **APPLICATION USAGE**

25872 Note that for IEEE Std 754-1985 **double**, $709.8 < x$ implies `exp(x)` has overflowed. The value
 25873 $x < -708.4$ implies `exp(x)` has underflowed.

25874 On error, the expressions (`math_errhandling & MATH_ERRNO`) and (`math_errhandling &`
 25875 `MATH_ERREXCEPT`) are independent of each other, but at least one of them must be non-zero.

25876 **RATIONALE**

25877 None.

25878 **FUTURE DIRECTIONS**

25879 None.

25880 **SEE ALSO**

25881 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#)

25882 XBD [Section 4.19](#) (on page 104), [<math.h>](#)

25883 **CHANGE HISTORY**

25884 First released in Issue 1. Derived from Issue 1 of the SVID.

25885 **Issue 5**

25886 The DESCRIPTION is updated to indicate how an application should check for an error. This
 25887 text was previously published in the APPLICATION USAGE section.

25888 **Issue 6**

25889 The `expf()` and `expl()` functions are added for alignment with the ISO/IEC 9899:1999 standard.

25890 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 25891 revised to align with the ISO/IEC 9899:1999 standard.

25892 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 25893 marked.

25894 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/26 is applied, adding the example to the
 25895 EXAMPLES section.

25896 **NAME**
 25897 exp2, exp2f, exp2l — exponential base 2 functions

25898 **SYNOPSIS**
 25899 #include <math.h>
 25900 double exp2(double x);
 25901 float exp2f(float x);
 25902 long double exp2l(long double x);

25903 **DESCRIPTION**

25904 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25905 conflict between the requirements described here and the ISO C standard is unintentional. This
 25906 volume of POSIX.1-200x defers to the ISO C standard.

25907 These functions shall compute the base-2 exponential of x .

25908 An application wishing to check for error situations should set *errno* to zero and call
 25909 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 25910 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 25911 zero, an error has occurred.

25912 **RETURN VALUE**

25913 Upon successful completion, these functions shall return 2^x .

25914 If the correct value would cause overflow, a range error shall occur and *exp2()*, *exp2f()*, and
 25915 *exp2l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 25916 respectively.

25917 If the correct value would cause underflow, and is not representable, a range error may occur,
 25918 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

25919 MX If x is NaN, a NaN shall be returned.

25920 If x is ± 0 , 1 shall be returned.

25921 If x is $-\text{Inf}$, $+\text{0}$ shall be returned.

25922 If x is $+\text{Inf}$, x shall be returned.

25923 If the correct value would cause underflow, and is representable, a range error may occur and
 25924 the correct value shall be returned.

25925 **ERRORS**

25926 These functions shall fail if:

25927 Range Error The result overflows.

25928 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25929 then *errno* shall be set to [ERANGE]. If the integer expression
 25930 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 25931 floating-point exception shall be raised.

25932 These functions may fail if:

25933 Range Error The result underflows.

25934 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25935 then *errno* shall be set to [ERANGE]. If the integer expression
 25936 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 25937 floating-point exception shall be raised.

25938
25939

25940
25941
25942

25943
25944

25945
25946

25947
25948

25949
25950
25951

25952
25953

EXAMPLES

None.

APPLICATION USAGE

For IEEE Std 754-1985 **double**, $1024 \leq x$ implies $\text{exp2}(x)$ has overflowed. The value $x < -1022$ implies $\text{exp}(x)$ has underflowed.

On error, the expressions $(\text{math_errhandling} \ \& \ \text{MATH_ERRNO})$ and $(\text{math_errhandling} \ \& \ \text{MATH_ERREXCEPT})$ are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[exp\(\)](#), [fclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [log\(\)](#)

XBD Section 4.19 (on page 104), [<math.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

DRAFT

25954 **NAME**
 25955 expm1, expm1f, expm1l — compute exponential functions

25956 **SYNOPSIS**
 25957 #include <math.h>
 25958 double expm1(double x);
 25959 float expm1f(float x);
 25960 long double expm1l(long double x);

25961 **DESCRIPTION**
 25962 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 25963 conflict between the requirements described here and the ISO C standard is unintentional. This
 25964 volume of POSIX.1-200x defers to the ISO C standard.

25965 These functions shall compute $e^x-1.0$.

25966 An application wishing to check for error situations should set *errno* to zero and call
 25967 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 25968 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 25969 zero, an error has occurred.

25970 **RETURN VALUE**
 25971 Upon successful completion, these functions return $e^x-1.0$.
 25972 If the correct value would cause overflow, a range error shall occur and *expm1()*, *expm1f()*, and
 25973 *expm1l()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 25974 respectively.

25975 MX If *x* is NaN, a NaN shall be returned.
 25976 If *x* is ± 0 , ± 0 shall be returned.
 25977 If *x* is $-\text{Inf}$, -1 shall be returned.
 25978 If *x* is $+\text{Inf}$, *x* shall be returned.
 25979 If *x* is subnormal, a range error may occur and *x* should be returned.

25980 **ERRORS**
 25981 These functions shall fail if:
 25982 Range Error The result overflows.
 25983 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25984 then *errno* shall be set to [ERANGE]. If the integer expression
 25985 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 25986 floating-point exception shall be raised.

25987 These functions may fail if:
 25988 MX Range Error The value of *x* is subnormal.
 25989 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 25990 then *errno* shall be set to [ERANGE]. If the integer expression
 25991 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 25992 floating-point exception shall be raised.

25993
25994
25995
25996
25997
25998
25999
26000
26001
26002
26003
26004
26005
26006
26007
26008
26009
26010
26011
26012
26013
26014
26015
26016
26017
26018
26019
26020
26021
26022

EXAMPLES

None.

APPLICATION USAGE

The value of $\text{expm1}(x)$ may be more accurate than $\text{exp}(x)-1.0$ for small values of x .

The $\text{expm1}()$ and $\text{log1p}()$ functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

$\text{expm1}(n * \text{log1p}(x)) / x$

when x is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

For IEEE Std 754-1985 **double**, $709.8 < x$ implies $\text{expm1}(x)$ has overflowed.

On error, the expressions (math_errhandling & MATH_ERRNO) and (math_errhandling & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[exp\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [ilogb\(\)](#), [log1p\(\)](#)

XBD Section 4.19 (on page 104), [<math.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The $\text{expm1f}()$ and $\text{expm1l}()$ functions are added for alignment with the ISO/IEC 9899:1999 standard.

The $\text{expm1}()$ function is no longer marked as an extension.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

26023 **NAME**
 26024 `fabs, fabsf, fabsl` — absolute value function

26025 **SYNOPSIS**
 26026 `#include <math.h>`
 26027 `double fabs(double x);`
 26028 `float fabsf(float x);`
 26029 `long double fabsl(long double x);`

26030 **DESCRIPTION**
 26031 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26032 conflict between the requirements described here and the ISO C standard is unintentional. This
 26033 volume of POSIX.1-200x defers to the ISO C standard.

26034 These functions shall compute the absolute value of their argument x , $|x|$.

26035 **RETURN VALUE**
 26036 Upon successful completion, these functions shall return the absolute value of x .

26037 MX If x is NaN, a NaN shall be returned.
 26038 If x is ± 0 , $+0$ shall be returned.
 26039 If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

26040 **ERRORS**
 26041 No errors are defined.

26042 EXAMPLES

Computing the 1-Norm of a Floating-Point Vector

26043 This example shows the use of `fabs()` to compute the 1-norm of a vector defined as follows:

26044 $\text{norm1}(v) = |v[0]| + |v[1]| + \dots + |v[n-1]|$

26045 where $|x|$ denotes the absolute value of x , n denotes the vector's dimension $v[i]$ denotes the i -th
 26046 component of v ($0 \leq i < n$).

```
26047 #include <math.h>
26048
26049 double
26050 norm1(const double v[], const int n)
26051 {
26052     int    i;
26053     double n1_v; /* 1-norm of v */
26054
26055     n1_v = 0;
26056     for (i=0; i<n; i++) {
26057         n1_v += fabs (v[i]);
26058     }
26059     return n1_v;
26060 }
```

26060 **APPLICATION USAGE**
 26061 None.

26062
26063
26064
26065
26066
26067
26068
26069
26070
26071
26072
26073
26074
26075
26076
26077
26078
26079
26080
26081

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan()

XBD <math.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

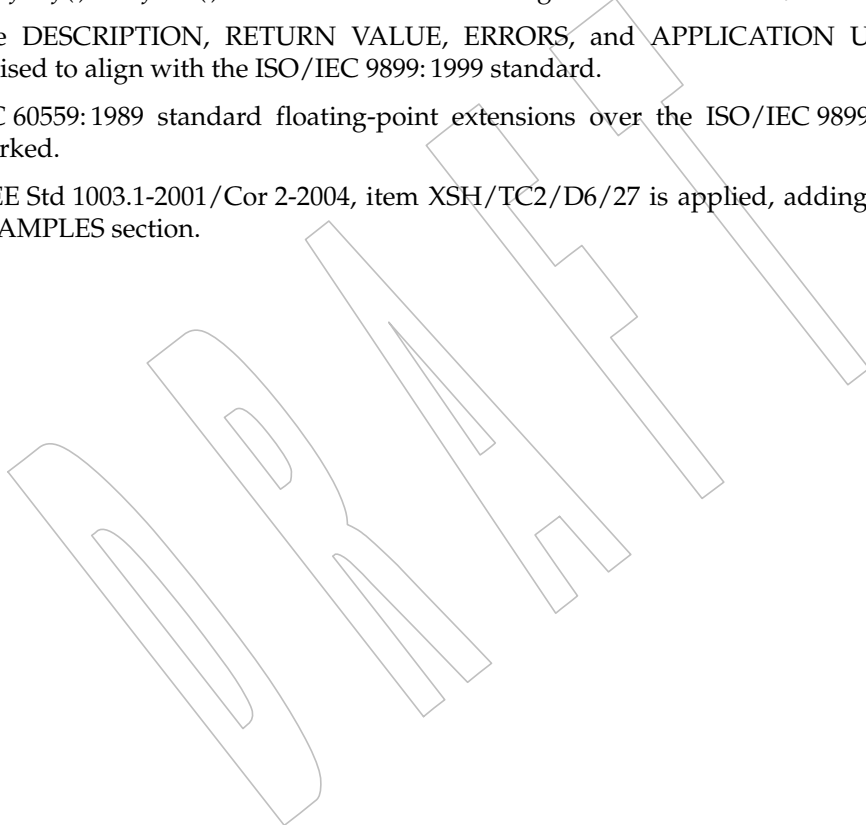
Issue 6

The *fabsf()* and *fabsl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

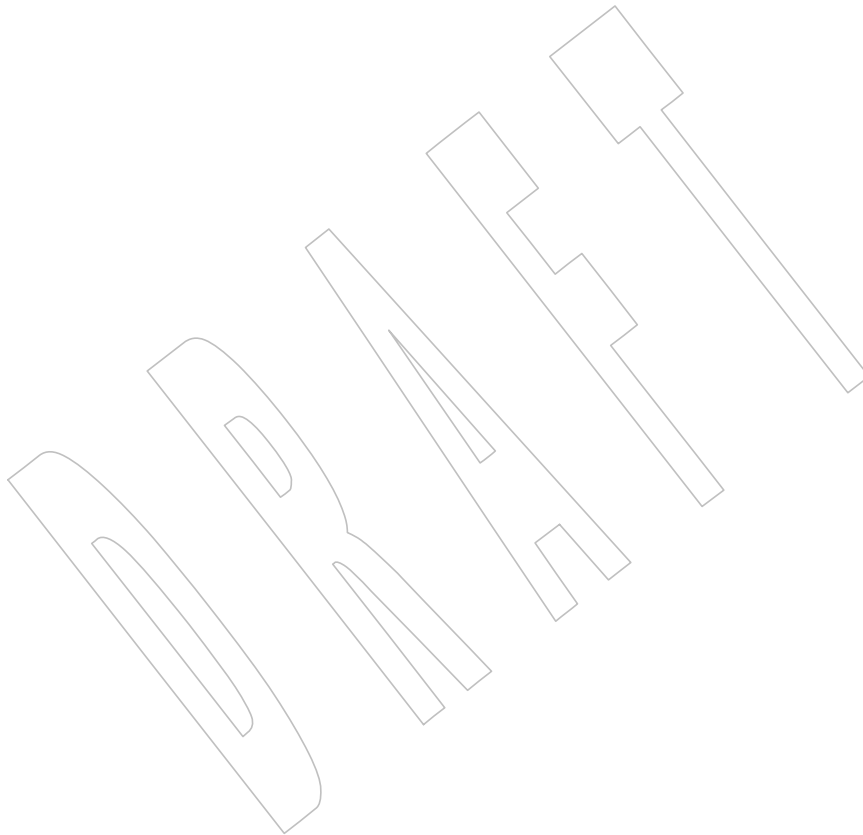
IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/27 is applied, adding the example to the EXAMPLES section.



26082 **NAME**
26083 `faccessat` — determine accessibility of a file relative to directory file descriptor

26084 **SYNOPSIS**
26085 `#include <unistd.h>`
26086 `int faccessat(int fd, const char *path, int amode, int flag);`

26087 **DESCRIPTION**
26088 Refer to [access\(\)](#).



26089 **NAME**

26090 **fattach** — attach a STREAMS-based file descriptor to a file in the file system name space
 26091 **(STREAMS)**

26092 **SYNOPSIS**

```
26093 OB XSR #include <stropts.h>
26094 int fattach(int fildes, const char *path);
```

26095 **DESCRIPTION**

26096 The *fattach()* function shall attach a STREAMS-based file descriptor to a file, effectively
 26097 associating a pathname with *fildes*. The application shall ensure that the *fildes* argument is a
 26098 valid open file descriptor associated with a STREAMS file. The *path* argument points to a
 26099 pathname of an existing file. The application shall have the appropriate privileges or be the
 26100 owner of the file named by *path* and have write permission. A successful call to *fattach()* shall
 26101 cause all pathnames that name the file named by *path* to name the STREAMS file associated with
 26102 *fildes*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more
 26103 than one file and can have several pathnames associated with it.

26104 The attributes of the named STREAMS file shall be initialized as follows: the permissions, user
 26105 ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1,
 26106 and the size and device identifier are set to those of the STREAMS file associated with *fildes*. If
 26107 any attributes of the named STREAMS file are subsequently changed (for example, by *chmod()*),
 26108 neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildes*
 26109 refers shall be affected.

26110 File descriptors referring to the underlying file, opened prior to an *fattach()* call, shall continue to
 26111 refer to the underlying file.

26112 **RETURN VALUE**

26113 Upon successful completion, *fattach()* shall return 0. Otherwise, -1 shall be returned and *errno*
 26114 set to indicate the error.

26115 **ERRORS**

26116 The *fattach()* function shall fail if:

- | | | |
|-------|----------------|---|
| 26117 | [EACCES] | Search permission is denied for a component of the path prefix, or the process is the owner of <i>path</i> but does not have write permissions on the file named by <i>path</i> . |
| 26118 | | |
| 26119 | | |
| 26120 | [EBADF] | The <i>fildes</i> argument is not a valid open file descriptor. |
| 26121 | [EBUSY] | The file named by <i>path</i> is currently a mount point or has a STREAMS file attached to it. |
| 26122 | | |
| 26123 | [ELOOP] | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument. |
| 26124 | | |
| 26125 | [ENAMETOOLONG] | The size of <i>path</i> exceeds {PATH_MAX} or a component of <i>path</i> is longer than {NAME_MAX}. |
| 26126 | | |
| 26127 | | |
| 26128 | [ENOENT] | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string. |
| 26129 | [ENOTDIR] | A component of the path prefix is not a directory. |

fattach()

- 26130 [EPERM] The effective user ID of the process is not the owner of the file named by *path*
26131 and the process does not have appropriate privilege.
- 26132 The *fattach()* function may fail if:
- 26133 [EINVAL] The *fd* argument does not refer to a STREAMS file.
- 26134 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
26135 resolution of the *path* argument.
- 26136 [ENAMETOOLONG]
26137 Pathname resolution of a symbolic link produced an intermediate result
26138 whose length exceeds {PATH_MAX}.
- 26139 [EXDEV] A link to a file on another file system was attempted.

EXAMPLES**Attaching a File Descriptor to a File**

26142 In the following example, *fd* refers to an open STREAMS file. The call to *fattach()* associates this
26143 STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-**
26144 **STREAM**, prior to breaking the attachment via a call to *fdetach()*, will instead create a new file
26145 handle referring to the STREAMS file associated with *fd*.

```
26146 #include <stropts.h>
26147 ...
26148     int fd;
26149     char *filename = "/tmp/named-STREAM";
26150     int ret;
26151
26152     ret = fattach(fd, filename);
```

APPLICATION USAGE

26153 The *fattach()* function behaves similarly to the traditional *mount()* function in the way a file is
26154 temporarily replaced by the root directory of the mounted file system. In the case of *fattach()*, the
26155 replaced file need not be a directory and the replacing file is a STREAMS file.

RATIONALE

26156 The file attributes of a file which has been the subject of an *fattach()* call are specifically set
26157 because of an artefact of the original implementation. The internal mechanism was the same as
26158 for the *mount()* function. Since *mount()* is typically only applied to directories, the effects when
26159 applied to a regular file are a little surprising, especially as regards the link count which rigidly
26160 remains one, even if there were several links originally and despite the fact that all original links
26161 refer to the STREAM as long as the *fattach()* remains in effect.

FUTURE DIRECTIONS

26163 The *fattach()* function may be removed in a future version.

SEE ALSO

26166 *fdetach()*, *isastream()*

26167 XBD **<stropts.h>**

CHANGE HISTORY

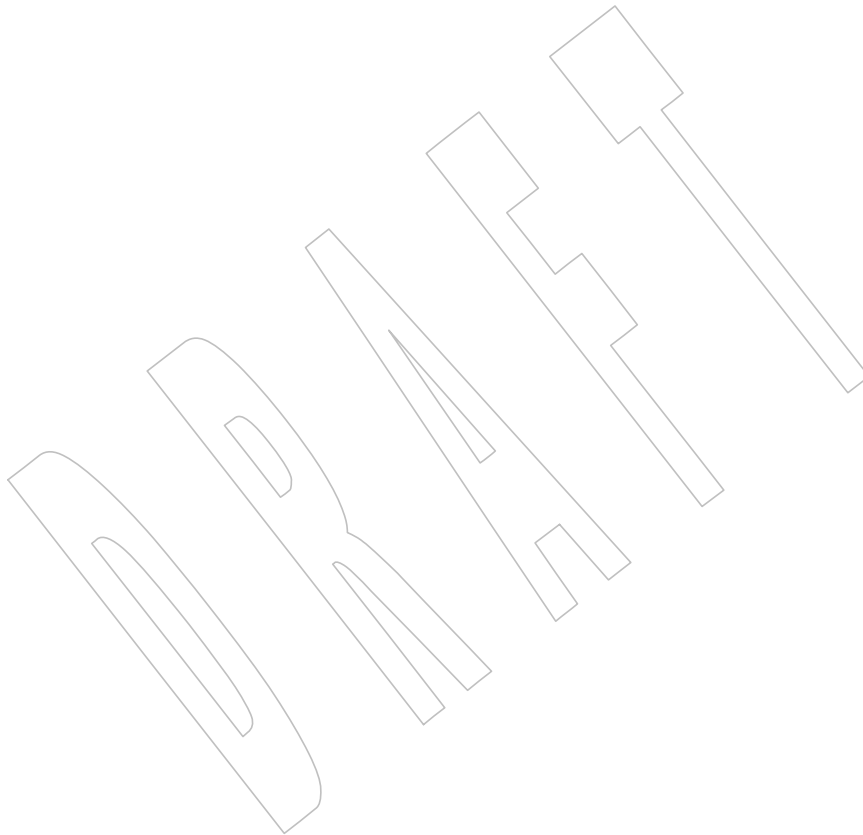
26168 First released in Issue 4, Version 2.

Issue 5

26170 Moved from X/OPEN UNIX extension to BASE.

26172 The [EXDEV] error is added to the list of optional errors in the ERRORS section.

- 26173 **Issue 6**
- 26174 This function is marked as part of the XSI STREAMS Option Group.
- 26175 The normative text is updated to avoid use of the term “must” for application requirements.
- 26176 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
- 26177 [ELOOP] error condition is added.
- 26178 **Issue 7**
- 26179 The *fattach()* function is marked obsolescent.



26180 NAME

26181 `fchdir` — change working directory

26182 SYNOPSIS

26183 `#include <unistd.h>`

26184 `int fchdir(int fildev);`

26185 DESCRIPTION

26186 The `fchdir()` function shall be equivalent to `chdir()` except that the directory that is to be the new
26187 current working directory is specified by the file descriptor *fildev*.

26188 A conforming application can obtain a file descriptor for a file of type directory using `open()`,
26189 provided that the file status flags and access modes do not contain `O_WRONLY` or `O_RDWR`.

26190 RETURN VALUE

26191 Upon successful completion, `fchdir()` shall return 0. Otherwise, it shall return `-1` and set `errno` to
26192 indicate the error. On failure the current working directory shall remain unchanged.

26193 ERRORS

26194 The `fchdir()` function shall fail if:

26195 `[EACCES]` Search permission is denied for the directory referenced by *fildev*.

26196 `[EBADF]` The *fildev* argument is not an open file descriptor.

26197 `[ENOTDIR]` The open file descriptor *fildev* does not refer to a directory.

26198 The `fchdir()` may fail if:

26199 `[EINTR]` A signal was caught during the execution of `fchdir()`.

26200 `[EIO]` An I/O error occurred while reading from or writing to the file system.

26201 EXAMPLES

26202 None.

26203 APPLICATION USAGE

26204 None.

26205 RATIONALE

26206 None.

26207 FUTURE DIRECTIONS

26208 None.

26209 SEE ALSO

26210 [`chdir\(\)`](#), [`dirfd\(\)`](#)

26211 XBD [`<unistd.h>`](#)

26212 CHANGE HISTORY

26213 First released in Issue 4, Version 2.

26214 Issue 5

26215 Moved from X/OPEN UNIX extension to BASE.

26216 Issue 7

26217 The `fchdir()` function is moved from the XSI option to the Base.

26218 **NAME**

26219 fchmod — change mode of a file

26220 **SYNOPSIS**

26221 #include <sys/stat.h>

26222 int fchmod(int *fildes*, mode_t *mode*);26223 **DESCRIPTION**26224 The *fchmod()* function shall be equivalent to *chmod()* except that the file whose permissions are
26225 changed is specified by the file descriptor *fildes*.26226 SHM If *fildes* references a shared memory object, the *fchmod()* function need only affect the S_IRUSR,
26227 S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.26228 TYM If *fildes* references a typed memory object, the behavior of *fchmod()* is unspecified.26229 If *fildes* refers to a socket, the behavior of *fchmod()* is unspecified.26230 OB XSR If *fildes* refers to a STREAM (which is *fattach()*-ed into the file system name space) the call
26231 returns successfully, doing nothing.26232 **RETURN VALUE**26233 Upon successful completion, *fchmod()* shall return 0. Otherwise, it shall return -1 and set *errno* to
26234 indicate the error.26235 **ERRORS**26236 The *fchmod()* function shall fail if:26237 [EBADF] The *fildes* argument is not an open file descriptor.26238 [EPERM] The effective user ID does not match the owner of the file and the process does
26239 not have appropriate privilege.26240 [EROFS] The file referred to by *fildes* resides on a read-only file system.26241 The *fchmod()* function may fail if:26242 XSI [EINTR] The *fchmod()* function was interrupted by a signal.26243 XSI [EINVAL] The value of the *mode* argument is invalid.26244 [EINVAL] The *fildes* argument refers to a pipe and the implementation disallows
26245 execution of *fchmod()* on a pipe.26246 **EXAMPLES**26247 **Changing the Current Permissions for a File**26248 The following example shows how to change the permissions for a file named **/home/cnd/mod1**
26249 so that the owner and group have read/write/execute permissions, but the world only has
26250 read/write permissions.

26251 #include <sys/stat.h>

26252 #include <fcntl.h>

26253 mode_t mode;

26254 int fildes;

26255 ...

26256 fildes = open("/home/cnd/mod1", O_RDWR);

26257 fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);

fchmod()26258 **APPLICATION USAGE**

26259 None.

26260 **RATIONALE**

26261 None.

26262 **FUTURE DIRECTIONS**

26263 None.

26264 **SEE ALSO**26265 *chmod, chown, creat(), fcntl(), fstatat(), fstatvfs(), mknod(), open(), read, write*

26266 XBD <sys/stat.h>

26267 **CHANGE HISTORY**

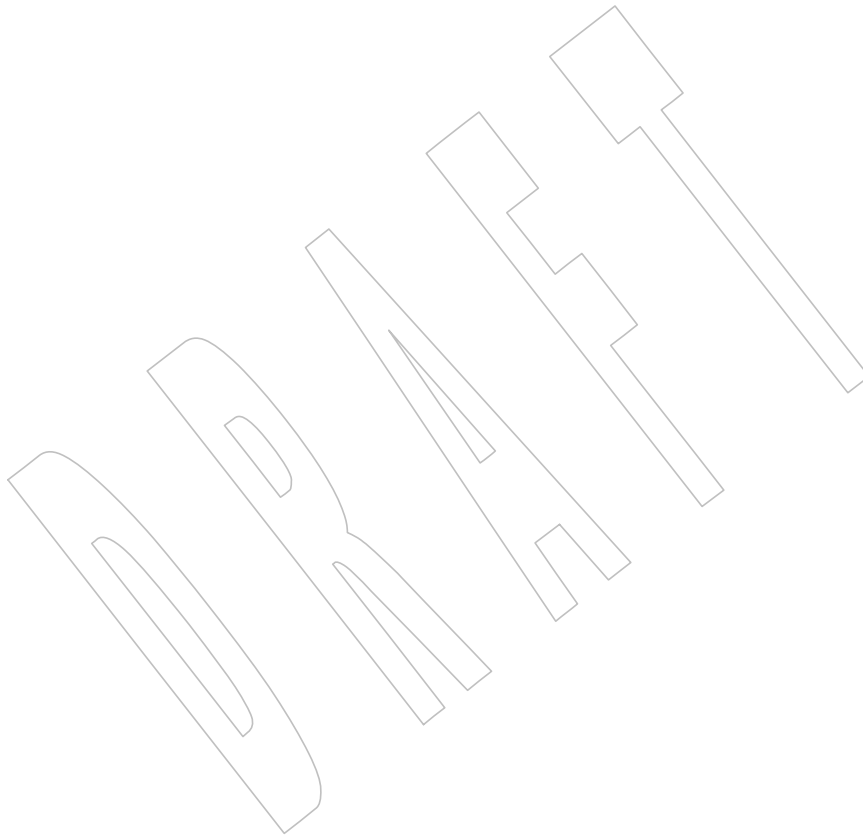
26268 First released in Issue 4, Version 2.

26269 **Issue 5**26270 Moved from X/OPEN UNIX extension to BASE and aligned with *fchmod()* in the POSIX
26271 Realtime Extension. Specifically, the second paragraph of the DESCRIPTION is added and a
26272 second instance of [EINVAL] is defined in the list of optional errors.26273 **Issue 6**26274 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by stating that *fchmod()*
26275 behavior is unspecified for typed memory objects.

26276 **NAME**
26277 fchmodat — change mode of a file relative to directory file descriptor

26278 **SYNOPSIS**
26279 #include <sys/stat.h>
26280 int fchmodat(int *fd*, const char **path*, mode_t *mode*, int *flag*);

26281 **DESCRIPTION**
26282 Refer to *chmod*.



26283 **NAME**26284 `fchown` — change owner and group of a file26285 **SYNOPSIS**26286 `#include <unistd.h>`26287 `int fchown(int fildev, uid_t owner, gid_t group);`26288 **DESCRIPTION**26289 The `fchown()` function shall be equivalent to `chown()` except that the file whose owner and group
26290 are changed is specified by the file descriptor *fildev*.26291 **RETURN VALUE**26292 Upon successful completion, `fchown()` shall return 0. Otherwise, it shall return `-1` and set *errno* to
26293 indicate the error.26294 **ERRORS**26295 The `fchown()` function shall fail if:26296 [EBADF] The *fildev* argument is not an open file descriptor.26297 [EPERM] The effective user ID does not match the owner of the file or the process does
26298 not have appropriate privilege and `_POSIX_CHOWN_RESTRICTED` indicates
26299 that such privilege is required.26300 [EROFS] The file referred to by *fildev* resides on a read-only file system.26301 The `fchown()` function may fail if:26302 [EINVAL] The owner or group ID is not a value supported by the implementation. The
26303 OB XSR *fildev* argument refers to a pipe or socket or an `fattach()`-ed `STREAM` and the
26304 implementation disallows execution of `fchown()` on a pipe.

26305 [EIO] A physical I/O error has occurred.

26306 [EINTR] The `fchown()` function was interrupted by a signal which was caught.26307 **EXAMPLES**26308 **Changing the Current Owner of a File**26309 The following example shows how to change the owner of a file named `/home/cnd/mod1` to
26310 “jones” and the group to “cnd”.26311 The numeric value for the user ID is obtained by extracting the user ID from the user database
26312 entry associated with “jones”. Similarly, the numeric value for the group ID is obtained by
26313 extracting the group ID from the group database entry associated with “cnd”. This example
26314 assumes the calling program has appropriate privileges.26315 `#include <sys/types.h>`
26316 `#include <unistd.h>`
26317 `#include <fcntl.h>`
26318 `#include <pwd.h>`
26319 `#include <grp.h>`

26320 `struct passwd *pwd;`
26321 `struct group *grp;`
26322 `int fildev;`
26323 `...`
26324 `fildev = open("/home/cnd/mod1", O_RDWR);`

```

26325     pwd = getpwnam("jones");
26326     grp = getgrnam("cnd");
26327     fchown(fildes, pwd->pw_uid, grp->gr_gid);

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown

XBD <unistd.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes were made to align with the IEEE P1003.1a draft standard:

- Clarification is added that a call to *fchown()* may not be allowed on a pipe.

The *fchown()* function is defined as mandatory.

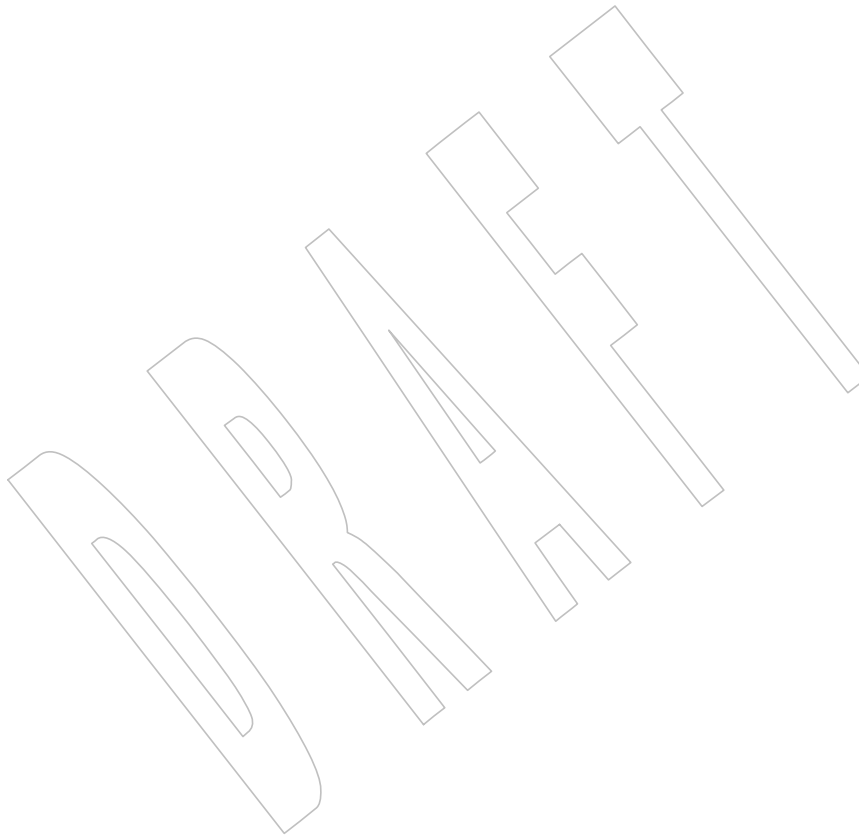
Issue 7

Functionality relating to XSI STREAMS is marked obsolescent.

26347 **NAME**
26348 fchownat — change owner and group of a file relative to directory file descriptor

26349 **SYNOPSIS**
26350 #include <unistd.h>
26351 int fchownat(int *fd*, const char **path*, uid_t *owner*, gid_t *group*,
26352 int *flag*);

26353 **DESCRIPTION**
26354 Refer to *chown*.



26355 **NAME**26356 `fclose` — close a stream26357 **SYNOPSIS**26358 `#include <stdio.h>`26359 `int fclose(FILE *stream);`26360 **DESCRIPTION**

26361 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 26362 conflict between the requirements described here and the ISO C standard is unintentional. This
 26363 volume of POSIX.1-200x defers to the ISO C standard.

26364 The `fclose()` function shall cause the stream pointed to by `stream` to be flushed and the associated
 26365 file to be closed. Any unwritten buffered data for the stream shall be written to the file; any
 26366 unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be
 26367 disassociated from the file and any buffer set by the `setbuf()` or `setvbuf()` function shall be
 26368 disassociated from the stream. If the associated buffer was automatically allocated, it shall be
 26369 deallocated.

26370 CX If the file is not already at EOF, and the file is one capable of seeking, the file offset of the
 26371 underlying open file description shall be adjusted so that the next operation on the open file
 26372 description deals with the byte after the last one read from or written to the stream being closed.

26373 The `fclose()` function shall mark for update the last data modification and last file status change
 26374 timestamps of the underlying file, if the stream was writable, and if buffered data remains that
 26375 has not yet been written to the file. The `fclose()` function shall perform the equivalent of a `close()`
 26376 on the file descriptor that is associated with the stream pointed to by `stream`.

26377 After the call to `fclose()`, any use of `stream` results in undefined behavior.

26378 **RETURN VALUE**

26379 CX Upon successful completion, `fclose()` shall return 0; otherwise, it shall return EOF and set `errno`
 26380 to indicate the error.

26381 **ERRORS**

26382 The `fclose()` function shall fail if:

26383 CX [EAGAIN] The O_NONBLOCK flag is set for the file descriptor underlying `stream` and
 26384 the thread would be delayed in the write operation.

26385 CX [EBADF] The file descriptor underlying stream is not valid.

26386 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.

26387 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
 26388 process.

26389 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
 26390 offset maximum associated with the corresponding stream.

26391 CX [EINTR] The `fclose()` function was interrupted by a signal.

26392 CX [EIO] The process is a member of a background process group attempting to write to
 26393 its controlling terminal, TOSTOP is set, the process is neither ignoring nor
 26394 blocking SIGTTOU, and the process group of the process is orphaned. This
 26395 error may also be returned under implementation-defined conditions.

fclose()

System Interfaces

26396 CX [ENOMEM] The underlying stream was created by *open_memstream()* or
26397 *open_wmemstream()* and insufficient memory is available.

26398 CX [ENOSPC] There was no free space remaining on the device containing the file or in the
26399 buffer used by the *fmemopen()* function.

26400 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
26401 any process. A SIGPIPE signal shall also be sent to the thread.

26402 The *fclose()* function may fail if:

26403 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
26404 capabilities of the device.

EXAMPLES

26405 None.
26406

APPLICATION USAGE

26407 None.
26408

RATIONALE

26409 None.
26410

FUTURE DIRECTIONS

26411 None.
26412

SEE ALSO

26413 *close()*, *fmemopen()*, *fopen()*, *getrlimit()*, *open_memstream()*, *ulimit*
26414

26415 XBD <stdio.h> |

CHANGE HISTORY

26416 First released in Issue 1. Derived from Issue 1 of the SVID.
26417

Issue 5

26418 Large File Summit extensions are added.
26419

Issue 6

26420 Extensions beyond the ISO C standard are marked.
26421

26422 The following new requirements on POSIX implementations derive from alignment with the
26423 Single UNIX Specification:

- 26424 • The [EFBIG] error is added as part of the large file support extensions.
- 26425 • The [ENXIO] optional error condition is added.

26426 The DESCRIPTION is updated to note that the stream and any buffer are disassociated whether
26427 or not the call succeeds. This is for alignment with the ISO/IEC 9899:1999 standard.

26428 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/28 is applied, updating the [EAGAIN]
26429 error in the ERRORS section from “the process would be delayed” to “the thread would be
26430 delayed”.

Issue 7

26431 Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file
26432 descriptors and streams.
26433

26434 The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open
26435 Group Technical Standard, 2006, Extended API Set Part 1.

26436 Changes are made related to support for finegrained timestamps. +

26437 **NAME**

26438 fcntl — file control

26439 **SYNOPSIS**

26440 #include <fcntl.h>

26441 int fcntl(int *fil-des*, int *cmd*, ...);26442 **DESCRIPTION**26443 The *fcntl()* function shall perform the operations described below on open files. The *fil-des*
26444 argument is a file descriptor.26445 The available values for *cmd* are defined in <fcntl.h> and are as follows:

26446 **F_DUPFD** Return a new file descriptor which shall be the lowest numbered available
26447 (that is, not already open) file descriptor greater than or equal to the third
26448 argument, *arg*, taken as an integer of type **int**. The new file descriptor shall
26449 refer to the same open file description as the original file descriptor, and shall
26450 share any locks. The FD_CLOEXEC flag associated with the new file
26451 descriptor shall be cleared to keep the file open across calls to one of the *exec*
26452 functions.

26453 **F_GETFD** Get the file descriptor flags defined in <fcntl.h> that are associated with the
26454 file descriptor *fil-des*. File descriptor flags are associated with a single file
26455 descriptor and do not affect other file descriptors that refer to the same file.

26456 **F_SETFD** Set the file descriptor flags defined in <fcntl.h>, that are associated with *fil-des*,
26457 to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the
26458 third argument is 0, the file shall remain open across the *exec* functions;
26459 otherwise, the file shall be closed upon successful execution of one of the *exec*
26460 functions.

26461 **F_GETFL** Get the file status flags and file access modes, defined in <fcntl.h>, for the file
26462 description associated with *fil-des*. The file access modes can be extracted from
26463 the return value using the mask O_ACCMODE, which is defined in <fcntl.h>.
26464 File status flags and file access modes are associated with the file description
26465 and do not affect other file descriptors that refer to the same file with different
26466 open file descriptions.

26467 **F_SETFL** Set the file status flags, defined in <fcntl.h>, for the file description associated
26468 with *fil-des* from the corresponding bits in the third argument, *arg*, taken as
26469 type **int**. Bits corresponding to the file access mode and the file creation flags,
26470 as defined in <fcntl.h>, that are set in *arg* shall be ignored. If any bits in *arg*
26471 other than those mentioned here are changed by the application, the result is
26472 unspecified.

26473 **F_GETOWN** If *fil-des* refers to a socket, get the process or process group ID specified to
26474 receive SIGURG signals when out-of-band data is available. Positive values
26475 indicate a process ID; negative values, other than -1, indicate a process group
26476 ID. If *fil-des* does not refer to a socket, the results are unspecified.

26477 **F_SETOWN** If *fil-des* refers to a socket, set the process or process group ID specified to
26478 receive SIGURG signals when out-of-band data is available, using the value of
26479 the third argument, *arg*, taken as type **int**. Positive values indicate a process
26480 ID; negative values, other than -1, indicate a process group ID. If *fil-des* does
26481 not refer to a socket, the results are unspecified.

26482 The following values for *cmd* are available for advisory record locking. Record locking shall be

26483		supported for regular files, and may be supported for other files.
26484	F_GETLK	Get the first lock which blocks the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type struct flock , defined in <fcntl.h> . The information retrieved shall overwrite the information passed to <i>fcntl()</i> in the structure flock . If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.
26485		
26486		
26487		
26488		
26489		
26490	F_SETLK	Set or clear a file segment lock according to the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type struct flock , defined in <fcntl.h> . F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in <fcntl.h> . If a shared or exclusive lock cannot be set, <i>fcntl()</i> shall return immediately with a return value of -1 .
26491		
26492		
26493		
26494		
26495		
26496		
26497	F_SETLKW	This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while <i>fcntl()</i> is waiting for a region, <i>fcntl()</i> shall be interrupted. Upon return from the signal handler, <i>fcntl()</i> shall return -1 with <i>errno</i> set to [EINTR], and the lock operation shall not be done.
26498		
26499		
26500		
26501		
26502		
26503		Additional implementation-defined values for <i>cmd</i> may be defined in <fcntl.h> . Their names shall start with F_.
26504		
26505		When a shared lock is set on a segment of a file, other processes shall be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the file descriptor was not opened with read access.
26506		
26507		
26508		
26509		An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock shall fail if the file descriptor was not opened with write access.
26510		
26511		
26512		The structure flock describes the type (<i>l_type</i>), starting offset (<i>l_whence</i>), relative offset (<i>l_start</i>), size (<i>l_len</i>), and process ID (<i>l_pid</i>) of the segment of the file to be affected.
26513		
26514		The value of <i>l_whence</i> is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative offset <i>l_start</i> bytes shall be measured from the start of the file, current position, or end of the file, respectively. The value of <i>l_len</i> is the number of consecutive bytes to be locked. The value of <i>l_len</i> may be negative (where the definition of off_t permits negative values of <i>l_len</i>). The <i>l_pid</i> field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, when a blocking lock is found, the values returned in the flock structure shall be as follows:
26515		
26516		
26517		
26518		
26519		
26520		
26521	<i>l_type</i>	Type of blocking lock found.
26522	<i>l_whence</i>	SEEK_SET.
26523	<i>l_start</i>	Start of the blocking lock.
26524	<i>l_len</i>	Length of the blocking lock.
26525	<i>l_pid</i>	Process ID of the process that holds the blocking lock.
26526		If the command is F_SETLKW and the process must wait for another process to release a lock, then the range of bytes to be locked shall be determined before the <i>fcntl()</i> function blocks. If the file size or file descriptor seek offset change while <i>fcntl()</i> is blocked, this shall not affect the range of bytes locked.
26527		
26528		
26529		

26530 If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start+l_len-1*. If *l_len* is
 26531 negative, the area affected shall start at *l_start+l_len* and end at *l_start-1*. Locks may start and
 26532 extend beyond the current end of a file, but shall not extend before the beginning of the file. A
 26533 lock shall be set to extend to the largest possible value of the file offset for that file by setting
 26534 *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to *SEEK_SET*, the whole file
 26535 shall be locked.

26536 There shall be at most one type of lock set for each byte in the file. Before a successful return
 26537 from an *F_SETLK* or an *F_SETLKW* request when the calling process has previously existing
 26538 locks on bytes in the region specified by the request, the previous lock type for each byte in the
 26539 specified region shall be replaced by the new lock type. As specified above under the
 26540 descriptions of shared locks and exclusive locks, an *F_SETLK* or an *F_SETLKW* request
 26541 (respectively) shall fail or block when another process has existing locks on bytes in the specified
 26542 region and the type of any of those locks conflicts with the type specified in the request.

26543 All locks associated with a file for a given process shall be removed when a file descriptor for
 26544 that file is closed by that process or the process holding that file descriptor terminates. Locks are
 26545 not inherited by a child process.

26546 A potential for deadlock occurs if a process controlling a locked region is put to sleep by
 26547 attempting to lock the locked region of another process. If the system detects that sleeping until
 26548 a locked region is unlocked would cause a deadlock, *fcntl()* shall fail with an *[EDEADLK]* error.

26549 An unlock (*F_UNLCK*) request in which *l_len* is non-zero and the offset of the last byte of the
 26550 requested segment is the maximum value for an object of type *off_t*, when the process has an
 26551 existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall
 26552 be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0.
 26553 Otherwise, an unlock (*F_UNLCK*) request shall attempt to unlock only the requested segment.

26554 SHM When the file descriptor *fildev* refers to a shared memory object, the behavior of *fcntl()* shall be
 26555 the same as for a regular file except the effect of the following values for the argument *cmd* shall
 26556 be unspecified: *F_SETFL*, *F_GETLK*, *F_SETLK*, and *F_SETLKW*.

26557 TYM If *fildev* refers to a typed memory object, the result of the *fcntl()* function is unspecified.

26558 RETURN VALUE

26559 Upon successful completion, the value returned shall depend on *cmd* as follows:

26560	<i>F_DUPFD</i>	A new file descriptor.
26561	<i>F_GETFD</i>	Value of flags defined in <i><fcntl.h></i> . The return value shall not be negative.
26562	<i>F_SETFD</i>	Value other than <i>-1</i> .
26563	<i>F_GETFL</i>	Value of file status flags and access modes. The return value is not negative.
26564	<i>F_SETFL</i>	Value other than <i>-1</i> .
26565	<i>F_GETLK</i>	Value other than <i>-1</i> .
26566	<i>F_SETLK</i>	Value other than <i>-1</i> .
26567	<i>F_SETLKW</i>	Value other than <i>-1</i> .
26568	<i>F_GETOWN</i>	Value of the socket owner process or process group; this will not be <i>-1</i> .
26569	<i>F_SETOWN</i>	Value other than <i>-1</i> .
26570		Otherwise, <i>-1</i> shall be returned and <i>errno</i> set to indicate the error.

26571 **ERRORS**26572 The *fcntl()* function shall fail if:

26573 [EACCES] or [EAGAIN]

26574 The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK)
26575 or exclusive (F_WRLCK) lock and the segment of a file to be locked is already
26576 exclusive-locked by another process, or the type is an exclusive lock and some
26577 portion of the segment of a file to be locked is already shared-locked or
26578 exclusive-locked by another process.

26579 [EBADF]

26580 The *fildev* argument is not a valid open file descriptor, or the argument *cmd* is
26581 F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK),
26582 and *fildev* is not a valid file descriptor open for reading, or the type of lock,
26583 *l_type*, is an exclusive lock (F_WRLCK), and *fildev* is not a valid file descriptor
open for writing.

26584 [EINTR]

The *cmd* argument is F_SETLKW and the function was interrupted by a signal.

26585 [EINVAL]

26586 The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD and *arg* is
26587 negative or greater than or equal to {OPEN_MAX}, or the *cmd* argument is
26588 F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by *arg* is not
valid, or *fildev* refers to a file that does not support locking.

26589 [EMFILE]

26590 The argument *cmd* is F_DUPFD and all file descriptors available to the process
26591 are currently open, or no file descriptors greater than or equal to *arg* are
available.

26592 [ENOLCK]

26593 The argument *cmd* is F_SETLK or F_SETLKW and satisfying the lock or unlock
26594 request would result in the number of locked regions in the system exceeding
a system-imposed limit.

26595 [EOVERFLOW]

One of the values to be returned cannot be represented correctly.

26596 [EOVERFLOW]

26597 The *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or,
26598 if *l_len* is non-zero, the largest offset of any byte in the requested segment
cannot be represented correctly in an object of type *off_t*.26599 The *fcntl()* function may fail if:

26600 [EDEADLK]

26601 The *cmd* argument is F_SETLKW, the lock is blocked by a lock from another
26602 process, and putting the calling process to sleep to wait for that lock to become
free would cause a deadlock.26603 **EXAMPLES**26604 **Locking and Unlocking a File**26605 The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then
26606 later remove it. F_SETLK is used to perform a non-blocking lock request so that the process does
26607 not have to wait if an incompatible lock is held by another process; instead the process can take
26608 some other action.26609 #include <stdlib.h>
26610 #include <unistd.h>
26611 #include <fcntl.h>
26612 #include <errno.h>
26613 #include <stdio.h>26614 int
26615 main(int argc, char *argv[])
26616 {

```

26617         int fd;
26618         struct flock fl;

26619         fd = open("testfile", O_RDWR);
26620         if (fd == -1)
26621             /* Handle error */;

26622         /* Make a non-blocking request to place a write lock
26623            on bytes 100-109 of testfile */

26624         fl.l_type = F_WRLCK;
26625         fl.l_whence = SEEK_SET;
26626         fl.l_start = 100;
26627         fl.l_len = 10;

26628         if (fcntl(fd, F_SETLK, &fl) == -1) {
26629             if (errno == EACCES || errno == EAGAIN) {
26630                 printf("Already locked by another process\n");
26631                 /* We can't get the lock at the moment */
26632             } else {
26633                 /* Handle unexpected error */;
26634             }
26635         } else { /* Lock was granted... */
26636             /* Perform I/O on bytes 100 to 109 of file */
26637             /* Unlock the locked bytes */

26638             fl.l_type = F_UNLCK;
26639             fl.l_whence = SEEK_SET;
26640             fl.l_start = 100;
26641             fl.l_len = 10;
26642             if (fcntl(fd, F_SETLK, &fl) == -1)
26643                 /* Handle error */;
26644         }
26645         exit(EXIT_SUCCESS);
26646     } /* main */

```

Setting the Close-on-Exec Flag

The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```

26649 #include <unistd.h>
26650 #include <fcntl.h>
26651 ...
26652     int flags;

26653     flags = fcntl(fd, F_GETFD);
26654     if (flags == -1)
26655         /* Handle error */;
26656     flags |= FD_CLOEXEC;
26657     if (fcntl(fd, F_SETFD, flags) == -1)
26658         /* Handle error */;

```

APPLICATION USAGE

None.

RATIONALE

The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

The *arg* values to `F_GETFD`, `F_SETFD`, `F_GETFL`, and `F_SETFL` all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation on them, rather than assuming that only the values defined by this volume of POSIX.1-200x are valid. It is a common error to forget this, particularly in the case of `F_SETFD`.

This volume of POSIX.1-200x permits concurrent read and write access to file data using the `fcntl()` function; this is a change from the 1984 `/usr/group` standard and early proposals. Without concurrency controls, this feature may not be fully utilized without occasional loss of data.

Data losses occur in several ways. One case occurs when several processes try to update the same record, without sequencing controls; several updates may occur in parallel and the last writer “wins”. Another case is a bit-tree or other internal list-based database that is undergoing reorganization. Without exclusive use to the tree segment by the updating process, other reading processes chance getting lost in the database when the index blocks are split, condensed, inserted, or deleted. While `fcntl()` is useful for many applications, it is not intended to be overly general and does not handle the bit-tree example well.

This facility is only required for regular files because it is not appropriate for many devices such as terminals and network connections.

Since `fcntl()` works with “any file descriptor associated with that file, however it is obtained”, the file descriptor may have been inherited through a `fork()` or `exec` operation and thus may affect a file that another process also has open.

The use of the open file description to identify what to lock requires extra calls and presents problems if several processes are sharing an open file description, but there are too many implementations of the existing mechanism for this volume of POSIX.1-200x to use different specifications.

Another consequence of this model is that closing any file descriptor for a given file (whether or not it is the same open file description that created the lock) causes the locks on that file to be relinquished for that process. Equivalently, any close for any file/process pair relinquishes the locks owned on that file for that process. But note that while an open file description may be shared through `fork()`, locks are not inherited through `fork()`. Yet locks may be inherited through one of the `exec` functions.

The identification of a machine in a network environment is outside the scope of this volume of POSIX.1-200x. Thus, an `l_sysid` member, such as found in System V, is not included in the locking structure.

Changing of lock types can result in a previously locked region being split into smaller regions.

Mandatory locking was a major feature of the 1984 `/usr/group` standard.

For advisory file record locking to be effective, all processes that have access to a file must cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode record locking is important when it cannot be assumed that all processes are cooperating. For example, if one user uses an editor to update a file at the same time that a second user executes another process that updates the same file and if only one of the two processes is using advisory locking, the processes are not cooperating. Enforcement-mode record locking would protect against accidental collisions.

26707 Secondly, advisory record locking requires a process using locking to bracket each I/O operation
 26708 with lock (or test) and unlock operations. With enforcement-mode file and record locking, a
 26709 process can lock the file once and unlock when all I/O operations have been completed.
 26710 Enforcement-mode record locking provides a base that can be enhanced; for example, with
 26711 sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so
 26712 other processes could read it, but none of them could write it.

26713 Mandatory locks were omitted for several reasons:

- 26714 1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most
 26715 implementations; this was confusing, at best.
- 26716 2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
- 26717 3. Any publicly readable file could be locked by anyone. Many historical implementations
 26718 keep the password database in a publicly readable file. A malicious user could thus
 26719 prohibit logins. Another possibility would be to hold open a long-distance telephone line.
- 26720 4. Some demand-paged historical implementations offer memory mapped files, and
 26721 enforcement cannot be done on that type of file.

26722 Since sleeping on a region is interrupted with any signal, *alarm()* may be used to provide a
 26723 timeout facility in applications requiring it. This is useful in deadlock detection. Since
 26724 implementation of full deadlock detection is not always feasible, the [EDEADLK] error was
 26725 made optional.

26726 FUTURE DIRECTIONS

26727 None.

26728 SEE ALSO

26729 *alarm()*, *close()*, *exec*, *open()*, *sigaction()*

26730 XBD [<fcntl.h>](#), [<signal.h>](#)

26731 CHANGE HISTORY

26732 First released in Issue 1. Derived from Issue 1 of the SVID.

26733 Issue 5

26734 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 26735 Threads Extension.

26736 Large File Summit extensions are added.

26737 Issue 6

26738 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

26739 The following new requirements on POSIX implementations derive from alignment with the
 26740 Single UNIX Specification:

- 26741 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 26742 required for conforming implementations of previous POSIX specifications, it was not
 26743 required for UNIX applications.
- 26744 • In the DESCRIPTION, sentences describing behavior when *l_len* is negative are now
 26745 mandated, and the description of unlock (F_UNLOCK) when *l_len* is non-negative is
 26746 mandated.
- 26747 • In the ERRORS section, the [EINVAL] error condition has the case mandated when the *cmd*
 26748 is invalid, and two [EOVERFLOW] error conditions are added.

26749 The F_GETOWN and F_SETOWN values are added for sockets.

26750 The following changes were made to align with the IEEE P1003.1a draft standard:

26751

- Clarification is added that the extent of the bytes locked is determined prior to the blocking action.

26752

26753

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *fcntl()* results are unspecified for typed memory objects.

26754

26755

The normative text is updated to avoid use of the term “must” for application requirements.

26756

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/29 is applied, adding the example to the EXAMPLES section.

26757

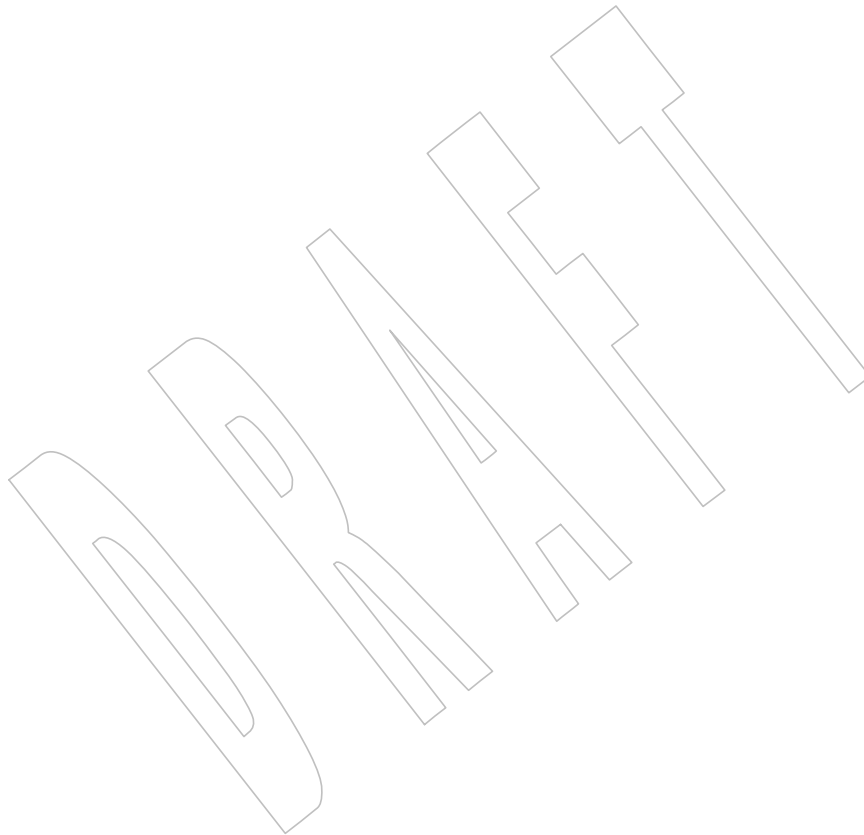
26758

Issue 7

26759

The optional `<unistd.h>` header is removed from this function, since `<fcntl.h>` now defines SEEK_SET, SEEK_CUR, and SEEK_END as part of the Base.

26760



26761 **NAME**
 26762 `fdatasync` — synchronize the data of a file (**REALTIME**)

26763 **SYNOPSIS**

26764 SIO `#include <unistd.h>`
 26765 `int fdatasync(int fildes);`

26766 **DESCRIPTION**

26767 The `fdatasync()` function shall force all currently queued I/O operations associated with the file
 26768 indicated by file descriptor `fil`des to the synchronized I/O completion state.

26769 The functionality shall be equivalent to `fsync()` with the symbol `_POSIX_SYNCHRONIZED_IO`
 26770 defined, with the exception that all I/O operations shall be completed as defined for
 26771 synchronized I/O data integrity completion.

26772 **RETURN VALUE**

26773 If successful, the `fdatasync()` function shall return the value 0; otherwise, the function shall return
 26774 the value `-1` and set `errno` to indicate the error. If the `fdatasync()` function fails, outstanding I/O
 26775 operations are not guaranteed to have been completed.

26776 **ERRORS**

26777 The `fdatasync()` function shall fail if:

26778 [EBADF] The `fil`des argument is not a valid file descriptor open for writing.

26779 [EINVAL] This implementation does not support synchronized I/O for this file.

26780 In the event that any of the queued I/O operations fail, `fdatasync()` shall return the error
 26781 conditions defined for `read()` and `write()`.

26782 **EXAMPLES**

26783 None.

26784 **APPLICATION USAGE**

26785 None.

26786 **RATIONALE**

26787 None.

26788 **FUTURE DIRECTIONS**

26789 None.

26790 **SEE ALSO**

26791 [aio_fsync\(\)](#), [fcntl\(\)](#), [fsync\(\)](#), [open\(\)](#), [read\(\)](#), [write\(\)](#)

26792 XBD [<unistd.h>](#)

26793 **CHANGE HISTORY**

26794 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

26795 **Issue 6**

26796 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 26797 implementation does not support the Synchronized Input and Output option.

26798 The `fdatasync()` function is marked as part of the Synchronized Input and Output option.

26799 **NAME**26800 `fdetach` — detach a name from a STREAMS-based file descriptor (**STREAMS**)26801 **SYNOPSIS**

```
26802 OB XSR #include <stropts.h>
26803 int fdetach(const char *path);
```

26804 **DESCRIPTION**

26805 The `fdetach()` function shall detach a STREAMS-based file from the file to which it was attached
 26806 by a previous call to `fattach()`. The `path` argument points to the pathname of the attached
 26807 STREAMS file. The process shall have appropriate privileges or be the owner of the file. A
 26808 successful call to `fdetach()` shall cause all pathnames that named the attached STREAMS file to
 26809 again name the file to which the STREAMS file was attached. All subsequent operations on `path`
 26810 shall operate on the underlying file and not on the STREAMS file.

26811 All open file descriptions established while the STREAMS file was attached to the file referenced
 26812 by `path` shall still refer to the STREAMS file after the `fdetach()` has taken effect.

26813 If there are no open file descriptors or other references to the STREAMS file, then a successful
 26814 call to `fdetach()` shall be equivalent to performing the last `close()` on the attached file.

26815 **RETURN VALUE**

26816 Upon successful completion, `fdetach()` shall return 0; otherwise, it shall return `-1` and set `errno` to
 26817 indicate the error.

26818 **ERRORS**

26819 The `fdetach()` function shall fail if:

- 26820 [EACCES] Search permission is denied on a component of the path prefix.
- 26821 [EINVAL] The `path` argument names a file that is not currently attached.
- 26822 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
 26823 argument.
- 26824 [ENAMETOOLONG]
 26825 The size of a pathname exceeds {PATH_MAX} or a pathname component is
 26826 longer than {NAME_MAX}.
- 26827 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.
- 26828 [ENOTDIR] A component of the path prefix is not a directory.
- 26829 [EPERM] The effective user ID is not the owner of `path` and the process does not have
 26830 appropriate privileges.

26831 The `fdetach()` function may fail if:

- 26832 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 26833 resolution of the `path` argument.
- 26834 [ENAMETOOLONG]
 26835 Pathname resolution of a symbolic link produced an intermediate result
 26836 whose length exceeds {PATH_MAX}.

26837 **EXAMPLES**26838 **Detaching a File**

26839 The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to
 26840 which it was attached by a previous, successful call to *fattach()*. Subsequent calls to open this
 26841 file refer to the underlying file, not to the STREAMS file.

```
26842 #include <stropts.h>
26843 ...
26844     char *filename = "/tmp/named-STREAM";
26845     int ret;
26846
26847     ret = fdetach(filename);
```

26847 **APPLICATION USAGE**

26848 None.

26849 **RATIONALE**

26850 None.

26851 **FUTURE DIRECTIONS**

26852 The *fdetach()* function may be removed in a future version.

26853 **SEE ALSO**

26854 *fattach()*

26855 XBD [<stropts.h>](#)

26856 **CHANGE HISTORY**

26857 First released in Issue 4, Version 2.

26858 **Issue 5**

26859 Moved from X/OPEN UNIX extension to BASE.

26860 **Issue 6**

26861 The normative text is updated to avoid use of the term “must” for application requirements.

26862 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
 26863 [ELOOP] error condition is added.

26864 **Issue 7**

26865 The *fdetach()* function is marked obsolescent.

26866 **NAME**26867 `fdim`, `fdimf`, `fdiml` — compute positive difference between two floating-point numbers26868 **SYNOPSIS**26869 `#include <math.h>`26870 `double fdim(double x, double y);`26871 `float fdimf(float x, float y);`26872 `long double fdiml(long double x, long double y);`26873 **DESCRIPTION**26874 CX The functionality described on this reference page is aligned with the ISO C standard. Any
26875 conflict between the requirements described here and the ISO C standard is unintentional. This
26876 volume of POSIX.1-200x defers to the ISO C standard.26877 These functions shall determine the positive difference between their arguments. If x is greater
26878 than y , $x-y$ is returned. If x is less than or equal to y , $+0$ is returned.26879 An application wishing to check for error situations should set *errno* to zero and call
26880 *feclearexcept*(`FE_ALL_EXCEPT`) before calling these functions. On return, if *errno* is non-zero or
26881 *fetestexcept*(`FE_INVALID` | `FE_DIVBYZERO` | `FE_OVERFLOW` | `FE_UNDERFLOW`) is non-
26882 zero, an error has occurred.26883 **RETURN VALUE**

26884 Upon successful completion, these functions shall return the positive difference value.

26885 If $x-y$ is positive and overflows, a range error shall occur and *fdim*(), *fdimf*(), and *fdiml*() shall
26886 return the value of the macro `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL`, respectively.26887 XSI If $x-y$ is positive and underflows, a range error may occur, and either $(x-y)$ (if representable), or
26888 0.0 (if supported), or an implementation-defined value shall be returned.26889 MX If x or y is NaN, a NaN shall be returned.26890 **ERRORS**26891 The *fdim*() function shall fail if:

26892 Range Error The result overflows.

26893 If the integer expression (*math_errhandling* & `MATH_ERRNO`) is non-zero,
26894 then *errno* shall be set to `[ERANGE]`. If the integer expression
26895 (*math_errhandling* & `MATH_ERREXCEPT`) is non-zero, then the overflow
26896 floating-point exception shall be raised.26897 The *fdim*() function may fail if:

26898 Range Error The result underflows.

26899 If the integer expression (*math_errhandling* & `MATH_ERRNO`) is non-zero,
26900 then *errno* shall be set to `[ERANGE]`. If the integer expression
26901 (*math_errhandling* & `MATH_ERREXCEPT`) is non-zero, then the underflow
26902 floating-point exception shall be raised.

26903
26904
26905
26906
26907
26908
26909
26910
26911
26912
26913
26914
26915
26916
26917
26918
26919

EXAMPLES

None.

APPLICATION USAGE

On implementations supporting IEEE Std 754-1985, $x-y$ cannot underflow, and hence the 0.0 return value is shaded as an extension for implementations supporting the XSI option rather than an MX extension.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

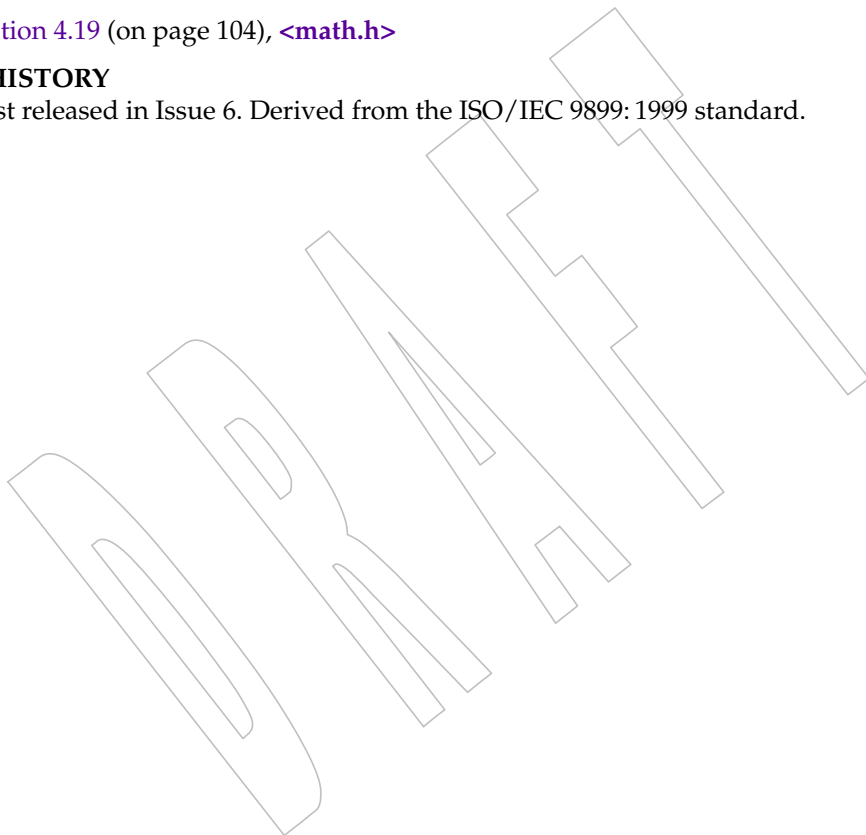
SEE ALSO

feclearexcept(), *fetestexcept()*, *fmax()*, *fmin()*

Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.



26920 **NAME**26921 `fdopen` — associate a stream with a file descriptor26922 **SYNOPSIS**

```
26923 CX #include <stdio.h>
26924 FILE *fdopen(int fil-des, const char *mode);
```

26925 **DESCRIPTION**26926 The `fdopen()` function shall associate a stream with a file descriptor.26927 The *mode* argument is a character string having one of the following values:

26928	<i>r</i> or <i>rb</i>	Open a file for reading.
26929	<i>w</i> or <i>wb</i>	Open a file for writing.
26930	<i>a</i> or <i>ab</i>	Open a file for writing at end-of-file.
26931	<i>r+</i> or <i>rb+</i> or <i>r+b</i>	Open a file for update (reading and writing).
26932	<i>w+</i> or <i>wb+</i> or <i>w+b</i>	Open a file for update (reading and writing).
26933	<i>a+</i> or <i>ab+</i> or <i>a+b</i>	Open a file for update (reading and writing) at end-of-file.

26934 The meaning of these flags is exactly as specified in `open()`, except that modes beginning with *w*
26935 shall not cause truncation of the file.26936 Additional values for the *mode* argument may be supported by an implementation.26937 The application shall ensure that the mode of the stream as expressed by the *mode* argument is
26938 allowed by the file access mode of the open file description to which *fil-des* refers. The file
26939 position indicator associated with the new stream is set to the position indicated by the file offset
26940 associated with the file descriptor.26941 The error and end-of-file indicators for the stream shall be cleared. The `fdopen()` function may
26942 cause the last data access timestamp of the underlying file to be marked for update.26943 SHM If *fil-des* refers to a shared memory object, the result of the `fdopen()` function is unspecified.26944 TYM If *fil-des* refers to a typed memory object, the result of the `fdopen()` function is unspecified.26945 The `fdopen()` function shall preserve the offset maximum previously set for the open file
26946 description corresponding to *fil-des*.26947 **RETURN VALUE**26948 Upon successful completion, `fdopen()` shall return a pointer to a stream; otherwise, a null pointer
26949 shall be returned and *errno* set to indicate the error.26950 **ERRORS**

26951	The <code>fdopen()</code> function shall fail if:		+
26952	[EMFILE]	{STREAM_MAX} streams are currently open in the calling process.	+
26953	The <code>fdopen()</code> function may fail if:		
26954	[EBADF]	The <i>fil-des</i> argument is not a valid file descriptor.	
26955	[EINVAL]	The <i>mode</i> argument is not a valid mode.	
26956	[EMFILE]	{FOPEN_MAX} streams are currently open in the calling process.	-

26957 [ENOMEM] Insufficient space to allocate a buffer.

26958 EXAMPLES

26959 None.

26960 APPLICATION USAGE

26961 File descriptors are obtained from calls like *open()*, *dup()*, *creat()*, or *pipe()*, which open files but
26962 do not return streams.

26963 RATIONALE

26964 The file descriptor may have been obtained from *open()*, *creat()*, *pipe()*, *dup()*, *fcntl()*, or *socket()*;
26965 inherited through *fork()*, *posix_spawn()*, or *exec*; or perhaps obtained by other means.

26966 The meanings of the *mode* arguments of *fdopen()* and *fopen()* differ. With *fdopen()*, open for write
26967 (*w* or *w+*) does not truncate, and append (*a* or *a+*) cannot create for writing. The *mode* argument
26968 formats that include *a b* are allowed for consistency with the ISO C standard function *fopen()*.
26969 The *b* has no effect on the resulting stream. Although not explicitly required by this volume of
26970 POSIX.1-200x, a good implementation of append (*a*) mode would cause the *O_APPEND* flag to
26971 be set.

26972 FUTURE DIRECTIONS

26973 None.

26974 SEE ALSO

26975 Section 2.5.1 (on page 470), *fclose()*, *fmemopen()*, *fopen()*, *open()*, *open_memstream()*,
26976 *posix_spawn()*, *socket()*

26977 XBD <stdio.h>

26978 CHANGE HISTORY

26979 First released in Issue 1. Derived from Issue 1 of the SVID.

26980 Issue 5

26981 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

26982 Large File Summit extensions are added.

26983 Issue 6

26984 The following new requirements on POSIX implementations derive from alignment with the
26985 Single UNIX Specification:

- 26986 • In the DESCRIPTION, the use and setting of the *mode* argument are changed to include
26987 binary streams.
- 26988 • In the DESCRIPTION, text is added for large file support to indicate setting of the offset
26989 maximum in the open file description.
- 26990 • All errors identified in the ERRORS section are added.
- 26991 • In the DESCRIPTION, text is added that the *fdopen()* function may cause *st_atime* to be
26992 updated.

26993 The following changes were made to align with the IEEE P1003.1a draft standard:

- 26994 • Clarification is added that it is the responsibility of the application to ensure that the mode
26995 is compatible with the open file descriptor.

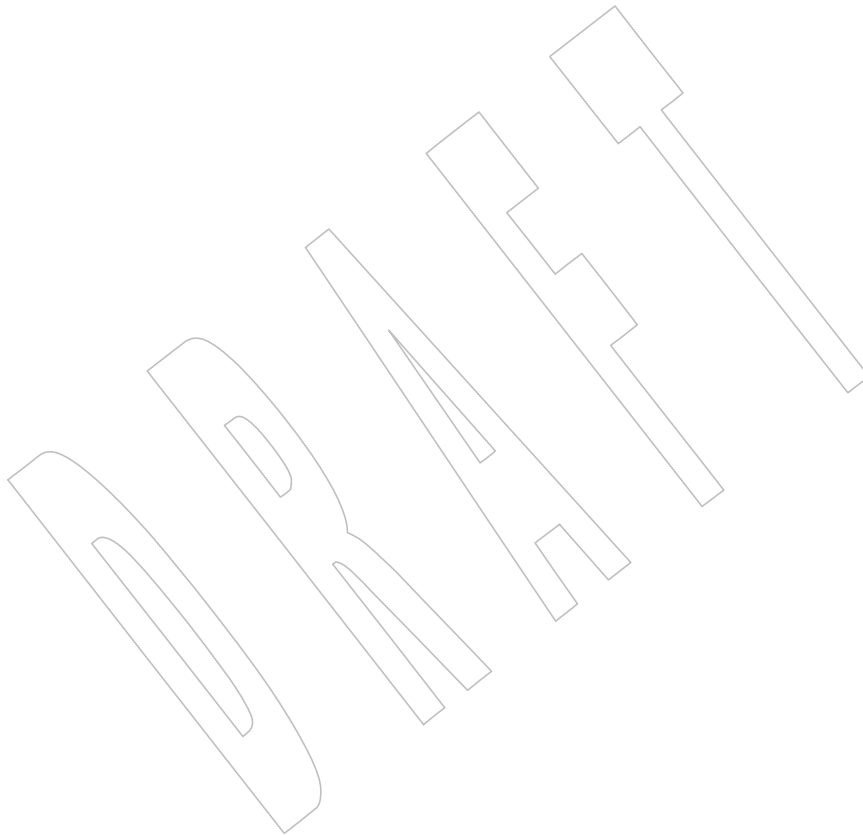
26996 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
26997 *fdopen()* results are unspecified for typed memory objects.

26998 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/30 is applied, making corrections to the
26999 RATIONALE.

fdopen()27000
27001
27002**Issue 7**

SD5-XSH-ERN-149 is applied, adding the {STREAM_MAX} [EMFILE] error condition. +

Changes are made related to support for finegrained timestamps.



27003 **NAME**

27004 fdopendir, opendir — open directory associated with file descriptor

27005 **SYNOPSIS**

```
27006 #include <dirent.h>
27007 DIR *fdopendir(int fd);
27008 DIR *opendir(const char *dirname);
```

27009 **DESCRIPTION**

27010 The *fdopendir()* function shall be equivalent to the *opendir()* function except that the directory is
 27011 specified by a file descriptor rather than by a name. The file offset associated with the file
 27012 descriptor at the time of the call determines which entries are returned.

27013 Upon successful return from *fdopendir()*, the file descriptor is under the control of the system,
 27014 and if any attempt is made to close the file descriptor, or to modify the state of the associated
 27015 description other than by means of *closedir()*, *readdir()*, *readdir_r()*, or *rewinddir()*, the behavior is
 27016 implementation-defined. Upon calling *closedir()* the file descriptor shall be closed.

27017 It is unspecified whether the FD_CLOEXEC flag will be set on the file descriptor by a successful
 27018 call to *fdopendir()*.

27019 The *opendir()* function shall open a directory stream corresponding to the directory named by
 27020 the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is
 27021 implemented using a file descriptor, applications shall only be able to open up to a total of
 27022 {OPEN_MAX} files and directories.

27023 If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the
 27024 O_DIRECTORY flag was passed to *open()*.

27025 **RETURN VALUE**

27026 Upon successful completion, these functions shall return a pointer to an object of type **DIR**.
 27027 Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

27028 **ERRORS**

27029 The *fdopendir()* function shall fail if:

27030 [EBADF] The *fd* argument is not a valid file descriptor open for reading.

27031 [ENOTDIR] The descriptor *fd* is not associated with a directory.

27032 The *opendir()* function shall fail if:

27033 [EACCES] Search permission is denied for the component of the path prefix of *dirname* or
 27034 read permission is denied for *dirname*.

27035 [ELOOP] A loop exists in symbolic links encountered during resolution of the *dirname*
 27036 argument.

27037 [ENAMETOOLONG]

27038 The length of the *dirname* argument exceeds {PATH_MAX} or a pathname
 27039 component is longer than {NAME_MAX}.

27040 [ENOENT] A component of *dirname* does not name an existing directory or *dirname* is an
 27041 empty string.

27042 [ENOTDIR] A component of *dirname* is not a directory.

- 27043 The *opendir()* function may fail if:
- 27044 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
27045 resolution of the *dirname* argument.
- 27046 [EMFILE] All file descriptors available to the process are currently open.
- 27047 [ENAMETOOLONG]
27048 As a result of encountering a symbolic link in resolution of the *dirname*
27049 argument, the length of the substituted pathname string exceeded
27050 {PATH_MAX}.
- 27051 [ENFILE] Too many files are currently open in the system.

EXAMPLES**Open a Directory Stream**

The following program fragment demonstrates how the *opendir()* function is used.

```
27055 #include <dirent.h>
27056 ...
27057     DIR *dir;
27058     struct dirent *dp;
27059 ...
27060     if ((dir = opendir(".")) == NULL) {
27061         perror("Cannot open .");
27062         exit(1);
27063     }
27064     while ((dp = readdir(dir)) != NULL) {
27065         ...
```

Find And Open a File

The following program fragment searches through a given directory looking for files larger than 1 MB.

```
27069 #include <stdio.h>
27070 #include <dirent.h>
27071 #include <fcntl.h>
27072 #include <sys/stat.h>
27073 #include <stdlib.h>
27074 #include <unistd.h>
27075
27076 int
27077 main(int argc, char *argv[])
27078 {
27079     struct stat statbuf;
27080     DIR *d;
27081     struct dirent *dp;
27082     int dfd, ffd;
27083
27084     if ((d = fdopendir((dfd = open("./tmp", O_RDONLY)))) == NULL) {
27085         fprintf(stderr, "Cannot open ./tmp directory\n");
27086         exit(1);
27087     }
27088     while ((dp = readdir(d)) != NULL) {
27089         if (dp->d_name[0] == '.')
27090             continue;
```

```

27089         /* there is a possible race condition here as the file           +
27090         * could be renamed between the readdir and the open */         +
27091         if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {         +
27092             perror(dp->d_name);                                         +
27093             continue;                                                  +
27094         }                                                               +
27095         if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {+
27096             /* found it ... */                                         +
27097             printf("%s: %dk\n", dp->d_name, statbuf.st_size / 1024);    +
27098         }                                                               +
27099         close(ffd);                                                    +
27100     }                                                                    +
27101     closedir(d);                                                       +
27102     close(dfd);                                                        +
27103     return 0;                                                          +
27104 }                                                                       +

```

APPLICATION USAGE

The *opendir()* function should be used in conjunction with *readdir()*, *closedir()*, and *rewinddir()* to examine the contents of the directory (see the EXAMPLES section in *readdir()*). This method is recommended for portability.

RATIONALE

The purpose of the *fdopendir()* function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *opendir()*, resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-200x does not mandate that the directory stream be implemented using file descriptors. The description of *closedir()* clarifies that if a file descriptor is used for the directory stream, it is mandatory that *closedir()* deallocate the file descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the `FD_CLOEXEC` had been set for the file descriptor.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-200x does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a "normal" entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-200x imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed or after a *fork()* or one of the *exec* function calls.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir(), *dirfd()*, *fstatat()*, *open()*, *readdir()*, *rewinddir()*, *symlink()*

XBD `<dirent.h>`, `<limits.h>`, `<sys/types.h>`

27137
27138

27139
27140

27141
27142

27143
27144
27145

27146
27147

27148
27149

27150
27151

27152
27153

27154

CHANGE HISTORY

First released in Issue 2.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The `fdopendir()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

An additional example is added.

+

27155 **NAME**27156 `feclearexcept` — clear floating-point exception27157 **SYNOPSIS**27158 `#include <fenv.h>`27159 `int feclearexcept(int excepts);`27160 **DESCRIPTION**27161 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27162 conflict between the requirements described here and the ISO C standard is unintentional. This
27163 volume of POSIX.1-200x defers to the ISO C standard.27164 The *feclearexcept()* function shall attempt to clear the supported floating-point exceptions
27165 represented by *excepts*.27166 **RETURN VALUE**27167 If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept()*
27168 shall return zero. Otherwise, it shall return a non-zero value.27169 **ERRORS**

27170 No errors are defined.

27171 **EXAMPLES**

27172 None.

27173 **APPLICATION USAGE**

27174 None.

27175 **RATIONALE**

27176 None.

27177 **FUTURE DIRECTIONS**

27178 None.

27179 **SEE ALSO**27180 *fegetexceptflag()*, *feraiseexcept()*, *fetestexcept()*27181 XBD [<fenv.h>](#)27182 **CHANGE HISTORY**

27183 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27184 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27185 **NAME**
 27186 `fegetenv, fesetenv` — get and set current floating-point environment

27187 **SYNOPSIS**
 27188 `#include <fenv.h>`
 27189 `int fegetenv(fenv_t *envp);`
 27190 `int fesetenv(const fenv_t *envp);`

27191 **DESCRIPTION**
 27192 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27193 conflict between the requirements described here and the ISO C standard is unintentional. This
 27194 volume of POSIX.1-200x defers to the ISO C standard.

27195 The `fegetenv()` function shall attempt to store the current floating-point environment in the object
 27196 pointed to by `envp`.

27197 The `fesetenv()` function shall attempt to establish the floating-point environment represented by
 27198 the object pointed to by `envp`. The argument `envp` shall point to an object set by a call to
 27199 `fegetenv()` or `feholdexcept()`, or equal a floating-point environment macro. The `fesetenv()` function
 27200 does not raise floating-point exceptions, but only installs the state of the floating-point status
 27201 flags represented through its argument.

27202 **RETURN VALUE**
 27203 If the representation was successfully stored, `fegetenv()` shall return zero. Otherwise, it shall
 27204 return a non-zero value. If the environment was successfully established, `fesetenv()` shall return
 27205 zero. Otherwise, it shall return a non-zero value.

27206 **ERRORS**
 27207 No errors are defined.

27208 **EXAMPLES**
 27209 None.

27210 **APPLICATION USAGE**
 27211 None.

27212 **RATIONALE**
 27213 None.

27214 **FUTURE DIRECTIONS**
 27215 None.

27216 **SEE ALSO**
 27217 [feholdexcept\(\)](#), [feupdateenv\(\)](#)

27218 XBD [<fenv.h>](#)

27219 **CHANGE HISTORY**
 27220 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.
 27221 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27222 **NAME**

27223 fegetexceptflag, fesetexceptflag — get and set floating-point status flags

27224 **SYNOPSIS**

27225 #include <fenv.h>

27226 int fegetexceptflag(fexcept_t *flagp, int excepts);

27227 int fesetexceptflag(const fexcept_t *flagp, int excepts);

27228 **DESCRIPTION**

27229 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27230 conflict between the requirements described here and the ISO C standard is unintentional. This
 27231 volume of POSIX.1-200x defers to the ISO C standard.

27232 The *fegetexceptflag()* function shall attempt to store an implementation-defined representation of
 27233 the states of the floating-point status flags indicated by the argument *excepts* in the object
 27234 pointed to by the argument *flagp*.

27235 The *fesetexceptflag()* function shall attempt to set the floating-point status flags indicated by the
 27236 argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by
 27237 *flagp* shall have been set by a previous call to *fegetexceptflag()* whose second argument
 27238 represented at least those floating-point exceptions represented by the argument *excepts*. This
 27239 function does not raise floating-point exceptions, but only sets the state of the flags.

27240 **RETURN VALUE**

27241 If the representation was successfully stored, *fegetexceptflag()* shall return zero. Otherwise, it
 27242 shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions
 27243 were successfully set, *fesetexceptflag()* shall return zero. Otherwise, it shall return a non-zero
 27244 value.

27245 **ERRORS**

27246 No errors are defined.

27247 **EXAMPLES**

27248 None.

27249 **APPLICATION USAGE**

27250 None.

27251 **RATIONALE**

27252 None.

27253 **FUTURE DIRECTIONS**

27254 None.

27255 **SEE ALSO**27256 *feclearexcept()*, *feraiseexcept()*, *fetestexcept()*

27257 XBD <fenv.h>

27258 **CHANGE HISTORY**

27259 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

27260 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27261 **NAME**
 27262 fegetround, fesetround — get and set current rounding direction

27263 **SYNOPSIS**
 27264 #include <fenv.h>
 27265 int fegetround(void);
 27266 int fesetround(int round);

27267 **DESCRIPTION**
 27268 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27269 conflict between the requirements described here and the ISO C standard is unintentional. This
 27270 volume of POSIX.1-200x defers to the ISO C standard.

27271 The *fegetround()* function shall get the current rounding direction.

27272 The *fesetround()* function shall establish the rounding direction represented by its argument
 27273 *round*. If the argument is not equal to the value of a rounding direction macro, the rounding
 27274 direction is not changed.

27275 **RETURN VALUE**
 27276 The *fegetround()* function shall return the value of the rounding direction macro representing the
 27277 current rounding direction or a negative value if there is no such rounding direction macro or
 27278 the current rounding direction is not determinable.

27279 The *fesetround()* function shall return a zero value if and only if the requested rounding direction
 27280 was established.

27281 **ERRORS**
 27282 No errors are defined.

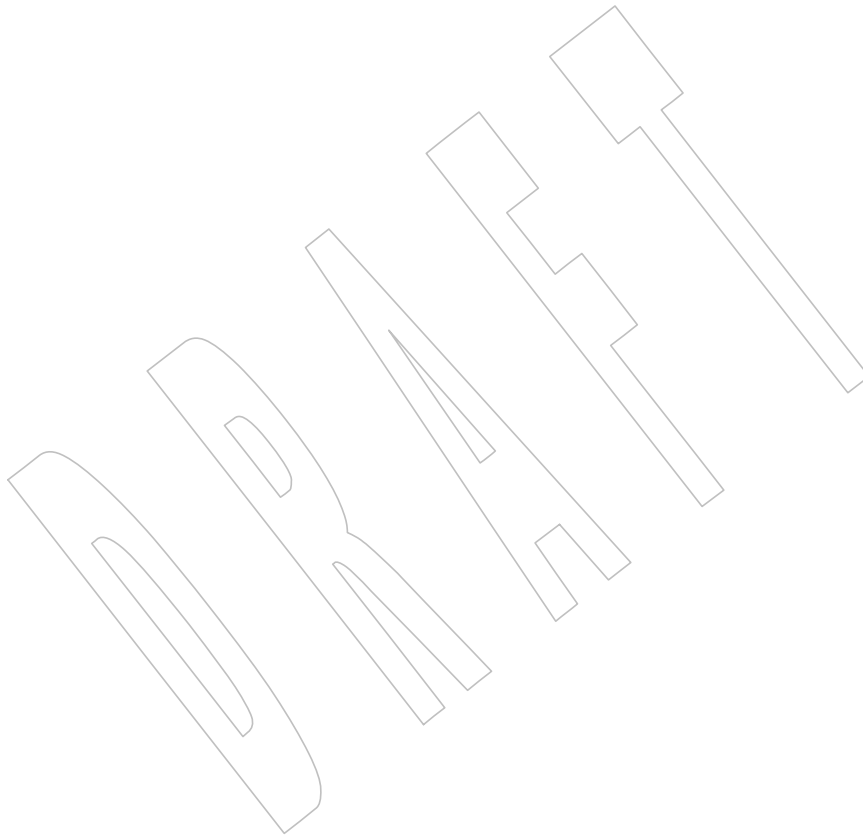
27283 **EXAMPLES**
 27284 The following example saves, sets, and restores the rounding direction, reporting an error and
 27285 aborting if setting the rounding direction fails:

```
27286 #include <fenv.h>
27287 #include <assert.h>
27288 void f(int round_dir)
27289 {
27290     #pragma STDC FENV_ACCESS ON
27291     int save_round;
27292     int setround_ok;
27293     save_round = fegetround();
27294     setround_ok = fesetround(round_dir);
27295     assert(setround_ok == 0);
27296     /* ... */
27297     fesetround(save_round);
27298     /* ... */
27299 }
```

27300 **APPLICATION USAGE**
 27301 None.

27302 **RATIONALE**
 27303 None.

- 27304 **FUTURE DIRECTIONS**
- 27305 None.
- 27306 **SEE ALSO**
- 27307 XBD <[fenv.h](#)>
- 27308 **CHANGE HISTORY**
- 27309 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.
- 27310 ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.



feholdexcept()27311 **NAME**

27312 feholdexcept — save current floating-point environment

27313 **SYNOPSIS**

27314 #include <fenv.h>

27315 int feholdexcept(fenv_t *envp);

27316 **DESCRIPTION**27317 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27318 conflict between the requirements described here and the ISO C standard is unintentional. This
27319 volume of POSIX.1-200x defers to the ISO C standard.27320 The *feholdexcept()* function shall save the current floating-point environment in the object
27321 pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on
27322 floating-point exceptions) mode, if available, for all floating-point exceptions.27323 **RETURN VALUE**27324 The *feholdexcept()* function shall return zero if and only if non-stop floating-point exception
27325 handling was successfully installed.27326 **ERRORS**

27327 No errors are defined.

27328 **EXAMPLES**

27329 None.

27330 **APPLICATION USAGE**

27331 None.

27332 **RATIONALE**27333 The *feholdexcept()* function should be effective on typical IEC 60559:1989 standard
27334 implementations which have the default non-stop mode and at least one other mode for trap
27335 handling or aborting. If the implementation provides only the non-stop mode, then installing the
27336 non-stop mode is trivial.27337 **FUTURE DIRECTIONS**

27338 None.

27339 **SEE ALSO**27340 *fegetenv()*, *feupdateenv()*

27341 XBD <fenv.h>

27342 **CHANGE HISTORY**

27343 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

27344 **NAME**
 27345 feof — test end-of-file indicator on a stream

27346 **SYNOPSIS**
 27347 #include <stdio.h>
 27348 int feof(FILE *stream);

27349 **DESCRIPTION**
 27350 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27351 conflict between the requirements described here and the ISO C standard is unintentional. This
 27352 volume of POSIX.1-200x defers to the ISO C standard.

27353 The *feof()* function shall test the end-of-file indicator for the stream pointed to by *stream*.

27354 **RETURN VALUE**
 27355 The *feof()* function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

27356 **ERRORS**
 27357 No errors are defined.

27358 **EXAMPLES**
 27359 None.

27360 **APPLICATION USAGE**
 27361 None.

27362 **RATIONALE**
 27363 None.

27364 **FUTURE DIRECTIONS**
 27365 None.

27366 **SEE ALSO**
 27367 *clearerr()*, *ferror()*, *fopen()*

27368 XBD <stdio.h>

27369 **CHANGE HISTORY**
 27370 First released in Issue 1. Derived from Issue 1 of the SVID.

27371 **NAME**
 27372 `feraiseexcept` — raise floating-point exception

27373 **SYNOPSIS**
 27374 `#include <fenv.h>`
 27375 `int feraiseexcept(int excepts);`

27376 **DESCRIPTION**
 27377 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27378 conflict between the requirements described here and the ISO C standard is unintentional. This
 27379 volume of POSIX.1-200x defers to the ISO C standard.

27380 The `feraiseexcept()` function shall attempt to raise the supported floating-point exceptions
 27381 represented by the argument `excepts`. The order in which these floating-point exceptions are
 27382 raised is unspecified. Whether the `feraiseexcept()` function additionally raises the inexact floating-
 27383 point exception whenever it raises the overflow or underflow floating-point exception is
 27384 implementation-defined.

27385 **RETURN VALUE**
 27386 If the argument is zero or if all the specified exceptions were successfully raised, `feraiseexcept()`
 27387 shall return zero. Otherwise, it shall return a non-zero value.

27388 **ERRORS**
 27389 No errors are defined.

27390 **EXAMPLES**
 27391 None.

27392 **APPLICATION USAGE**
 27393 The effect is intended to be similar to that of floating-point exceptions raised by arithmetic
 27394 operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

27395 **RATIONALE**
 27396 Raising overflow or underflow is allowed to also raise inexact because on some architectures the
 27397 only practical way to raise an exception is to execute an instruction that has the exception as a
 27398 side effect. The function is not restricted to accept only valid coincident expressions for atomic
 27399 operations, so the function can be used to raise exceptions accrued over several operations.

27400 **FUTURE DIRECTIONS**
 27401 None.

27402 **SEE ALSO**
 27403 [`feclearexcept\(\)`](#), [`fetestexceptflag\(\)`](#), [`fetestexcept\(\)`](#)

27404 XBD [`<fenv.h>`](#)

27405 **CHANGE HISTORY**
 27406 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.
 27407 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

27408 **NAME**
 27409 `ferror` — test error indicator on a stream

27410 **SYNOPSIS**
 27411 `#include <stdio.h>`
 27412 `int ferror(FILE *stream);`

27413 **DESCRIPTION**
 27414 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27415 conflict between the requirements described here and the ISO C standard is unintentional. This
 27416 volume of POSIX.1-200x defers to the ISO C standard.

27417 The `ferror()` function shall test the error indicator for the stream pointed to by `stream`.

27418 **RETURN VALUE**
 27419 The `ferror()` function shall return non-zero if and only if the error indicator is set for `stream`.

27420 **ERRORS**
 27421 No errors are defined.

27422 **EXAMPLES**
 27423 None.

27424 **APPLICATION USAGE**
 27425 None.

27426 **RATIONALE**
 27427 None.

27428 **FUTURE DIRECTIONS**
 27429 None.

27430 **SEE ALSO**
 27431 `clearerr()`, `feof()`, `fopen()`

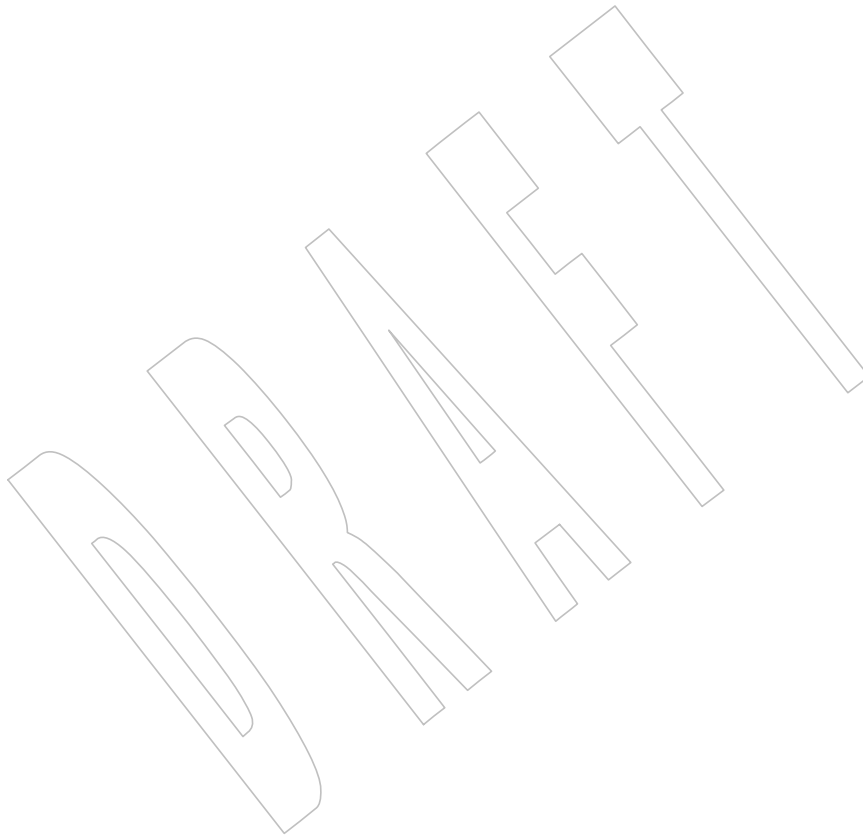
27432 XBD `<stdio.h>`

27433 **CHANGE HISTORY**
 27434 First released in Issue 1. Derived from Issue 1 of the SVID.

27435 **NAME**
27436 fesetenv — set current floating-point environment

27437 **SYNOPSIS**
27438 #include <fenv.h>
27439 int fesetenv(const fenv_t *envp);

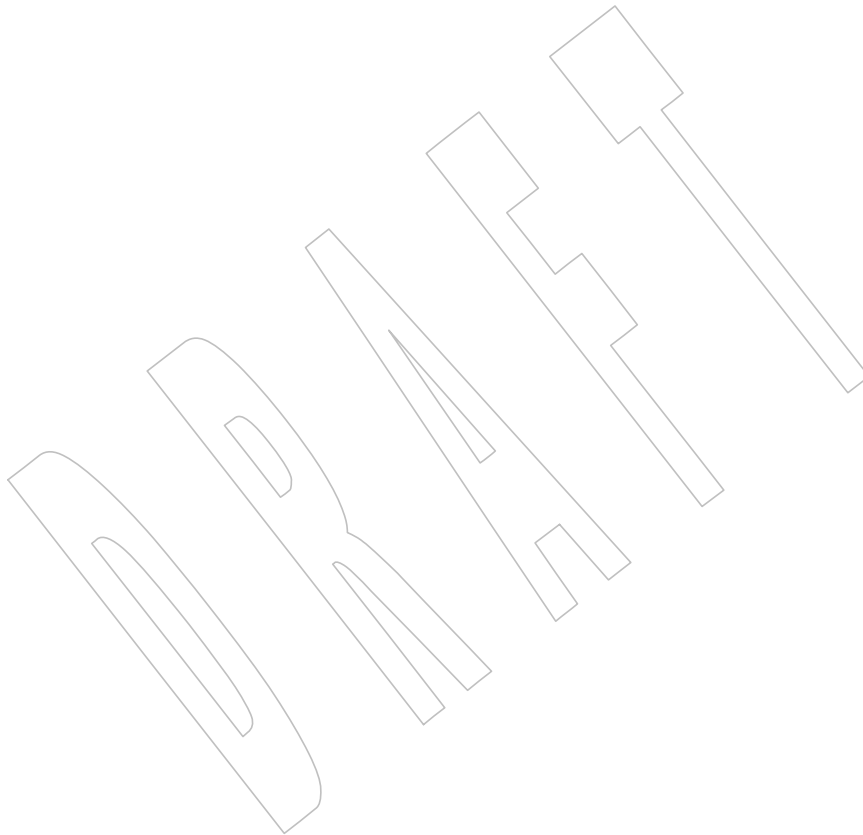
27440 **DESCRIPTION**
27441 Refer to *fegetenv()*.



27442 **NAME**
27443 fesetexceptflag — set floating-point status flags

27444 **SYNOPSIS**
27445 #include <fenv.h>
27446 int fesetexceptflag(const fexcept_t *flagp, int excepts);

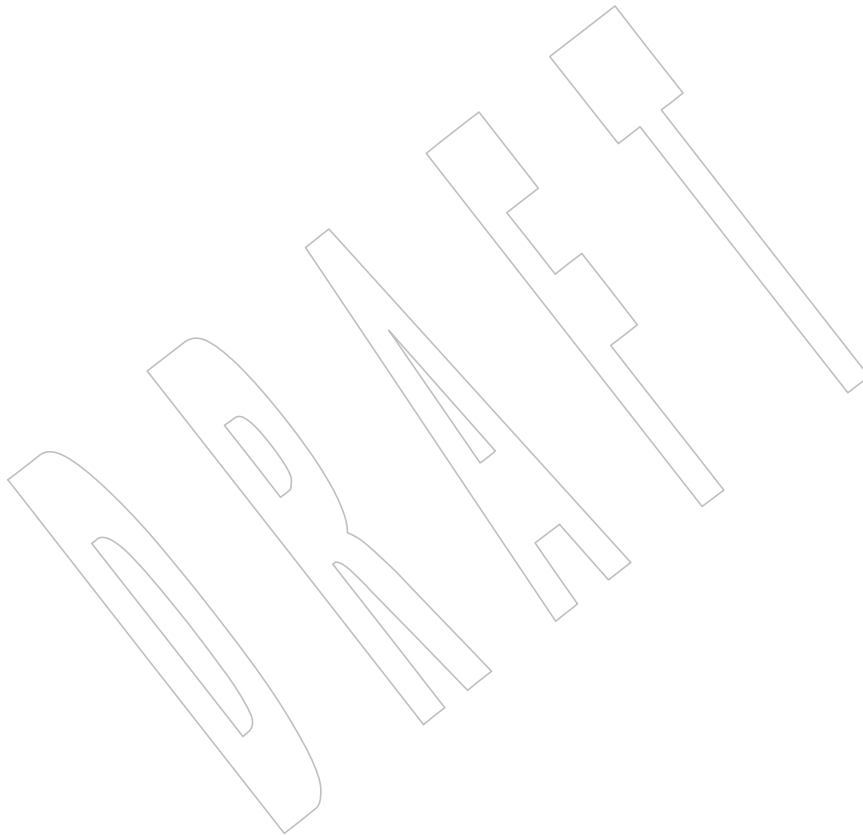
27447 **DESCRIPTION**
27448 Refer to *fegetexceptflag()*.



27449 **NAME**
27450 fesetround — set current rounding direction

27451 **SYNOPSIS**
27452 #include <fenv.h>
27453 int fesetround(int round);

27454 **DESCRIPTION**
27455 Refer to *fegetround()*.



27456 **NAME**

27457 fetestexcept — test floating-point exception flags

27458 **SYNOPSIS**

27459 #include <fenv.h>

27460 int fetestexcept(int *excepts*);27461 **DESCRIPTION**

27462 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27463 conflict between the requirements described here and the ISO C standard is unintentional. This
 27464 volume of POSIX.1-200x defers to the ISO C standard.

27465 The *fetestexcept()* function shall determine which of a specified subset of the floating-point
 27466 exception flags are currently set. The *excepts* argument specifies the floating-point status flags to
 27467 be queried.

27468 **RETURN VALUE**

27469 The *fetestexcept()* function shall return the value of the bitwise-inclusive OR of the floating-point
 27470 exception macros corresponding to the currently set floating-point exceptions included in
 27471 *excepts*.

27472 **ERRORS**

27473 No errors are defined.

27474 **EXAMPLES**

27475 The following example calls function *f()* if an invalid exception is set, and then function *g()* if an
 27476 overflow exception is set:

```
27477 #include <fenv.h>
27478 /* ... */
27479 {
27480     #pragma STDC FENV_ACCESS ON
27481     int set_excepts;
27482     feclearexcept(FE_INVALID | FE_OVERFLOW);
27483     // maybe raise exceptions
27484     set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
27485     if (set_excepts & FE_INVALID) f();
27486     if (set_excepts & FE_OVERFLOW) g();
27487     /* ... */
27488 }
```

27489 **APPLICATION USAGE**

27490 None.

27491 **RATIONALE**

27492 None.

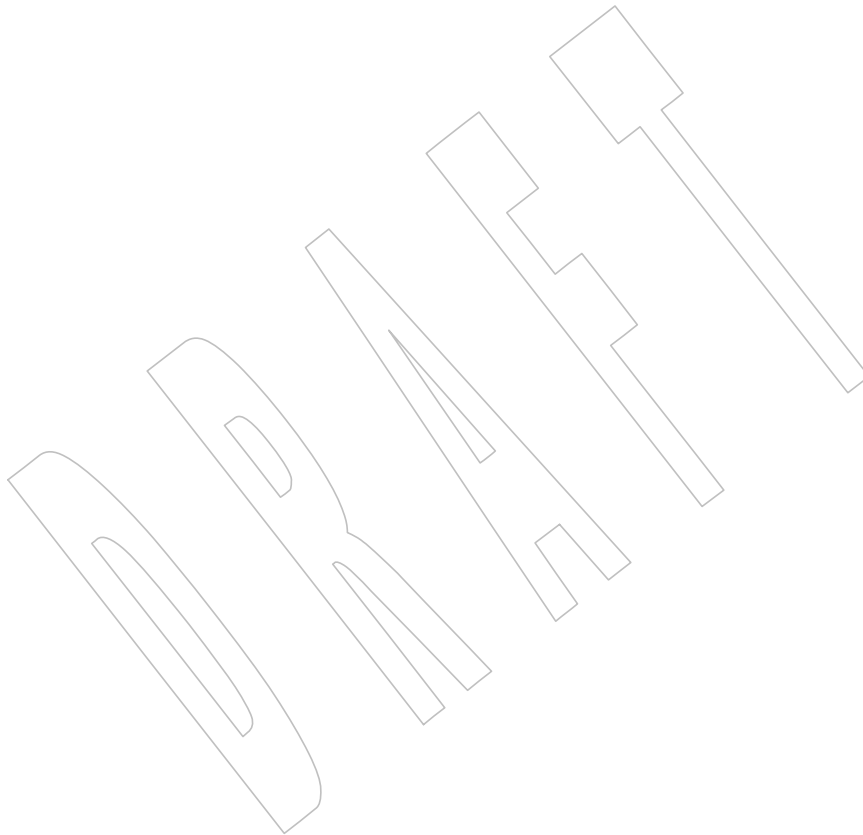
27493 **FUTURE DIRECTIONS**

27494 None.

27495 **SEE ALSO**27496 [feclearexcept\(\)](#), [fetestexceptflag\(\)](#), [feraiseexcept\(\)](#)27497 XBD [<fenv.h>](#)

27498
27499**CHANGE HISTORY**

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.



27500 **NAME**
 27501 feupdateenv — update floating-point environment

27502 **SYNOPSIS**
 27503 #include <fenv.h>
 27504 int feupdateenv(const fenv_t *envp);

27505 **DESCRIPTION**
 27506 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27507 conflict between the requirements described here and the ISO C standard is unintentional. This
 27508 volume of POSIX.1-200x defers to the ISO C standard.

27509 The *feupdateenv()* function shall attempt to save the currently raised floating-point exceptions in
 27510 its automatic storage, attempt to install the floating-point environment represented by the object
 27511 pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument
 27512 *envp* shall point to an object set by a call to *feholdexcept()* or *fegetenv()*, or equal a floating-point
 27513 environment macro.

27514 **RETURN VALUE**
 27515 The *feupdateenv()* function shall return a zero value if and only if all the required actions were
 27516 successfully carried out.

27517 **ERRORS**
 27518 No errors are defined.

27519 **EXAMPLES**
 27520 The following example shows sample code to hide spurious underflow floating-point
 27521 exceptions:

```
27522 #include <fenv.h>
27523 double f(double x)
27524 {
27525     #pragma STDC FENV_ACCESS ON
27526     double result;
27527     fenv_t save_env;
27528     feholdexcept(&save_env);
27529     // compute result
27530     if (/* test spurious underflow */)
27531         feclearexcept(FE_UNDERFLOW);
27532     feupdateenv(&save_env);
27533     return result;
27534 }
```

27535 **APPLICATION USAGE**
 27536 None.

27537 **RATIONALE**
 27538 None.

27539 **FUTURE DIRECTIONS**
 27540 None.

27541 **SEE ALSO**
 27542 *fegetenv()*, *feholdexcept()*
 27543 XBD <fenv.h>

27544

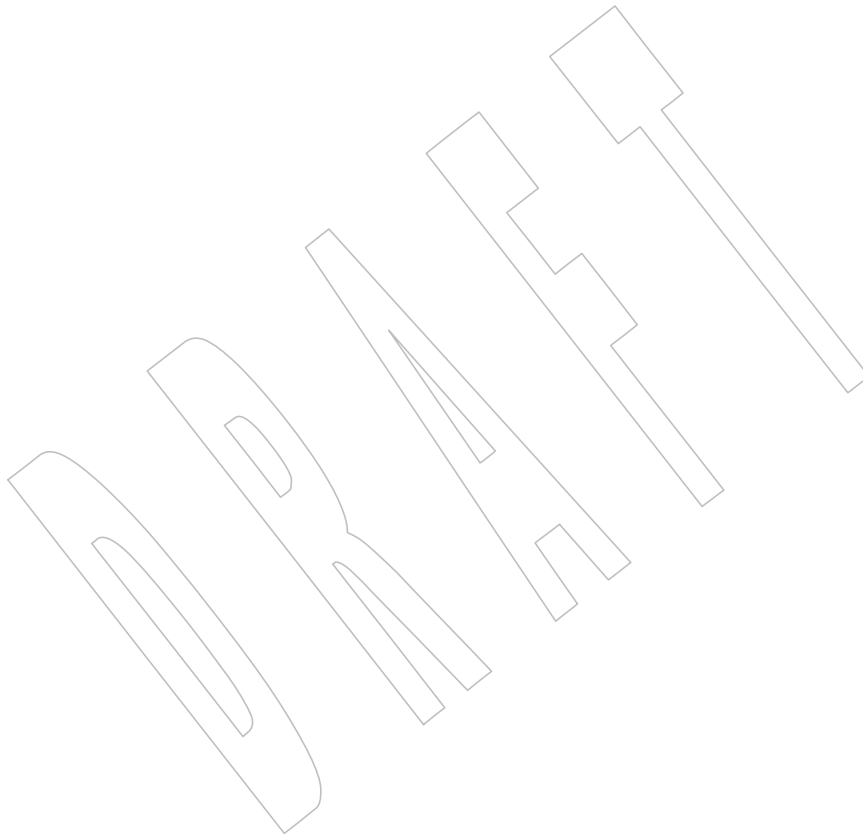
27545

27546

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

ISO/IEC 9899: 1999 standard, Technical Corrigendum 1 is incorporated.



27547 **NAME**
27548 fexecve — execute a file

27549 **SYNOPSIS**
27550 #include <unistd.h>
27551 int fexecve(int *fd*, char *const *argv*[], char *const *envp*[]);

27552 **DESCRIPTION**
27553 Refer to *exec*.



27554 **NAME**27555 `fflush` — flush a stream27556 **SYNOPSIS**27557 `#include <stdio.h>`27558 `int fflush(FILE *stream);`27559 **DESCRIPTION**27560 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27561 conflict between the requirements described here and the ISO C standard is unintentional. This
27562 volume of POSIX.1-200x defers to the ISO C standard.27563 If *stream* points to an output stream or an update stream in which the most recent operation was
27564 CX not input, *fflush()* shall cause any unwritten data for that stream to be written to the file, and the
27565 last data modification and last file status change timestamps of the underlying file shall be
27566 marked for update.27567 If *stream* is a null pointer, *fflush()* shall perform this flushing action on all streams for which the
27568 behavior is defined above.27569 CX For a stream open for reading, if the file is not already at EOF, and the file is one capable of
27570 seeking, the file offset of the underlying open file description shall be adjusted so that the next
27571 operation on the open file description deals with the byte after the last one read from or written
27572 to the stream being flushed.27573 **RETURN VALUE**27574 Upon successful completion, *fflush()* shall return 0; otherwise, it shall set the error indicator for
27575 CX the stream, return EOF, and set *errno* to indicate the error.27576 **ERRORS**27577 The *fflush()* function shall fail if:27578 CX **[EAGAIN]** The O_NONBLOCK flag is set for the file descriptor underlying *stream* and
27579 the thread would be delayed in the write operation.27580 CX **[EBADF]** The file descriptor underlying *stream* is not valid.27581 CX **[EFBIG]** An attempt was made to write a file that exceeds the maximum file size.27582 XSI **[EFBIG]** An attempt was made to write a file that exceeds the file size limit of the
27583 process.27584 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the
27585 offset maximum associated with the corresponding stream.27586 CX **[EINTR]** The *fflush()* function was interrupted by a signal.27587 CX **[EIO]** The process is a member of a background process group attempting to write to
27588 its controlling terminal, TOSTOP is set, the process is neither ignoring nor
27589 blocking SIGTTOU, and the process group of the process is orphaned. This
27590 error may also be returned under implementation-defined conditions.27591 CX **[ENOMEM]** The underlying stream was created by *open_memstream()* or
27592 *open_wmemstream()* and insufficient memory is available.27593 CX **[ENOSPC]** There was no free space remaining on the device containing the file or in the
27594 buffer used by the *fmemopen()* function.

27595 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by
27596 any process. A SIGPIPE signal shall also be sent to the thread.

27597 The *fflush()* function may fail if:

27598 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
27599 capabilities of the device.

27600 EXAMPLES

27601 Sending Prompts to Standard Output

27602 The following example uses *printf()* calls to print a series of prompts for information the user
27603 must enter from standard input. The *fflush()* calls force the output to standard output. The
27604 *fflush()* function is used because standard output is usually buffered and the prompt may not
27605 immediately be printed on the output or terminal. The *getline()* function calls read strings from
27606 standard input and place the results in variables, for use later in the program.

```
27607 char *user;
27608 char *oldpasswd;
27609 char *newpasswd;
27610 ssize_t llen;
27611 size_t blen;
27612 struct termios term;
27613 tcflag_t saveflag;

27614 printf("User name: ");
27615 fflush(stdout);
27616 blen = 0;
27617 llen = getline(&user, &blen, stdin);
27618 user[llen-1] = 0;
27619 tcgetattr(fileno(stdin), &term);
27620 saveflag = term.c_lflag;
27621 term.c_lflag &= ~ECHO;
27622 tcsetattr(fileno(stdin), TCSANOW, &term);
27623 printf("Old password: ");
27624 fflush(stdout);
27625 blen = 0;
27626 llen = getline(&oldpasswd, &blen, stdin);
27627 oldpasswd[llen-1] = 0;

27628 printf("\nNew password: ");
27629 fflush(stdout);
27630 blen = 0;
27631 llen = getline(&newpasswd, &blen, stdin);
27632 newpasswd[llen-1] = 0;
27633 term.c_lflag = saveflag;
27634 tcsetattr(fileno(stdin), TCSANOW, &term);
27635 free(user);
27636 free(oldpasswd);
27637 free(newpasswd);
```

27638 APPLICATION USAGE

27639 None.

27640
27641
27642
27643**RATIONALE**

Data buffered by the system may make determining the validity of the position of the current file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush()* on streams open for *read()* is not mandated by POSIX.1-200x.

27644
27645**FUTURE DIRECTIONS**

None.

27646
27647**SEE ALSO**

fmemopen(), *getrlimit()*, *open_memstream()*, *ulimit*

27648

XBD [<stdio.h>](#)

27649
27650**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

27651
27652**Issue 5**

Large File Summit extensions are added.

27653
27654**Issue 6**

Extensions beyond the ISO C standard are marked.

27655
27656

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

27657
27658

- The [EFBIG] error is added as part of the large file support extensions.
- The [ENXIO] optional error condition is added.

27659
27660

The RETURN VALUE section is updated to note that the error indicator shall be set for the stream. This is for alignment with the ISO/IEC 9899:1999 standard.

27661
27662
27663

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/31 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

27664
27665
27666**Issue 7**

Austin Group Interpretation 1003.1-2001 #002 is applied, clarifying the interaction of file descriptors and streams.

27667
27668

The [ENOSPC] error condition is updated and the [ENOMEM] error is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

27669

The EXAMPLES section is revised.

27670

Changes are made related to support for finegrained timestamps.

27671 **NAME**
 27672 ffs — find first set bit

27673 **SYNOPSIS**
 27674 XSI `#include <strings.h>`
 27675 `int ffs(int i);`

27676 **DESCRIPTION**
 27677 The `ffs()` function shall find the first bit set (beginning with the least significant bit) in *i*, and
 27678 return the index of that bit. Bits are numbered starting at one (the least significant bit).

27679 **RETURN VALUE**
 27680 The `ffs()` function shall return the index of the first bit set. If *i* is 0, then `ffs()` shall return 0.

27681 **ERRORS**
 27682 No errors are defined.

27683 **EXAMPLES**
 27684 None.

27685 **APPLICATION USAGE**
 27686 None.

27687 **RATIONALE**
 27688 None.

27689 **FUTURE DIRECTIONS**
 27690 None.

27691 **SEE ALSO**
 27692 XBD [<strings.h>](#)

27693 **CHANGE HISTORY**
 27694 First released in Issue 4, Version 2.

27695 **Issue 5**
 27696 Moved from X/OPEN UNIX extension to BASE.

27697 **NAME**27698 `fgetc` — get a byte from a stream27699 **SYNOPSIS**27700 `#include <stdio.h>`27701 `int fgetc(FILE *stream);`27702 **DESCRIPTION**

27703 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27704 conflict between the requirements described here and the ISO C standard is unintentional. This
 27705 volume of POSIX.1-200x defers to the ISO C standard.

27706 If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is
 27707 present, the *fgetc()* function shall obtain the next byte as an **unsigned char** converted to an **int**,
 27708 from the input stream pointed to by *stream*, and advance the associated file position indicator for
 27709 the stream (if defined). Since *fgetc()* operates on bytes, reading a character consisting of multiple
 27710 bytes (or “a multi-byte character”) may require multiple calls to *fgetc()*.

27711 CX The *fgetc()* function may mark the last data access timestamp of the file associated with *stream*
 27712 for update. The last data access timestamp shall be marked for update by the first successful
 27713 execution of *fgetc()*, *fgets()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()* using *stream* that
 27714 returns data not supplied by a prior call to *ungetc()*.

27715 **RETURN VALUE**

27716 Upon successful completion, *fgetc()* shall return the next byte from the input stream pointed to
 27717 by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the
 27718 end-of-file indicator for the stream shall be set and *fgetc()* shall return EOF. If a read error occurs,
 27719 CX the error indicator for the stream shall be set, *fgetc()* shall return EOF, and shall set *errno* to
 27720 indicate the error.

27721 **ERRORS**27722 The *fgetc()* function shall fail if data needs to be read and:

27723 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying *stream* and
 27724 the thread would be delayed in the *fgetc()* operation.

27725 CX **[EBADF]** The file descriptor underlying *stream* is not a valid file descriptor open for
 27726 reading.

27727 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data
 27728 was transferred.

27729 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process
 27730 group attempting to read from its controlling terminal, and either the process
 27731 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
 27732 This error may also be generated for implementation-defined reasons.

27733 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the
 27734 offset maximum associated with the corresponding stream.

27735 The *fgetc()* function may fail if:

27736 CX **[ENOMEM]** Insufficient storage space is available.

27737 CX **[ENXIO]** A request was made of a nonexistent device, or the request was outside the
 27738 capabilities of the device.

27739

EXAMPLES

27740

None.

27741

APPLICATION USAGE

27742

If the integer value returned by *fgetc()* is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

27743

27744

27745

The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an end-of-file condition.

27746

27747

RATIONALE

27748

None.

27749

FUTURE DIRECTIONS

27750

None.

27751

SEE ALSO

27752

feof(), *ferror()*, *fgets()*, *fread()*, *fscanf()*, *getchar()*, *getc()*, *gets()*, *ungetc()*

27753

XBD [<stdio.h>](#)

27754

CHANGE HISTORY

27755

First released in Issue 1. Derived from Issue 1 of the SVID.

27756

Issue 5

27757

Large File Summit extensions are added.

27758

Issue 6

27759

Extensions beyond the ISO C standard are marked.

27760

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

27761

27762

- The [EIO] and [Eoverflow] mandatory error conditions are added.

27763

- The [ENOMEM] and [ENXIO] optional error conditions are added.

27764

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

27765

- The DESCRIPTION is updated to clarify the behavior when the end-of-file indicator for the input stream is not set.

27766

27767

- The RETURN VALUE section is updated to note that the error indicator shall be set for the stream.

27768

27769

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/32 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

27770

27771

27772

Issue 7

27773

Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark the last data access timestamp for update.

27774

27775 **NAME**27776 `fgetpos` — get current file position information27777 **SYNOPSIS**27778 `#include <stdio.h>`27779 `int fgetpos(FILE *restrict stream, fpos_t *restrict pos);`27780 **DESCRIPTION**27781 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27782 conflict between the requirements described here and the ISO C standard is unintentional. This
27783 volume of POSIX.1-200x defers to the ISO C standard.27784 The `fgetpos()` function shall store the current values of the parse state (if any) and file position
27785 indicator for the stream pointed to by `stream` in the object pointed to by `pos`. The value stored
27786 contains unspecified information usable by `fsetpos()` for repositioning the stream to its position
27787 at the time of the call to `fgetpos()`.27788 **RETURN VALUE**27789 Upon successful completion, `fgetpos()` shall return 0; otherwise, it shall return a non-zero value
27790 and set `errno` to indicate the error.27791 **ERRORS**27792 The `fgetpos()` function shall fail if:27793 CX **[EOVERFLOW]** The current value of the file position cannot be represented correctly in an
27794 object of type `fpos_t`.27795 The `fgetpos()` function may fail if:27796 CX **[EBADF]** The file descriptor underlying `stream` is not valid.27797 CX **[ESPIPE]** The file descriptor underlying `stream` is associated with a pipe, FIFO, or socket.27798 **EXAMPLES**

27799 None.

27800 **APPLICATION USAGE**

27801 None.

27802 **RATIONALE**

27803 None.

27804 **FUTURE DIRECTIONS**

27805 None.

27806 **SEE ALSO**27807 `fopen()`, `ftell()`, `rewind()`, `ungetc()`27808 XBD `<stdio.h>`27809 **CHANGE HISTORY**

27810 First released in Issue 4. Derived from the ISO C standard.

27811 **Issue 5**

27812 Large File Summit extensions are added.

27813
27814
27815
27816
27817
27818
27819

Issue 6

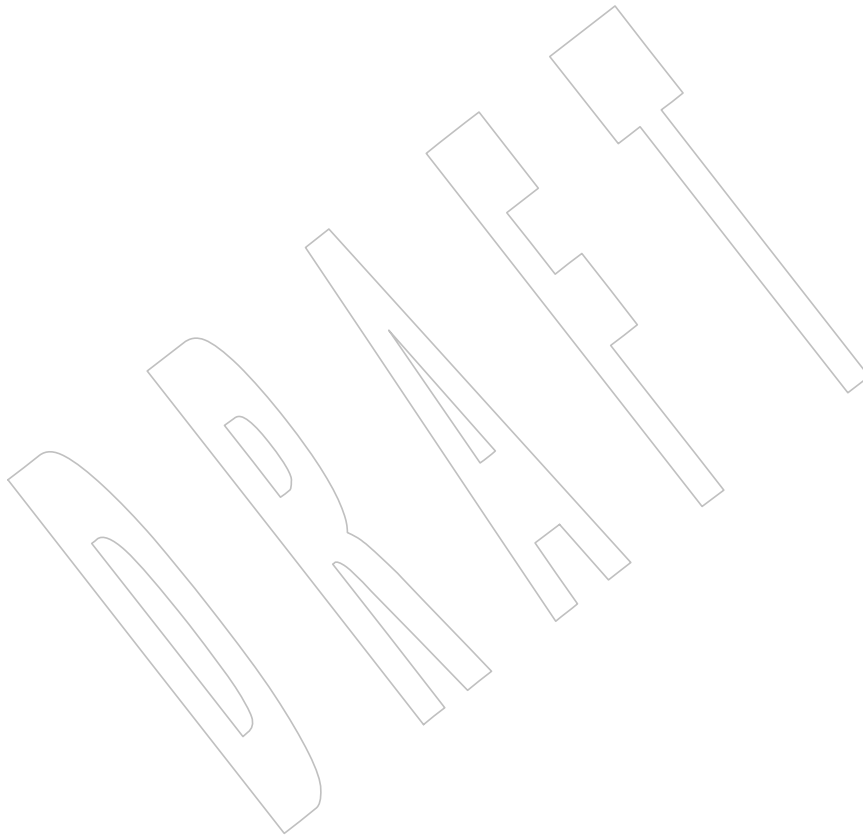
Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EBADF] and [ESPIPE] optional error conditions are added.

An additional [ESPIPE] error condition is added for sockets.

The prototype for *fgetpos()* is changed for alignment with the ISO/IEC 9899:1999 standard.



27820 **NAME**27821 `fgets` — get a string from a stream27822 **SYNOPSIS**27823 `#include <stdio.h>`27824 `char *fgets(char *restrict s, int n, FILE *restrict stream);`27825 **DESCRIPTION**

27826 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27827 conflict between the requirements described here and the ISO C standard is unintentional. This
 27828 volume of POSIX.1-200x defers to the ISO C standard.

27829 The `fgets()` function shall read bytes from *stream* into the array pointed to by *s*, until *n*−1 bytes
 27830 are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered.
 27831 The string is then terminated with a null byte.

27832 CX The `fgets()` function may mark the last data access timestamp of the file associated with *stream*
 27833 for update. The last data access timestamp shall be marked for update by the first successful
 27834 execution of `fgetc()`, `fgets()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()`, or `scanf()` using *stream* that
 27835 returns data not supplied by a prior call to `ungetc()`.

27836 **RETURN VALUE**

27837 Upon successful completion, `fgets()` shall return *s*. If the stream is at end-of-file, the end-of-file
 27838 indicator for the stream shall be set and `fgets()` shall return a null pointer. If a read error occurs,
 27839 CX the error indicator for the stream shall be set, `fgets()` shall return a null pointer, and shall set
 27840 `errno` to indicate the error.

27841 **ERRORS**27842 Refer to `fgetc()`.27843 **EXAMPLES**27844 **Reading Input**

27845 The following example uses `fgets()` to read each line of input. {LINE_MAX}, which defines the
 27846 maximum size of the input line, is defined in the <limits.h> header.

```
27847 #include <stdio.h>
27848 ...
27849 char line[LINE_MAX];
27850 ...
27851 while (fgets(line, LINE_MAX, fp) != NULL) {
27852     ...
27853 }
27854 ...
```

27855 **APPLICATION USAGE**

27856 None.

27857 **RATIONALE**

27858 None.

27859 **FUTURE DIRECTIONS**

27860 None.

27861

SEE ALSO

27862

fgetc(), *fopen()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *getdelim()*, *gets()*, *ungetc()*

27863

XBD <stdio.h>

27864

CHANGE HISTORY

27865

First released in Issue 1. Derived from Issue 1 of the SVID.

27866

Issue 6

27867

Extensions beyond the ISO C standard are marked.

27868

The prototype for *fgets()* is changed for alignment with the ISO/IEC 9899:1999 standard.

27869

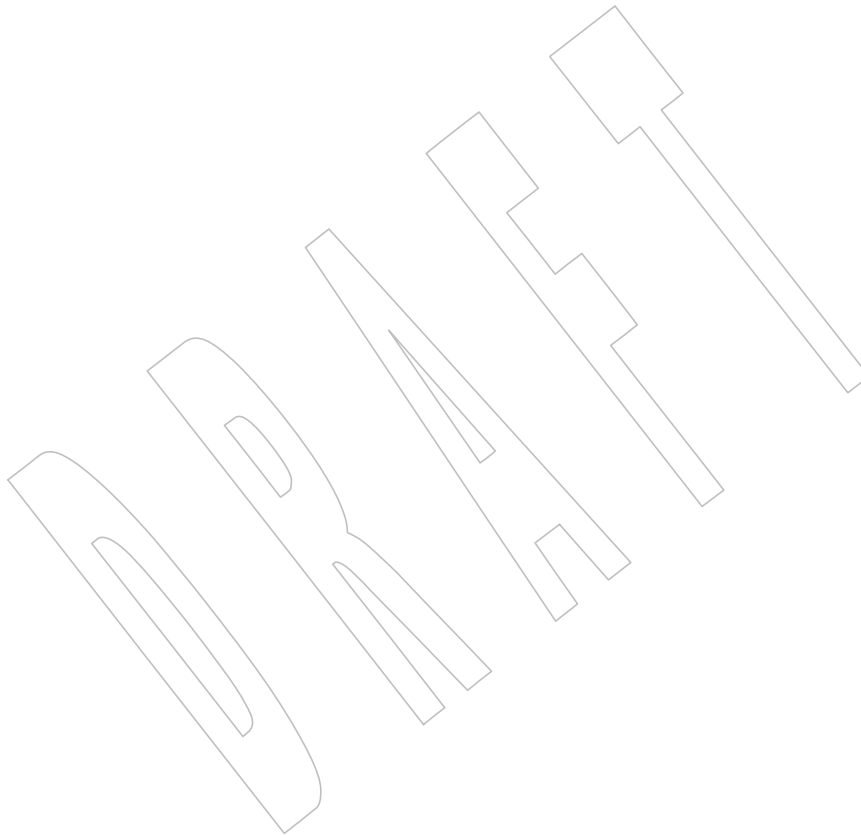
Issue 7

27870

Austin Group Interpretation 1003.1-2001 #051 is applied, updating the list of functions that mark

27871

the last data access timestamp for update.



27872 **NAME**27873 `fgetwc` — get a wide-character code from a stream27874 **SYNOPSIS**27875 `#include <stdio.h>`27876 `#include <wchar.h>`27877 `wint_t fgetwc(FILE *stream);`27878 **DESCRIPTION**27879 CX The functionality described on this reference page is aligned with the ISO C standard. Any
27880 conflict between the requirements described here and the ISO C standard is unintentional. This
27881 volume of POSIX.1-200x defers to the ISO C standard.27882 The `fgetwc()` function shall obtain the next character (if present) from the input stream pointed to
27883 by `stream`, convert that to the corresponding wide-character code, and advance the associated file
27884 position indicator for the stream (if defined).

27885 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

27886 CX The `fgetwc()` function may mark the last data access timestamp of the file associated with `stream` |
27887 for update. The last data access timestamp shall be marked for update by the first successful |
27888 execution of `fgetc()`, `fgets()`, `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()`, or `scanf()`
27889 using `stream` that returns data not supplied by a prior call to `ungetc()` or `ungetwc()`.27890 **RETURN VALUE**27891 Upon successful completion, the `fgetwc()` function shall return the wide-character code of the
27892 character read from the input stream pointed to by `stream` converted to a type `wint_t`. If the end-
27893 of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for
27894 the stream shall be set and `fgetwc()` shall return WEOF. If a read error occurs, the error indicator
27895 CX for the stream shall be set, `fgetwc()` shall return WEOF, and shall set `errno` to indicate the error. If
27896 an encoding error occurs, the error indicator for the stream shall be set, `fgetwc()` shall return
27897 WEOF, and shall set `errno` to indicate the error.27898 **ERRORS**27899 The `fgetwc()` function shall fail if data needs to be read and:27900 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and
27901 the thread would be delayed in the `fgetwc()` operation.27902 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for
27903 reading.27904 **[EILSEQ]** The data obtained from the input stream does not form a valid character.27905 CX **[EINTR]** The read operation was terminated due to the receipt of a signal, and no data
27906 was transferred.27907 CX **[EIO]** A physical I/O error has occurred, or the process is in a background process
27908 group attempting to read from its controlling terminal, and either the process
27909 is ignoring or blocking the SIGTTIN signal or the process group is orphaned.
27910 This error may also be generated for implementation-defined reasons.27911 CX **[EOVERFLOW]** The file is a regular file and an attempt was made to read at or beyond the
27912 offset maximum associated with the corresponding stream.

27913 The *fgetwc()* function may fail if:

27914 CX [ENOMEM] Insufficient storage space is available.

27915 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
27916 capabilities of the device.

27917 EXAMPLES

27918 None.

27919 APPLICATION USAGE

27920 The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an
27921 end-of-file condition.

27922 RATIONALE

27923 None.

27924 FUTURE DIRECTIONS

27925 None.

27926 SEE ALSO

27927 *feof()*, *ferror()*, *fopen()*

27928 XBD <stdio.h>, <wchar.h>

27929 CHANGE HISTORY

27930 First released in Issue 4. Derived from the MSE working draft.

27931 Issue 5

27932 The Optional Header (OH) marking is removed from <stdio.h>.

27933 Large File Summit extensions are added.

27934 Issue 6

27935 Extensions beyond the ISO C standard are marked.

27936 The following new requirements on POSIX implementations derive from alignment with the
27937 Single UNIX Specification:

- 27938 • The [EIO] and [EOVERFLOW] mandatory error conditions are added.
- 27939 • The [ENOMEM] and [ENXIO] optional error conditions are added.

27940 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/33 is applied, updating the [EAGAIN]
27941 error in the ERRORS section from “the process would be delayed” to “the thread would be
27942 delayed”.

27943 Issue 7

27944 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.

27945 Changes are made related to support for finegrained timestamps.

27946 **NAME**
 27947 `fgetws` — get a wide-character string from a stream

27948 **SYNOPSIS**
 27949 `#include <stdio.h>`
 27950 `#include <wchar.h>`

27951 `wchar_t *fgetws(wchar_t *restrict ws, int n,`
 27952 `FILE *restrict stream);`

27953 DESCRIPTION

27954 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 27955 conflict between the requirements described here and the ISO C standard is unintentional. This
 27956 volume of POSIX.1-200x defers to the ISO C standard.

27957 The `fgetws()` function shall read characters from the *stream*, convert these to the corresponding
 27958 wide-character codes, place them in the `wchar_t` array pointed to by *ws*, until *n*−1 characters are
 27959 read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is
 27960 encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character
 27961 code.

27962 If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

27963 CX The `fgetws()` function may mark the last data access timestamp of the file associated with *stream*
 27964 for update. The last data access timestamp shall be marked for update by the first successful
 27965 execution of `fgetc()`, `fgets()`, `fgetwc()`, `fgetws()`, `fread()`, `fscanf()`, `getc()`, `getchar()`, `gets()`, or `scanf()`
 27966 using *stream* that returns data not supplied by a prior call to `ungetc()` or `ungetwc()`.

27967 RETURN VALUE

27968 Upon successful completion, `fgetws()` shall return *ws*. If the end-of-file indicator for the stream is
 27969 set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and
 27970 `fgetws()` shall return a null pointer. If a read error occurs, the error indicator for the stream shall
 27971 CX be set, `fgetws()` shall return a null pointer, and shall set `errno` to indicate the error.

27972 ERRORS

27973 Refer to `fgetwc()`.

27974 EXAMPLES

27975 None.

27976 APPLICATION USAGE

27977 None.

27978 RATIONALE

27979 None.

27980 FUTURE DIRECTIONS

27981 None.

27982 SEE ALSO

27983 `fopen()`, `fread()`

27984 XBD `<stdio.h>`, `<wchar.h>`

27985 CHANGE HISTORY

27986 First released in Issue 4. Derived from the MSE working draft.

- 27987 **Issue 5**
27988 The Optional Header (OH) marking is removed from `<stdio.h>`.
- 27989 **Issue 6**
27990 Extensions beyond the ISO C standard are marked.
27991 The prototype for `fgetws()` is changed for alignment with the ISO/IEC 9899:1999 standard.
- 27992 **Issue 7**
27993 Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.
27994 Changes are made related to support for finegrained timestamps. +



27995 **NAME**
 27996 `fileno` — map a stream pointer to a file descriptor

27997 **SYNOPSIS**

27998 CX

```
#include <stdio.h>
```


 27999

```
int fileno(FILE *stream);
```

28000 **DESCRIPTION**

28001 The `fileno()` function shall return the integer file descriptor associated with the stream pointed to
 28002 by `stream`.

28003 **RETURN VALUE**

28004 Upon successful completion, `fileno()` shall return the integer value of the file descriptor
 28005 associated with `stream`. Otherwise, the value `-1` shall be returned and `errno` set to indicate the
 28006 error.

28007 **ERRORS**

28008 The `fileno()` function may fail if:
 28009 [EBADF] The `stream` argument is not a valid stream.

28010 **EXAMPLES**

28011 None.

28012 **APPLICATION USAGE**

28013 None.

28014 **RATIONALE**

28015 Without some specification of which file descriptors are associated with these streams, it is
 28016 impossible for an application to set up the streams for another application it starts with `fork()`
 28017 and `exec`. In particular, it would not be possible to write a portable version of the `sh` command
 28018 interpreter (although there may be other constraints that would prevent that portability).

28019 **FUTURE DIRECTIONS**

28020 None.

28021 **SEE ALSO**

28022 [Section 2.5.1](#) (on page 470), [dirfd\(\)](#), [fdopen\(\)](#), [fopen\(\)](#), [stdin](#)

28023 XBD [<stdio.h>](#)

28024 **CHANGE HISTORY**

28025 First released in Issue 1. Derived from Issue 1 of the SVID.

28026 **Issue 6**

28027 The following new requirements on POSIX implementations derive from alignment with the
 28028 Single UNIX Specification:

- 28029 • The [EBADF] optional error condition is added.

28030 **NAME**

28031 flockfile, ftrylockfile, funlockfile — stdio locking functions

28032 **SYNOPSIS**

```
28033 CX      #include <stdio.h>
28034
28034 void flockfile(FILE *file);
28035 int ftrylockfile(FILE *file);
28036 void funlockfile(FILE *file);
```

28037 **DESCRIPTION**

28038 These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects.
 28039 These functions can be used by a thread to delineate a sequence of I/O statements that are
 28040 executed as a unit.

28041 The *flockfile()* function shall acquire for a thread ownership of a (**FILE ***) object.

28042 The *ftrylockfile()* function shall acquire for a thread ownership of a (**FILE ***) object if the object is
 28043 available; *ftrylockfile()* is a non-blocking version of *flockfile()*.

28044 The *funlockfile()* function shall relinquish the ownership granted to the thread. The behavior is
 28045 undefined if a thread other than the current owner calls the *funlockfile()* function.

28046 The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This
 28047 count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is
 28048 unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***)
 28049 object. When the *flockfile()* function is called, if the count is zero or if the count is positive and
 28050 the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread
 28051 shall be suspended, waiting for the count to return to zero. Each call to *funlockfile()* shall
 28052 decrement the count. This allows matching calls to *flockfile()* (or successful calls to *ftrylockfile()*)
 28053 and *funlockfile()* to be nested.

28054 All functions that reference (**FILE ***) objects shall behave as if they use *flockfile()* and *funlockfile()*
 28055 internally to obtain ownership of these (**FILE ***) objects.

28056 **RETURN VALUE**

28057 None for *flockfile()* and *funlockfile()*.

28058 The *ftrylockfile()* function shall return zero for success and non-zero to indicate that the lock
 28059 cannot be acquired.

28060 **ERRORS**

28061 No errors are defined.

28062 **EXAMPLES**

28063 None.

28064 **APPLICATION USAGE**

28065 Applications using these functions may be subject to priority inversion, as discussed in XBD |
 28066 [Section 3.284](#) (on page 72).

28067 **RATIONALE**

28068 The *flockfile()* and *funlockfile()* functions provide an orthogonal mutual-exclusion lock for each
 28069 **FILE**. The *ftrylockfile()* function provides a non-blocking attempt to acquire a file lock,
 28070 analogous to *pthread_mutex_trylock()*.

28071 These locks behave as if they are the same as those used internally by *stdio* for thread-safety.
 28072 This both provides thread-safety of these functions without requiring a second level of internal

28073 locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

28074 Application developers and implementors should be aware that there are potential deadlock
 28075 problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested
 28076 via `{_IOLBF}`) require that certain input operations sometimes cause the buffered contents of
 28077 implementation-defined line-buffered output streams to be flushed. If two threads each hold the
 28078 lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring
 28079 **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can
 28080 typically be avoided by acquiring locks on input streams before locks on output streams if a
 28081 thread would be acquiring both.

28082 In summary, threads sharing *stdio* streams with other threads can use *flockfile()* and *funlockfile()*
 28083 to cause sequences of I/O performed by a single thread to be kept bundled. The only case where
 28084 the use of *flockfile()* and *funlockfile()* is required is to provide a scope protecting uses of the
 28085 `*_unlocked` functions/macros. This moves the cost/performance tradeoff to the optimal point.

28086 **FUTURE DIRECTIONS**

28087 None.

28088 **SEE ALSO**

28089 [*getc_unlocked\(\)*](#)

28090 XBD [Section 3.284](#) (on page 72), `<stdio.h>`

28091 **CHANGE HISTORY**

28092 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

28093 **Issue 6**

28094 These functions are marked as part of the Thread-Safe Functions option.

28095 **Issue 7**

28096 The *flockfile()*, *ftrylockfile()*, and *funlockfile()* functions are moved from the Thread-Safe Functions
 28097 option to the Base.

28098 **NAME**
 28099 floor, floorf, floorl — floor function

28100 **SYNOPSIS**
 28101 #include <math.h>
 28102 double floor(double x);
 28103 float floorf(float x);
 28104 long double floorl(long double x);

28105 **DESCRIPTION**
 28106 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28107 conflict between the requirements described here and the ISO C standard is unintentional. This
 28108 volume of POSIX.1-200x defers to the ISO C standard.

28109 These functions shall compute the largest integral value not greater than x .

28110 An application wishing to check for error situations should set *errno* to zero and call
 28111 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 28112 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 28113 zero, an error has occurred.

28114 **RETURN VALUE**
 28115 Upon successful completion, these functions shall return the largest integral value not greater
 28116 than x , expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the
 28117 function.

28118 MX If x is NaN, a NaN shall be returned.

28119 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

28120 XSI If the correct value would cause overflow, a range error shall occur and *floor()*, *floorf()*, and
 28121 *floorl()* shall return the value of the macro `-HUGE_VAL`, `-HUGE_VALF`, and `-HUGE_VALL`,
 28122 respectively.

28123 **ERRORS**
 28124 These functions shall fail if:

28125 XSI **Range Error** The result would cause an overflow.

28126 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28127 then *errno* shall be set to [ERANGE]. If the integer expression
 28128 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 28129 floating-point exception shall be raised.

28130 **EXAMPLES**
 28131 None.

28132 **APPLICATION USAGE**
 28133 The integral value returned by these functions might not be expressible as an **int** or **long**. The
 28134 return value should be tested before assigning it to an integer type to avoid the undefined
 28135 results of an integer overflow.

28136 The *floor()* function can only overflow when the floating-point representation has
 28137 `DBL_MANT_DIG > DBL_MAX_EXP`.

28138 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 28139 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28140
28141
28142
28143
28144
28145
28146
28147
28148
28149
28150
28151
28152
28153
28154
28155
28156
28157

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ceil(), *feclearexcept()*, *fetestexcept()*, *isnan()*

Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *floorf()* and *floorl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

28158 **NAME**

28159 fma, fmaf, fmal — floating-point multiply-add

28160 **SYNOPSIS**

```
28161 #include <math.h>
28162
28162 double fma(double x, double y, double z);
28163 float fmaf(float x, float y, float z);
28164 long double fmal(long double x, long double y, long double z);
```

28165 **DESCRIPTION**

28166 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28167 conflict between the requirements described here and the ISO C standard is unintentional. This
 28168 volume of POSIX.1-200x defers to the ISO C standard.

28169 These functions shall compute $(x * y) + z$, rounded as one ternary operation: they shall compute
 28170 the value (as if) to infinite precision and round once to the result format, according to the
 28171 rounding mode characterized by the value of FLT_ROUNDS.

28172 An application wishing to check for error situations should set *errno* to zero and call
 28173 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 28174 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 28175 zero, an error has occurred.

28176 **RETURN VALUE**

28177 Upon successful completion, these functions shall return $(x * y) + z$, rounded as one ternary
 28178 operation.

28179 MX If the result overflows or underflows, a range error may occur. On systems that support the IEC
 28180 60559 Floating-Point option, if the result overflows a range error shall occur.

28181 If *x* or *y* are NaN, a NaN shall be returned.

28182 If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a
 28183 domain error shall occur, and either a NaN (if supported), or an implementation-defined value
 28184 shall be returned.

28185 If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and
 28186 either a NaN (if supported), or an implementation-defined value shall be returned.

28187 If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a
 28188 domain error may occur.

28189 If $x*y$ is not $0*Inf$ nor $Inf*0$ and *z* is a NaN, a NaN shall be returned.

28190 **ERRORS**

28191 These functions shall fail if:

28192 MX **Domain Error** The value of $x*y+z$ is invalid, or the value $x*y$ is invalid and *z* is not a NaN.

28193 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28194 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 28195 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 28196 shall be raised.

28197 MX **Range Error** The result overflows.

28198 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 28199 then *errno* shall be set to [ERANGE]. If the integer expression
 28200 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow

28201 floating-point exception shall be raised.

28202 These functions may fail if:

28203 MX **Domain Error** The value $x*y$ is invalid and z is a NaN.

28204 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
28205 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
28206 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
28207 shall be raised.

28208 **Range Error** The result underflows.

28209 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
28210 then *errno* shall be set to [ERANGE]. If the integer expression
28211 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
28212 floating-point exception shall be raised.

28213 **Range Error** The result overflows.

28214 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
28215 then *errno* shall be set to [ERANGE]. If the integer expression
28216 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
28217 floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

28221 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
28222 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

28224 In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its
28225 unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT
28226 macro can be used to disallow use of floating multiply-add; and the *fma()* function guarantees
28227 its use where desired. Many current machines provide hardware floating multiply-add
28228 instructions; software implementation can be used for others.

FUTURE DIRECTIONS

None.

SEE ALSO*feclearexcept()*, *fetestexcept()*

XBD Section 4.19 (on page 104), <math.h>

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

28237 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #57 (SD5-XSH-ERN-69) is applied,
28238 adding a “may fail” range error for non-MX systems.

28239 **NAME**

28240 fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

28241 **SYNOPSIS**

28242 #include <math.h>

28243 double fmax(double x, double y);

28244 float fmaxf(float x, float y);

28245 long double fmaxl(long double x, long double y);

28246 **DESCRIPTION**28247 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28248 conflict between the requirements described here and the ISO C standard is unintentional. This
28249 volume of POSIX.1-200x defers to the ISO C standard.28250 MX These functions shall determine the maximum numeric value of their arguments. NaN
28251 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
28252 then these functions shall choose the numeric value.28253 **RETURN VALUE**28254 Upon successful completion, these functions shall return the maximum numeric value of their
28255 arguments.

28256 MX If just one argument is a NaN, the other argument shall be returned.

28257 If *x* and *y* are NaN, a NaN shall be returned.28258 **ERRORS**

28259 No errors are defined.

28260 **EXAMPLES**

28261 None.

28262 **APPLICATION USAGE**

28263 None.

28264 **RATIONALE**

28265 None.

28266 **FUTURE DIRECTIONS**

28267 None.

28268 **SEE ALSO**28269 *fdim()*, *fmin()*

28270 XBD <math.h>

28271 **CHANGE HISTORY**

28272 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28273 **Issue 7**

28274 Austin Group Interpretation 1003.1-2001 #007 is applied.

28275 **NAME**28276 `fmemopen` — open a memory buffer stream28277 **SYNOPSIS**

```
28278 CX #include <stdio.h>
28279 FILE *fmemopen(void *restrict buf, size_t size,
28280 const char *restrict mode);
```

28281 **DESCRIPTION**

28282 The `fmemopen()` function shall associate the buffer given by the `buf` and `size` arguments with a
 28283 stream. The `buf` argument shall be either a null pointer or point to a buffer that is at least `size`
 28284 bytes long.

28285 The `mode` argument is a character string having one of the following values:

28286	<code>r</code> or <code>rb</code>	Open the stream for reading.
28287	<code>w</code> or <code>wb</code>	Open the stream for writing.
28288	<code>a</code> or <code>ab</code>	Append; open the stream for writing at the first null byte.
28289	<code>r+</code> or <code>rb+</code> or <code>r+b</code>	Open the stream for update (reading and writing).
28290	<code>w+</code> or <code>wb+</code> or <code>w+b</code>	Open the stream for update (reading and writing). Truncate the buffer 28291 contents.
28292	<code>a+</code> or <code>ab+</code> or <code>a+b</code>	Append; open the stream for update (reading and writing); the initial 28293 position is at the first null byte.

28294 The character '`b`' shall have no effect.

28295 If a null pointer is specified as the `buf` argument, `fmemopen()` shall allocate `size` bytes of memory
 28296 as if by a call to `malloc()`. This buffer shall be automatically freed when the stream is closed.
 28297 Because this feature is only useful when the stream is opened for updating (because there is no
 28298 way to get a pointer to the buffer) the `fmemopen()` call may fail if the `mode` argument does not
 28299 include a '+'.

28300 The stream maintains a current position in the buffer. This position is initially set to either the
 28301 beginning of the buffer (for `r` and `w` modes) or to the first null byte in the buffer (for `a` modes). If
 28302 no null byte is found in append mode, the initial position is set to one byte after the end of the
 28303 buffer.

28304 If `buf` is a null pointer, the initial position shall always be set to the beginning of the buffer.

28305 The stream also maintains the size of the current buffer contents. For modes `r` and `r+` the size is
 28306 set to the value given by the `size` argument. For modes `w` and `w+` the initial size is zero and for
 28307 modes `a` and `a+` the initial size is either the position of the first null byte in the buffer or the value
 28308 of the `size` argument if no null byte is found.

28309 A read operation on the stream cannot advance the current buffer position behind the current
 28310 buffer size. Reaching the buffer size in a read operation counts as "end-of-file". Null bytes in the
 28311 buffer have no special meaning for reads. The read operation starts at the current buffer position
 28312 of the stream.

28313 A write operation starts either at the current position of the stream (if mode has not specified
 28314 '`a`' as the first character) or at the current size of the stream (if mode had '`a`' as the first
 28315 character). If the current position at the end of the write is larger than the current buffer size, the
 28316 current buffer size is set to the current position. A write operation on the stream cannot advance

28317 the current buffer size behind the size given in the *size* argument.

28318 When a stream open for writing is flushed or closed, a null byte is written at the current position
 28319 or at the end of the buffer, depending on the size of the contents. If a stream open for update is
 28320 flushed or closed and the last write has advanced the current buffer size, a null byte is written at
 28321 the end of the buffer if it fits.

28322 An attempt to seek a memory buffer stream to a negative position or to a position larger than the
 28323 buffer size given in the *size* argument shall fail.

28324 RETURN VALUE

28325 Upon successful completion, *fmemopen()* shall return a pointer to the object controlling the
 28326 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

28327 ERRORS

28328 The *fmemopen()* function shall fail if:

28329 [EINVAL] The *size* argument specifies a buffer size of zero.

28330 The *fmemopen()* function may fail if:

28331 [EINVAL] The value of the *mode* argument is not valid.

28332 [EINVAL] The *buf* argument is a null pointer and the *mode* argument does not include a
 28333 '+' character.

28334 [ENOMEM] The *buf* argument is a null pointer and the allocation of a buffer of length *size*
 28335 has failed.

28336 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

28337 EXAMPLES

```
28338 #include <stdio.h>
28339 static char buffer[] = "foobar";
28340 int
28341 main (void)
28342 {
28343     int ch;
28344     FILE *stream;
28345     stream = fmemopen(buffer, strlen (buffer), "r");
28346     if (stream == NULL)
28347         /* handle error */;
28348     while ((ch = fgetc(stream)) != EOF)
28349         printf("Got %c\n", ch);
28350     fclose(stream);
28351     return (0);
28352 }
```

28353 This program produces the following output:

```
28354 Got f
28355 Got o
28356 Got o
28357 Got b
28358 Got a
28359 Got r
```

fmemopen()28360 **APPLICATION USAGE**

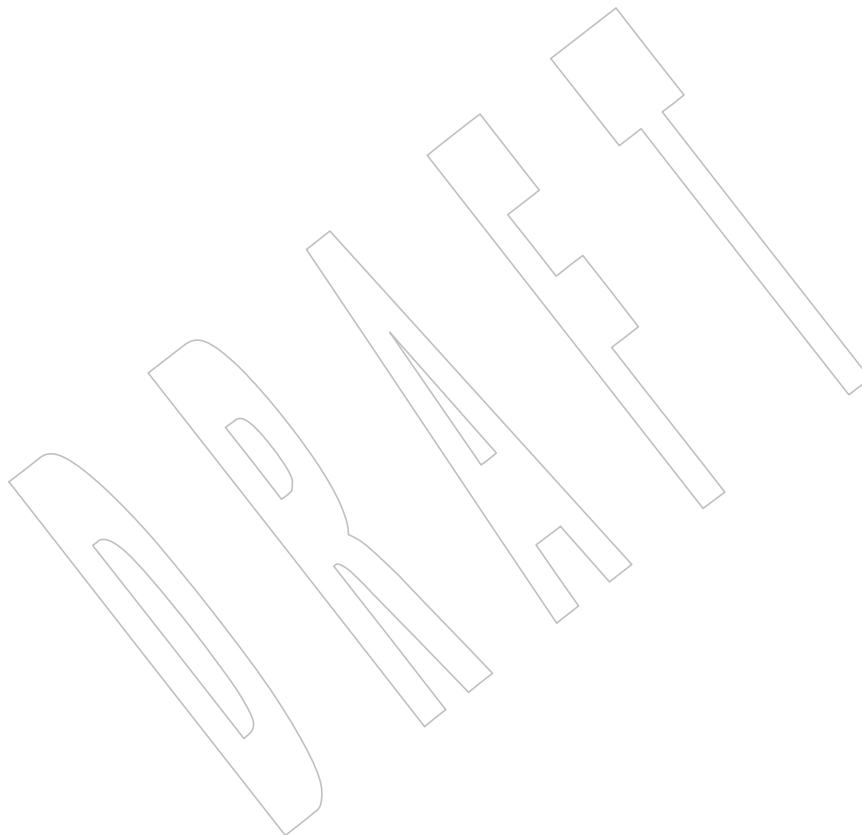
28361 None.

28362 **RATIONALE**28363 This interface has been introduced to eliminate many of the errors encountered in the
28364 construction of strings, notably overflowing of strings. This interface prevents overflow.28365 **FUTURE DIRECTIONS**

28366 None.

28367 **SEE ALSO**28368 *fdopen(), fopen(), freopen(), malloc(), open_memstream()*28369 XBD [<stdio.h>](#)28370 **CHANGE HISTORY**

28371 First released in Issue 7.



28372 **NAME**

28373 fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

28374 **SYNOPSIS**

28375 #include <math.h>

28376 double fmin(double x, double y);

28377 float fminf(float x, float y);

28378 long double fminl(long double x, long double y);

28379 **DESCRIPTION**28380 CX The functionality described on this reference page is aligned with the ISO C standard. Any
28381 conflict between the requirements described here and the ISO C standard is unintentional. This
28382 volume of POSIX.1-200x defers to the ISO C standard.28383 MX These functions shall determine the minimum numeric value of their arguments. NaN
28384 arguments shall be treated as missing data: if one argument is a NaN and the other numeric,
28385 then these functions shall choose the numeric value.28386 **RETURN VALUE**28387 Upon successful completion, these functions shall return the minimum numeric value of their
28388 arguments.

28389 MX If just one argument is a NaN, the other argument shall be returned.

28390 If *x* and *y* are NaN, a NaN shall be returned.28391 **ERRORS**

28392 No errors are defined.

28393 **EXAMPLES**

28394 None.

28395 **APPLICATION USAGE**

28396 None.

28397 **RATIONALE**

28398 None.

28399 **FUTURE DIRECTIONS**

28400 None.

28401 **SEE ALSO**28402 *fdim()*, *fmax()*

28403 XBD <math.h>

28404 **CHANGE HISTORY**

28405 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

28406 **Issue 7**

28407 Austin Group Interpretation 1003.1-2001 #008 is applied.

28408 **NAME**
 28409 `fmod, fmodf, fmodl` — floating-point remainder value function

28410 **SYNOPSIS**
 28411 `#include <math.h>`
 28412 `double fmod(double x, double y);`
 28413 `float fmodf(float x, float y);`
 28414 `long double fmodl(long double x, long double y);`

28415 **DESCRIPTION**
 28416 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28417 conflict between the requirements described here and the ISO C standard is unintentional. This
 28418 volume of POSIX.1-200x defers to the ISO C standard.

28419 These functions shall return the floating-point remainder of the division of x by y .
 28420 An application wishing to check for error situations should set `errno` to zero and call
 28421 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 28422 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 28423 zero, an error has occurred.

28424 **RETURN VALUE**
 28425 These functions shall return the value $x - i * y$, for some integer i such that, if y is non-zero, the
 28426 result has the same sign as x and magnitude less than the magnitude of y .

28427 If the correct value would cause underflow, and is not representable, a range error may occur,
 28428 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

28429 MX If x or y is NaN, a NaN shall be returned.
 28430 If y is zero, a domain error shall occur, and either a NaN (if supported), or an implementation-
 28431 defined value shall be returned.

28432 If x is infinite, a domain error shall occur, and either a NaN (if supported), or an
 28433 implementation-defined value shall be returned.

28434 If x is ± 0 and y is not zero, ± 0 shall be returned.

28435 If x is not infinite and y is $\pm \text{Inf}$, x shall be returned.

28436 If the correct value would cause underflow, and is representable, a range error may occur and
 28437 the correct value shall be returned.

28438 **ERRORS**
 28439 These functions shall fail if:

28440 MX **Domain Error** The x argument is infinite or y is zero.
 28441 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 28442 then `errno` shall be set to [EDOM]. If the integer expression `(math_errhandling`
 28443 `& MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception
 28444 shall be raised.

28445 These functions may fail if:

28446 **Range Error** The result underflows.
 28447 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 28448 then `errno` shall be set to [ERANGE]. If the integer expression
 28449 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow

28450 floating-point exception shall be raised.

28451 **EXAMPLES**

28452 None.

28453 **APPLICATION USAGE**

28454 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
28455 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

28456 **RATIONALE**

28457 None.

28458 **FUTURE DIRECTIONS**

28459 None.

28460 **SEE ALSO**

28461 *feclearexcept()*, *fetestexcept()*, *isnan()*

28462 Section 4.19 (on page 104), <math.h>

28463 **CHANGE HISTORY**

28464 First released in Issue 1. Derived from Issue 1 of the SVID.

28465 **Issue 5**

28466 The DESCRIPTION is updated to indicate how an application should check for an error. This
28467 text was previously published in the APPLICATION USAGE section.

28468 **Issue 6**

28469 The behavior for when the *y* argument is zero is now defined.

28470 The *fmodf()* and *fmodl()* functions are added for alignment with the ISO/IEC 9899:1999
28471 standard.

28472 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
28473 revised to align with the ISO/IEC 9899:1999 standard.

28474 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
28475 marked.

28476 **NAME**28477 `fmtmsg` — display a message in the specified format on standard error and/or a system console28478 **SYNOPSIS**

```
28479 XSI #include <fmtmsg.h>
28480 int fmtmsg(long classification, const char *label, int severity,
28481           const char *text, const char *action, const char *tag);
```

28482 **DESCRIPTION**28483 The `fmtmsg()` function shall display messages in a specified format instead of the traditional
28484 `printf()` function.28485 Based on a message's classification component, `fmtmsg()` shall write a formatted message either
28486 to standard error, to the console, or to both.28487 A formatted message consists of up to five components as defined below. The component
28488 *classification* is not part of a message displayed to the user, but defines the source of the message
28489 and directs the display of the formatted message.

28490 *classification* Contains the sum of identifying values constructed from the constants defined
28491 below. Any one identifier from a subclass may be used in combination with a
28492 single identifier from a different subclass. Two or more identifiers from the
28493 same subclass should not be used together, with the exception of identifiers
28494 from the display subclass. (Both display subclass identifiers may be used so
28495 that messages can be displayed to both standard error and the system
28496 console.)

28497 **Major Classifications**28498 Identifies the source of the condition. Identifiers are: MM_HARD
28499 (hardware), MM_SOFT (software), and MM_FIRM (firmware).28500 **Message Source Subclassifications**28501 Identifies the type of software in which the problem is detected.
28502 Identifiers are: MM_APPL (application), MM_UTIL (utility), and
28503 MM OPSYS (operating system).28504 **Display Subclassifications**28505 Indicates where the message is to be displayed. Identifiers are:
28506 MM_PRINT to display the message on the standard error stream,
28507 MM_CONSOLE to display the message on the system console. One or
28508 both identifiers may be used.28509 **Status Subclassifications**28510 Indicates whether the application can recover from the condition.
28511 Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-
28512 recoverable).28513 An additional identifier, MM_NULLMC, indicates that no classification
28514 component is supplied for the message.28515 *label* Identifies the source of the message. The format is two fields separated by a
28516 colon. The first field is up to 10 bytes, the second is up to 14 bytes.28517 *severity* Indicates the seriousness of the condition. Identifiers for the levels of *severity*
28518 are:

28519		MM_HALT	Indicates that the application has encountered a severe fault and is halting. Produces the string "HALT".
28520			
28521		MM_ERROR	Indicates that the application has detected a fault. Produces the string "ERROR".
28522			
28523		MM_WARNING	Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string "WARNING".
28524			
28525			
28526		MM_INFO	Provides information about a condition that is not in error. Produces the string "INFO".
28527			
28528		MM_NOSEV	Indicates that no severity level is supplied for the message.
28529	<i>text</i>		Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.
28530			
28531			
28532	<i>action</i>		Describes the first step to be taken in the error-recovery process. The <i>fmtmsg()</i> function precedes the action string with the prefix: "TO FIX:". The <i>action</i> string is not limited to a specific size.
28533			
28534			
28535	<i>tag</i>		An identifier that references on-line documentation for the message. Suggested usage is that <i>tag</i> includes the <i>label</i> and a unique identifying number. A sample <i>tag</i> is "XSI:cat:146".
28536			
28537			

28538 The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg()*
 28539 which message components it is to select when writing messages to standard error. The value of
 28540 *MSGVERB* shall be a colon-separated list of optional keywords. Valid keywords are: *label*,
 28541 *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the
 28542 component's value is not the component's null value, *fmtmsg()* shall include that component in
 28543 the message when writing the message to standard error. If *MSGVERB* does not include a
 28544 keyword for a message component, that component shall not be included in the display of the
 28545 message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the
 28546 null string, if its value is not of the correct format, or if it contains keywords other than the valid
 28547 ones listed above, *fmtmsg()* shall select all components.

28548 *MSGVERB* shall determine which components are selected for display to standard error. All
 28549 message components shall be included in console messages.

28550 RETURN VALUE

28551 The *fmtmsg()* function shall return one of the following values:

28552	MM_OK	The function succeeded.
28553	MM_NOTOK	The function failed completely.
28554	MM_NOMSG	The function was unable to generate a message on standard error, but otherwise succeeded.
28555		
28556	MM_NOCON	The function was unable to generate a console message, but otherwise succeeded.
28557		

28558 ERRORS

28559 None.

EXAMPLES

28560

28561 1. The following example of *fmtmsg()*:

28562 `fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",`
 28563 `"refer to cat in user's reference manual", "XSI:cat:001")`

28564 produces a complete message in the specified message format:

28565 `XSI:cat: ERROR: illegal option`
 28566 `TO FIX: refer to cat in user's reference manual XSI:cat:001`

28567 2. When the environment variable *MSGVERB* is set as follows:

28568 `MSGVERB=severity:text:action`

28569 and Example 1 is used, *fmtmsg()* produces:

28570 `ERROR: illegal option`
 28571 `TO FIX: refer to cat in user's reference manual`

APPLICATION USAGE

28572 One or more message components may be systematically omitted from messages generated by
 28573 an application by using the null value of the argument for that component.
 28574

RATIONALE

28575 None.
 28576

FUTURE DIRECTIONS

28577 None.
 28578

SEE ALSO

28579 *fprintf()*

28580 XBD [<fmtmsg.h>](#)

CHANGE HISTORY

28582 First released in Issue 4, Version 2.

Issue 5

28584 Moved from X/OPEN UNIX extension to BASE.
 28585

28586 **NAME**28587 `fnmatch` — match a filename or a pathname28588 **SYNOPSIS**28589 `#include <fnmatch.h>`28590 `int fnmatch(const char *pattern, const char *string, int flags);`28591 **DESCRIPTION**

28592 The `fnmatch()` function shall match patterns as described in XCU [Section 2.13.1](#) (on page 2278) and [Section 2.13.2](#) (on page 2279). It checks the string specified by the `string` argument to see if it matches the pattern specified by the `pattern` argument.

28595 The `flags` argument shall modify the interpretation of `pattern` and `string`. It is the bitwise-inclusive OR of zero or more of the flags defined in `<fnmatch.h>`. If the `FNM_PATHNAME` flag is set in `flags`, then a slash character (`'/'`) in `string` shall be explicitly matched by a slash in `pattern`; it shall not be matched by either the asterisk or question-mark special characters, nor by a bracket expression. If the `FNM_PATHNAME` flag is not set, the slash character shall be treated as an ordinary character.

28601 If `FNM_NOESCAPE` is not set in `flags`, a backslash character (`'\'`) in `pattern` followed by any other character shall match that second character in `string`. In particular, `"\\\"` shall match a backslash in `string`. If `FNM_NOESCAPE` is set, a backslash character shall be treated as an ordinary character.

28605 If `FNM_PERIOD` is set in `flags`, then a leading period (`'.'`) in `string` shall match a period in `pattern`; as described by rule 2 in XCU [Section 2.13.3](#) (on page 2279) where the location of “leading” is indicated by the value of `FNM_PATHNAME`:

- 28608 • If `FNM_PATHNAME` is set, a period is “leading” if it is the first character in `string` or if it immediately follows a slash.
- 28610 • If `FNM_PATHNAME` is not set, a period is “leading” only if it is the first character of `string`.

28612 If `FNM_PERIOD` is not set, then no special restrictions are placed on matching a period.

28613 **RETURN VALUE**

28614 If `string` matches the pattern specified by `pattern`, then `fnmatch()` shall return 0. If there is no match, `fnmatch()` shall return `FNM_NOMATCH`, which is defined in `<fnmatch.h>`. If an error occurs, `fnmatch()` shall return another non-zero value.

28617 **ERRORS**

28618 No errors are defined.

28619 **EXAMPLES**

28620 None.

28621 **APPLICATION USAGE**

28622 The `fnmatch()` function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The `find` utility is an example of this. It can also be used by the `pax` utility to process its `pattern` operands, or by applications that need to match strings in a similar manner.

28626 The name `fnmatch()` is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filenames, rather than pathnames, since it gives no special significance to the slash character. With the `FNM_PATHNAME` flag, `fnmatch()` does match pathnames, but without tilde expansion, parameter expansion, or special treatment for a period at the beginning of a filename.

fnmatch()28631
28632
28633
28634
28635
28636**RATIONALE**

This function replaced the REG_FILENAME flag of *regcomp()* in early proposals of this volume of POSIX.1-200x. It provides virtually the same functionality as the *regcomp()* and *regexexec()* functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was proposed for *regcomp()*, and would have had the opposite effect from FNM_PATHNAME), but with a simpler function and less system overhead.

28637
28638**FUTURE DIRECTIONS**

None.

28639
28640**SEE ALSO**

glob(), Section 2.6

28641

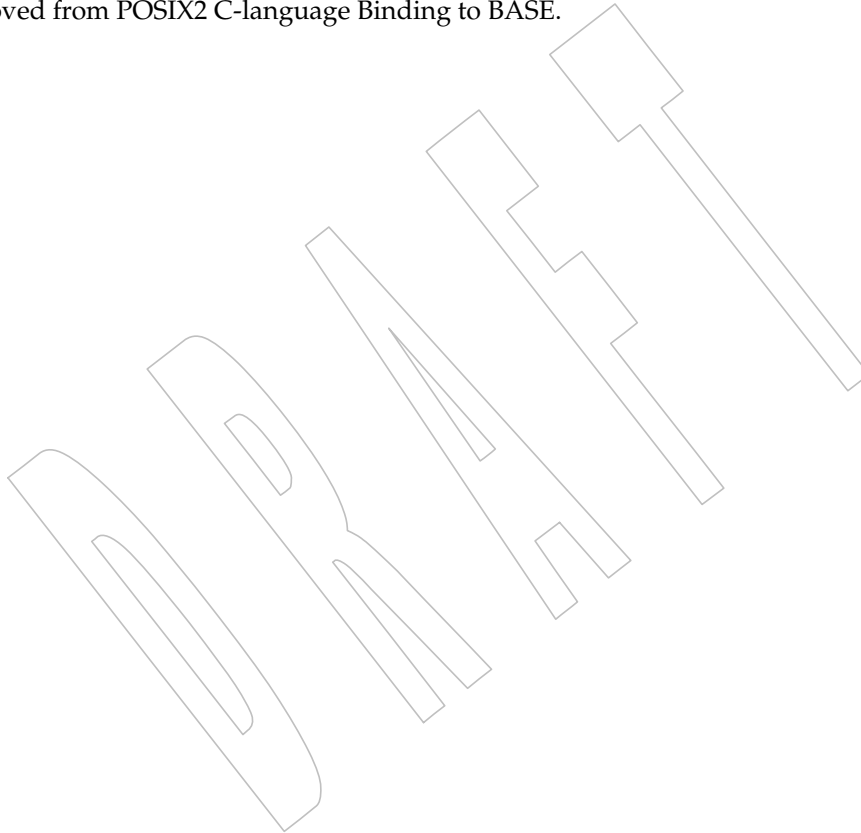
XBD <[fnmatch.h](#)>

28642
28643**CHANGE HISTORY**

First released in Issue 4. Derived from the ISO POSIX-2 standard.

28644
28645**Issue 5**

Moved from POSIX2 C-language Binding to BASE.



28646 **NAME**

28647 fopen — open a stream

28648 **SYNOPSIS**

28649 #include <stdio.h>

28650 FILE *fopen(const char *restrict filename, const char *restrict mode);

28651 **DESCRIPTION**

28652 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 28653 conflict between the requirements described here and the ISO C standard is unintentional. This
 28654 volume of POSIX.1-200x defers to the ISO C standard.

28655 The *fopen()* function shall open the file whose pathname is the string pointed to by *filename*, and
 28656 associates a stream with it.

28657 The *mode* argument points to a string. If the string is one of the following, the file shall be opened
 28658 in the indicated mode. Otherwise, the behavior is undefined.

28659 *r* or *rb* Open file for reading.

28660 *w* or *wb* Truncate to zero length or create file for writing.

28661 *a* or *ab* Append; open or create file for writing at end-of-file.

28662 *r+* or *rb+* or *r+b* Open file for update (reading and writing).

28663 *w+* or *wb+* or *w+b* Truncate to zero length or create file for update.

28664 *a+* or *ab+* or *a+b* Append; open or create file for update, writing at end-of-file.

28665 CX The character 'b' shall have no effect, but is allowed for ISO C standard conformance. Opening
 28666 a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not
 28667 exist or cannot be read.

28668 Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all
 28669 subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening
 28670 calls to *fseek()*.

28671 When a file is opened with update mode ('+' as the second or third character in the *mode*
 28672 argument), both input and output may be performed on the associated stream. However, the
 28673 application shall ensure that output is not directly followed by input without an intervening call
 28674 to *fflush()* or to a file positioning function (*fseek()*, *fsetpos()*, or *rewind()*), and input is not directly
 28675 followed by output without an intervening call to a file positioning function, unless the input
 28676 operation encounters end-of-file.

28677 When opened, a stream is fully buffered if and only if it can be determined not to refer to an
 28678 interactive device. The error and end-of-file indicators for the stream shall be cleared.

28679 CX If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, upon
 28680 successful completion, *fopen()* shall mark for update the last data access, last data modification,
 28681 and last file status change timestamps of the file and the last file status change and last data
 28682 modification timestamps of the parent directory.

28683 If *mode* is *w*, *wb*, *a*, *ab*, *w+*, *wb+*, *w+b*, *a+*, *ab+*, or *a+b*, and the file did not previously exist, the
 28684 *fopen()* function shall create a file as if it called the *creat()* function with a value appropriate for
 28685 the *path* argument interpreted from *filename* and a value of S_IRUSR | S_IWUSR | S_IRGRP |
 28686 S_IWGRP | S_IROTH | S_IWOTH for the *mode* argument.

28687 If *mode* is *w*, *wb*, *w+*, *wb+*, or *w+b*, and the file did previously exist, upon successful completion,
 28688 *fopen()* shall mark for update the last data modification and last file status change timestamps of

fopen()

28689 the file. The *fopen()* function shall allocate a file descriptor as *open()* does.

28690 XSI After a successful call to the *fopen()* function, the orientation of the stream shall be cleared, the

28691 encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an

28692 initial conversion state.

28693 CX The largest value that can be represented correctly in an object of type **off_t** shall be established

28694 as the offset maximum in the open file description.

RETURN VALUE

28695 Upon successful completion, *fopen()* shall return a pointer to the object controlling the stream.

28696 CX Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

28698 The *fopen()* function shall fail if:

28700 CX [EACCES] Search permission is denied on a component of the path prefix, or the file

28701 exists and the permissions specified by *mode* are denied, or the file does not

28702 exist and write permission is denied for the parent directory of the file to be

28703 created.

28704 CX [EINTR] A signal was caught during *fopen()*.

28705 CX [EISDIR] The named file is a directory and *mode* requires write access.

28706 CX [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*

28707 argument.

28708 CX [EMFILE] All file descriptors available to the process are currently open.

28709 CX [EMFILE] {STREAM_MAX} streams are currently open in the calling process. +

28710 CX [ENAMETOOLONG]

28711 The length of the *filename* argument exceeds {PATH_MAX} or a pathname

28712 component is longer than {NAME_MAX}.

28713 CX [ENFILE] The maximum allowable number of files is currently open in the system.

28714 CX [ENOENT] A component of *filename* does not name an existing file or *filename* is an empty

28715 string.

28716 CX [ENOSPC] The directory or file system that would contain the new file cannot be

28717 expanded, the file does not exist, and the file was to be created.

28718 CX [ENOTDIR] A component of the path prefix is not a directory.

28719 CX [ENXIO] The named file is a character special or block special file, and the device

28720 associated with this special file does not exist.

28721 CX [EOVERFLOW] The named file is a regular file and the size of the file cannot be represented

28722 correctly in an object of type **off_t**.

28723 CX [EROFS] The named file resides on a read-only file system and *mode* requires write

28724 access.

28725 The *fopen()* function may fail if:

28726 CX [EINVAL] The value of the *mode* argument is not valid.

28727 CX [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during

28728 resolution of the *path* argument.

28729 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process. -

28730	CX	[ENAMETOOLONG]	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
28731			
28732			
28733	CX	[ENOMEM]	Insufficient storage space is available.
28734	CX	[ETXTBSY]	The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.
28735			

EXAMPLES**Opening a File**

The following example tries to open the file named **file** for reading. The *fopen()* function returns a file pointer that is used in subsequent *fgets()* and *fclose()* calls. If the program cannot open the file, it just ignores it.

```
#include <stdio.h>
...
FILE *fp;
...
void rgrep(const char *file)
{
...
    if ((fp = fopen(file, "r")) == NULL)
        return;
...
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

creat(), *fclose()*, *fdopen()*, *fmemopen()*, *freopen()*, *open_memstream()*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

Large File Summit extensions are added.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [EOVERFLOW] condition is added. This change is to support large files.

28773

- The [ELOOP] mandatory error condition is added.

28774

28775

- The [EINVAL], [EMFILE], [ENAMETOOLONG], [ENOMEM], and [ETXTBSY] optional error conditions are added.

28776

The normative text is updated to avoid use of the term “must” for application requirements.

28777

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

28778

- The prototype for *fopen()* is updated.

28779

28780

- The DESCRIPTION is updated to note that if the argument *mode* points to a string other than those listed, then the behavior is undefined.

28781

28782

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

28783

Issue 7

28784

Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying the file creation mode. +

28785

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error. |

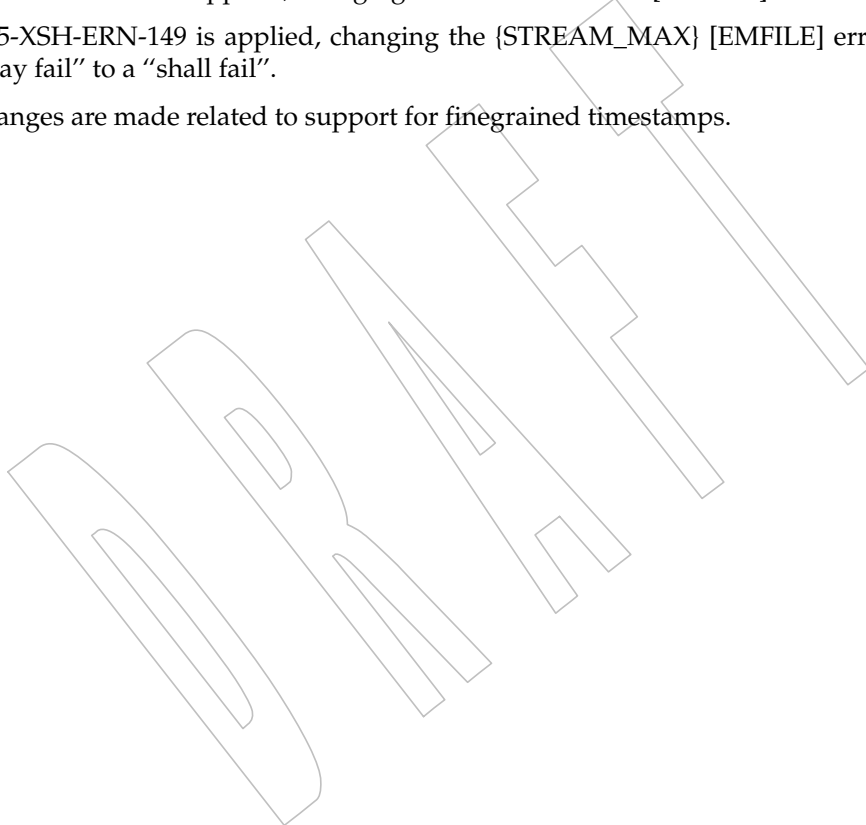
28786

28787

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a “may fail” to a “shall fail”. |

28788

Changes are made related to support for finegrained timestamps.



28789 **NAME**

28790 fork — create a new process

28791 **SYNOPSIS**

28792 #include <unistd.h>

28793 pid_t fork(void);

28794 **DESCRIPTION**28795 The *fork()* function shall create a new process. The new process (child process) shall be an exact
28796 copy of the calling process (parent process) except as detailed below:

- 28797 • The child process shall have a unique process ID.
- 28798 • The child process ID also shall not match any active process group ID.
- 28799 • The child process shall have a different parent process ID, which shall be the process ID of
28800 the calling process.
- 28801 • The child process shall have its own copy of the parent's file descriptors. Each of the
28802 child's file descriptors shall refer to the same open file description with the corresponding
28803 file descriptor of the parent.
- 28804 • The child process shall have its own copy of the parent's open directory streams. Each
28805 open directory stream in the child process may share directory stream positioning with the
28806 corresponding directory stream of the parent.
- 28807 • The child process shall have its own copy of the parent's message catalog descriptors.
- 28808 • The child process values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to
28809 0.
- 28810 • The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be
28811 canceled; see *alarm()*.
- 28812 XSI • All *semadj* values shall be cleared.
- 28813 • File locks set by the parent process shall not be inherited by the child process.
- 28814 • The set of signals pending for the child process shall be initialized to the empty set.
- 28815 XSI • Interval timers shall be reset in the child process.
- 28816 • Any semaphores that are open in the parent process shall also be open in the child process.
- 28817 ML • The child process shall not inherit any address space memory locks established by the
28818 parent process via calls to *mlockall()* or *mlock()*.
- 28819 • Memory mappings created in the parent shall be retained in the child process.
28820 MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE
28821 mappings in the child, and any modifications to the data in these mappings made by the
28822 parent prior to calling *fork()* shall be visible to the child. Any modifications to the data in
28823 MAP_PRIVATE mappings made by the parent after *fork()* returns shall be visible only to
28824 the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall
28825 be visible only to the child.
- 28826 PS • For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit
28827 the policy and priority settings of the parent process during a *fork()* function. For other
28828 scheduling policies, the policy and priority settings on *fork()* are implementation-defined.

fork()

- 28829
- Per-process timers created by the parent shall not be inherited by the child process.
- 28830 MSG
- The child process shall have its own copy of the message queue descriptors of the parent. Each of the message descriptors of the child shall refer to the same open message queue description as the corresponding message descriptor of the parent.
- 28831
- 28832
- 28833
- No asynchronous input or asynchronous output operations shall be inherited by the child process. Any use of asynchronous control blocks created by the parent produces undefined behavior.
- 28834
- 28835
- 28836
- A process shall be created with a single thread. If a multi-threaded process calls *fork()*, the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the *exec* functions is called. Fork handlers may be established by means of the *pthread_atfork()* function in order to maintain application invariants across *fork()* calls.
- 28837
- 28838
- 28839
- 28840
- 28841
- 28842
- 28843
- 28844
- When the application calls *fork()* from a signal handler and any of the fork handlers registered by *pthread_atfork()* calls a function that is not asynch-signal-safe, the behavior is undefined.
- 28845 OB TRC TRI
- If the Trace option and the Trace Inherit option are both supported:
- 28846
- 28847
- 28848
- 28849
- 28850
- 28851
- 28852
- If the calling process was being traced in a trace stream that had its inheritance policy set to `POSIX_TRACE_INHERITED`, the child process shall be traced into that trace stream, and the child process shall inherit the parent's mapping of trace event names to trace event type identifiers. If the trace stream in which the calling process was being traced had its inheritance policy set to `POSIX_TRACE_CLOSE_FOR_CHILD`, the child process shall not be traced into that trace stream. The inheritance policy is set by a call to the *posix_trace_attr_setinherited()* function.
- 28853 OB TRC
- If the Trace option is supported, but the Trace Inherit option is not supported:
- 28854
- The child process shall not be traced into any of the trace streams of its parent process.
- 28855 OB TRC
- If the Trace option is supported, the child process of a trace controller process shall not
- 28856
- < control the trace streams controlled by its parent process.
- 28857 CPT
- The initial value of the CPU-time clock of the child process shall be set to zero.
- 28858 TCT
- The initial value of the CPU-time clock of the single thread of the child process shall be set
- 28859
- to zero.

28860 All other process characteristics defined by POSIX.1-200x shall be the same in the parent and

28861 child processes. The inheritance of process characteristics not defined by POSIX.1-200x is

28862 unspecified by POSIX.1-200x.

28863 After *fork()*, both the parent and the child processes shall be capable of executing independently

28864 before either one terminates.

RETURN VALUE

28865

28866 Upon successful completion, *fork()* shall return 0 to the child process and shall return the

28867 process ID of the child process to the parent process. Both processes shall continue to execute

28868 from the *fork()* function. Otherwise, `-1` shall be returned to the parent process, no child process

28869 shall be created, and *errno* shall be set to indicate the error.

ERRORS

28870 The *fork()* function shall fail if:

- 28871
- 28872 [EAGAIN] The system lacked the necessary resources to create another process, or the
- 28873 system-imposed limit on the total number of processes under execution
- 28874 system-wide or by a single user {CHILD_MAX} would be exceeded.

28875 The *fork()* function may fail if:

28876 [ENOMEM] Insufficient storage space is available.

28877 EXAMPLES

28878 None.

28879 APPLICATION USAGE

28880 None.

28881 RATIONALE

28882 Many historical implementations have timing windows where a signal sent to a process group
 28883 (for example, an interactive SIGINT) just prior to or during execution of *fork()* is delivered to the
 28884 parent following the *fork()* but not to the child because the *fork()* code clears the child's set of
 28885 pending signals. This volume of POSIX.1-200x does not require, or even permit, this behavior.
 28886 However, it is pragmatic to expect that problems of this nature may continue to exist in
 28887 implementations that appear to conform to this volume of POSIX.1-200x and pass available
 28888 verification suites. This behavior is only a consequence of the implementation failing to make
 28889 the interval between signal generation and delivery totally invisible. From the application's
 28890 perspective, a *fork()* call should appear atomic. A signal that is generated prior to the *fork()*
 28891 should be delivered prior to the *fork()*. A signal sent to the process group after the *fork()* should
 28892 be delivered to both parent and child. The implementation may actually initialize internal data
 28893 structures corresponding to the child's set of pending signals to include signals sent to the
 28894 process group during the *fork()*. Since the *fork()* call can be considered as atomic from the
 28895 application's perspective, the set would be initialized as empty and such signals would have
 28896 arrived after the *fork()*; see also `<signal.h>`.

28897 One approach that has been suggested to address the problem of signal inheritance across *fork()*
 28898 is to add an [EINTR] error, which would be returned when a signal is detected during the call.
 28899 While this is preferable to losing signals, it was not considered an optimal solution. Although it
 28900 is not recommended for this purpose, such an error would be an allowable extension for an
 28901 implementation.

28902 The [ENOMEM] error value is reserved for those implementations that detect and distinguish
 28903 such a condition. This condition occurs when an implementation detects that there is not enough
 28904 memory to create the process. This is intended to be returned when [EAGAIN] is inappropriate
 28905 because there can never be enough memory (either primary or secondary storage) to perform
 28906 the operation. Since *fork()* duplicates an existing process, this must be a condition where there is
 28907 sufficient memory for one such process, but not for two. Many historical implementations
 28908 actually return [ENOMEM] due to temporary lack of memory, a case that is not generally
 28909 distinct from [EAGAIN] from the perspective of a conforming application.

28910 Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it
 28911 and it should be reserved for the error condition specified there. The condition is not applicable
 28912 on many implementations.

28913 IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork()*.
 28914 A system that single-threads processes was clearly not intended and is considered an
 28915 unacceptable "toy implementation" of this volume of POSIX.1-200x. The only objection
 28916 anticipated to the phrase "executing independently" is testability, but this assertion should be
 28917 testable. Such tests require that both the parent and child can block on a detectable action of the
 28918 other, such as a write to a pipe or a signal. An interactive exchange of such actions should be
 28919 possible for the system to conform to the intent of this volume of POSIX.1-200x.

28920 The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it
 28921 occurs or not is not in any practical sense under the control of the application because the
 28922 condition is usually a consequence of the user's use of the system, not of the application's code.
 28923 Thus, no application can or should rely upon its occurrence under any circumstances, nor
 28924 should the exact semantics of what concept of "user" is used be of concern to the application

28925 developer. Validation writers should be cognizant of this limitation.

28926 There are two reasons why POSIX programmers call *fork()*. One reason is to create a new thread
28927 of control within the same program (which was originally only possible in POSIX by creating a
28928 new process); the other is to create a new process running a different program. In the latter case,
28929 the call to *fork()* is soon followed by a call to one of the *exec* functions.

28930 The general problem with making *fork()* work in a multi-threaded world is what to do with all
28931 of the threads. There are two alternatives. One is to copy all of the threads into the new process.
28932 This causes the programmer or implementation to deal with threads that are suspended on
28933 system calls or that might be about to execute system calls that should not be executed in the
28934 new process. The other alternative is to copy only the thread that calls *fork()*. This creates the
28935 difficulty that the state of process-local resources is usually held in process memory. If a thread
28936 that is not calling *fork()* holds a resource, that resource is never released in the child process
28937 because the thread whose job it is to release the resource does not exist in the child process.

28938 When a programmer is writing a multi-threaded program, the first described use of *fork()*,
28939 creating new threads in the same program, is provided by the *pthread_create()* function. The
28940 *fork()* function is thus used only to run new programs, and the effects of calling functions that
28941 require certain resources between the call to *fork()* and the call to an *exec* function are undefined.

28942 The addition of the *forkall()* function to the standard was considered and rejected. The *forkall()*
28943 function lets all the threads in the parent be duplicated in the child. This essentially duplicates
28944 the state of the parent in the child. This allows threads in the child to continue processing and
28945 allows locks and the state to be preserved without explicit *pthread_atfork()* code. The calling
28946 process has to ensure that the threads processing state that is shared between the parent and
28947 child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall()*. For
28948 example, if a thread is reading a file descriptor in the parent when *forkall()* is called, then two
28949 threads (one in the parent and one in the child) are reading the file descriptor after the *forkall()*.
28950 If this is not desired behavior, the parent process has to synchronize with such threads before
28951 calling *forkall()*.

28952 While the *fork()* function is async-signal-safe, there is no way for an implementation to
28953 determine whether the fork handlers established by *pthread_atfork()* are async-signal-safe. The
28954 fork handlers may attempt to execute portions of the implementation that are not async-signal-
28955 safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore
28956 undefined for the fork handlers to execute functions that are not async-signal-safe when *fork()*
28957 is called from a signal handler.

28958 When *forkall()* is called, threads, other than the calling thread, that are in functions that can
28959 return with an [EINTR] error may have those functions return [EINTR] if the implementation
28960 cannot ensure that the function behaves correctly in the parent and child. In particular,
28961 *pthread_cond_wait()* and *pthread_cond_timedwait()* need to return in order to ensure that the
28962 condition has not changed. These functions can be awakened by a spurious condition wakeup
28963 rather than returning [EINTR].

28964 FUTURE DIRECTIONS

28965 None.

28966 SEE ALSO

28967 [alarm\(\)](#), [exec](#), [fcntl\(\)](#), [posix_trace_attr_getinherited\(\)](#), [posix_trace_eventid_equal\(\)](#), [pthread_atfork\(\)](#),
28968 [semop\(\)](#), [signal\(\)](#), [times\(\)](#)

28969 XBD Section 4.11 (on page 98), [<sys/types.h>](#), [<unistd.h>](#)

28970
28971
28972
28973
28974
28975
28976
28977
28978
28979
28980
28981
28982
28983
28984
28985
28986
28987
28988
28989
28990
28991
28992
28993

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is changed for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of `fork()` on a pending alarm call in the child process is clarified.

The description of CPU-time clock semantics is added for alignment with IEEE Std 1003.1d-1999.

The description of tracing semantics is added for alignment with IEEE Std 1003.1q-2000.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/17 is applied, adding text to the DESCRIPTION and RATIONALE relating to fork handlers registered by the `pthread_atfork()` function and async-signal safety.

Issue 7

Austin Group Interpretation 1003.1-2001 #080 is applied, clarifying the status of asynchronous input and asynchronous output operations and asynchronous control lists in the DESCRIPTION.

Functionality relating to the Asynchronous Input and Output, Memory Mapped Files, Timers, and Threads options is moved to the Base.

Functionality relating to message catalog descriptors is moved from the XSI option to the Base.

28994 **NAME**
 28995 fpathconf, pathconf — get configurable pathname variables

28996 **SYNOPSIS**
 28997 #include <unistd.h>
 28998 long fpathconf(int *fildes*, int *name*);
 28999 long pathconf(const char **path*, int *name*);

29000 **DESCRIPTION**
 29001 The *fpathconf()* and *pathconf()* functions shall determine the current value of a configurable limit
 29002 or option (*variable*) that is associated with a file or directory.

29003 For *pathconf()*, the *path* argument points to the pathname of a file or directory.

29004 For *fpathconf()*, the *fildes* argument is an open file descriptor.

29005 The *name* argument represents the variable to be queried relative to that file or directory.
 29006 Implementations shall support all of the variables listed in the following table and may support
 29007 others. The variables in the following table come from <limits.h> or <unistd.h> and the
 29008 symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*.

Variable	Value of <i>name</i>	Requirements
{FILESIZEBITS}	_PC_FILESIZEBITS	3,4
{LINK_MAX}	_PC_LINK_MAX	1
{MAX_CANON}	_PC_MAX_CANON	2
{MAX_INPUT}	_PC_MAX_INPUT	2
{NAME_MAX}	_PC_NAME_MAX	3,4
{PATH_MAX}	_PC_PATH_MAX	4,5
{PIPE_BUF}	_PC_PIPE_BUF	6
{POSIX2_SYMLINKS}	_PC_2_SYMLINKS	4
{POSIX_ALLOC_SIZE_MIN}	_PC_ALLOC_SIZE_MIN	10
{POSIX_REC_INCR_XFER_SIZE}	_PC_REC_INCR_XFER_SIZE	10
{POSIX_REC_MAX_XFER_SIZE}	_PC_REC_MAX_XFER_SIZE	10
{POSIX_REC_MIN_XFER_SIZE}	_PC_REC_MIN_XFER_SIZE	10
{POSIX_REC_XFER_ALIGN}	_PC_REC_XFER_ALIGN	10
{SYMLINK_MAX}	_PC_SYMLINK_MAX	4,9
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8
_POSIX_TIMESTAMP_RESOLUTION	_PC_TIMESTAMP_RESOLUTION	1

29031 Requirements

- 29032 1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
- 29033 2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an
 29034 implementation supports an association of the variable name with the specified file.
- 29035 3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the
 29036 directory.

- 29037 4. If *path* or *filde*s does not refer to a directory, it is unspecified whether an implementation
29038 supports an association of the variable name with the specified file.
- 29039 5. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of a
29040 relative pathname when the specified directory is the working directory.
- 29041 6. If *path* refers to a FIFO, or *filde*s refers to a pipe or FIFO, the value returned shall apply to
29042 the referenced object. If *path* or *filde*s refers to a directory, the value returned shall apply to
29043 any FIFO that exists or can be created within the directory. If *path* or *filde*s refers to any
29044 other type of file, it is unspecified whether an implementation supports an association of
29045 the variable name with the specified file.
- 29046 7. If *path* or *filde*s refers to a directory, the value returned shall apply to any files, other than
29047 directories, that exist or can be created within the directory.
- 29048 8. If *path* or *filde*s refers to a directory, it is unspecified whether an implementation supports
29049 an association of the variable name with the specified file.
- 29050 9. If *path* or *filde*s refers to a directory, the value returned shall be the maximum length of the
29051 string that a symbolic link in that directory can contain.
- 29052 10. If *path* or *filde*s does not refer to a regular file, it is unspecified whether an
29053 implementation supports an association of the variable name with the specified file. If an
29054 implementation supports such an association for other than a regular file, the value
29055 returned is unspecified.

RETURN VALUE

29056 If *name* is an invalid value, both *pathconf()* and *fpathconf()* shall return -1 and set *errno* to
29057 indicate the error.
29058

29059 If the variable corresponding to *name* has no limit for the *path* or file descriptor, both *pathconf()*
29060 and *fpathconf()* shall return -1 without changing *errno*. If the implementation needs to use *path*
29061 to determine the value of *name* and the implementation does not support the association of *name*
29062 with the file specified by *path*, or if the process did not have appropriate privileges to query the
29063 file specified by *path*, or *path* does not exist, *pathconf()* shall return -1 and set *errno* to indicate the
29064 error.

29065 If the implementation needs to use *filde*s to determine the value of *name* and the implementation
29066 does not support the association of *name* with the file specified by *filde*s, or if *filde*s is an invalid
29067 file descriptor, *fpathconf()* shall return -1 and set *errno* to indicate the error.

29068 Otherwise, *pathconf()* or *fpathconf()* shall return the current variable value for the file or
29069 directory without changing *errno*. The value returned shall not be more restrictive than the
29070 corresponding value available to the application when it was compiled with the
29071 implementation's `<limits.h>` or `<unistd.h>`.

29072 If the variable corresponding to *name* is dependent on an unsupported option, the results are
29073 unspecified.

ERRORS

29074 The *pathconf()* function shall fail if:
29075

29076 [EINVAL] The value of *name* is not valid.

29077 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
29078 argument.

29079 [E_OVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is +
29080 larger than `{LONG_MAX}`.

29081 The *pathconf()* function may fail if:

- 29082 [EACCES] Search permission is denied for a component of the path prefix.
- 29083 [EINVAL] The implementation does not support an association of the variable *name* with
29084 the specified file.
- 29085 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
29086 resolution of the *path* argument.
- 29087 [ENAMETOOLONG]
29088 The length of the *path* argument exceeds {PATH_MAX} or a pathname
29089 component is longer than {NAME_MAX}.
- 29090 [ENAMETOOLONG]
29091 As a result of encountering a symbolic link in resolution of the *path* argument,
29092 the length of the substituted pathname string exceeded {PATH_MAX}.
- 29093 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 29094 [ENOTDIR] A component of the path prefix is not a directory.
- 29095 The *fpathconf()* function shall fail if:
- 29096 [EINVAL] The value of *name* is not valid.
- 29097 [EOVERFLOW] The value of *name* is `_PC_TIMESTAMP_RESOLUTION` and the resolution is +
29098 larger than {LONG_MAX}.
- 29099 The *fpathconf()* function may fail if:
- 29100 [EBADF] The *fildev* argument is not a valid file descriptor.
- 29101 [EINVAL] The implementation does not support an association of the variable *name* with
29102 the specified file.

EXAMPLES

29103 None.
29104

APPLICATION USAGE

29105 Application developers should check whether an option, such as `_POSIX_ADVISORY_INFO`, is
29106 supported prior to obtaining and using values for related variables such as
29107 {`POSIX_ALLOC_SIZE_MIN`}.
29108

RATIONALE

29109 The *pathconf()* function was proposed immediately after the *sysconf()* function when it was
29110 realized that some configurable values may differ across file system, directory, or device
29111 boundaries.
29112

29113 For example, {`NAME_MAX`} frequently changes between System V and BSD-based file systems;
29114 System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file
29115 systems, an application would be forced to limit all pathname components to 14 bytes, as this
29116 would be the value specified in `<limits.h>` on such a system.

29117 Therefore, various useful values can be queried on any pathname or file descriptor, assuming
29118 that the appropriate permissions are in place.

29119 The value returned for the variable {`PATH_MAX`} indicates the longest relative pathname that
29120 could be given if the specified directory is the current working directory of the process. A
29121 process may not always be able to generate a name that long and use it if a subdirectory in the
29122 pathname crosses into a more restrictive file system.

29123 The value returned for the variable `_POSIX_CHOWN_RESTRICTED` also applies to directories
29124 that do not have file systems mounted on them. The value may change when crossing a mount
29125 point, so applications that need to know should check for each directory. (An even easier check
29126 is to try the *chown()* function and look for an error in case it happens.)

29127 Unlike the values returned by *sysconf()*, the pathname-oriented variables are potentially more
 29128 volatile and are not guaranteed to remain constant throughout the lifetime of the process. For
 29129 example, in between two calls to *pathconf()*, the file system in question may have been
 29130 unmounted and remounted with different characteristics.

29131 Also note that most of the errors are optional. If one of the variables always has the same value
 29132 on an implementation, the implementation need not look at *path* or *fildev* to return that value and
 29133 is, therefore, not required to detect any of the errors except the meaning of [EINVAL] that
 29134 indicates that the value of *name* is not valid for that variable.

29135 If the value of any of the limits is unspecified (logically infinite), they will not be defined in
 29136 <limits.h> and the *pathconf()* and *fpathconf()* functions return -1 without changing *errno*. This
 29137 can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set
 29138 to [EINVAL] in this case.

29139 Since -1 is a valid return value for the *pathconf()* and *fpathconf()* functions, applications should
 29140 set *errno* to zero before calling them and check *errno* only if the return value is -1.

29141 For the case of {SYMLINK_MAX}, since both *pathconf()* and *open()* follow symbolic links, there
 29142 is no way that *path* or *fildev* could refer to a symbolic link.

29143 It was the intention of IEEE Std 1003.1d-1999 that the following variables:

```
29144     {POSIX_ALLOC_SIZE_MIN}
29145     {POSIX_REC_INCR_XFER_SIZE}
29146     {POSIX_REC_MAX_XFER_SIZE}
29147     {POSIX_REC_MIN_XFER_SIZE}
29148     {POSIX_REC_XFER_ALIGN}
```

29149 only applied to regular files, but Note 10 also permits implementation of the advisory semantics
 29150 on other file types unique to an implementation (for example, a character special device).

29151 The [EOVERFLOW] error for _PC_TIMESTAMP_RESOLUTION cannot occur on POSIX- +
 29152 compliant file systems because POSIX requires a timestamp resolution no larger than one +
 29153 second. Even on 32-bit systems, this can be represented without overflow.

29154 FUTURE DIRECTIONS

29155 None.

29156 SEE ALSO

29157 *chown*, *confstr()*, *sysconf()*

29158 XBD <limits.h>, <unistd.h>

29159 XCU *getconf*

29160 CHANGE HISTORY

29161 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

29162 Issue 5

29163 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

29164 Large File Summit extensions are added.

29165 Issue 6

29166 The following new requirements on POSIX implementations derive from alignment with the
 29167 Single UNIX Specification:

- 29168 • The DESCRIPTION is updated to include {FILESIZEBITS}.
- 29169 • The [ELOOP] mandatory error condition is added.

29170

- A second [ENAMETOOLONG] is added as an optional error condition.

29171

The following changes were made to align with the IEEE P1003.1a draft standard:

29172

- The `_PC_SYMLINK_MAX` entry is added to the table in the DESCRIPTION.

29173

The following `fpathconf()` variables and their associated names are added for alignment with IEEE Std 1003.1d-1999:

29174

29175

```
{POSIX_ALLOC_SIZE_MIN}
```

29176

```
{POSIX_REC_INCR_XFER_SIZE}
```

29177

```
{POSIX_REC_MAX_XFER_SIZE}
```

29178

```
{POSIX_REC_MIN_XFER_SIZE}
```

29179

```
{POSIX_REC_XFER_ALIGN}
```

29180

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/18 is applied, changing the fourth paragraph of the DESCRIPTION and removing shading and margin markers from the table. This change is needed since implementations are required to support all of these symbols.

29181

29182

29183

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/34 is applied, adding the table entry for `POSIX2_SYMLINKS` in the DESCRIPTION.

29184

29185

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/35 is applied, updating the DESCRIPTION and RATIONALE sections to clarify behavior for the following variables:

29186

29187

```
{POSIX_ALLOC_SIZE_MIN}
```

29188

```
{POSIX_REC_INCR_XFER_SIZE}
```

29189

```
{POSIX_REC_MAX_XFER_SIZE}
```

29190

```
{POSIX_REC_MIN_XFER_SIZE}
```

29191

```
{POSIX_REC_XFER_ALIGN}
```

29192

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/36 is applied, updating the RETURN VALUE and APPLICATION USAGE sections to state that the results are unspecified if a variable is dependent on an unsupported option, and advising application developers to check for supported options prior to obtaining and using such values.

29193

29194

29195

29196

Issue 7

29197

Changes are made related to support for finegrained timestamps.

29198 **NAME**
 29199 fpclassify — classify real floating type

29200 **SYNOPSIS**
 29201 #include <math.h>
 29202 int fpclassify(real-floating x);

29203 **DESCRIPTION**
 29204 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 29205 conflict between the requirements described here and the ISO C standard is unintentional. This
 29206 volume of POSIX.1-200x defers to the ISO C standard.

29207 The *fpclassify()* macro shall classify its argument value as NaN, infinite, normal, subnormal,
 29208 zero, or into another implementation-defined category. First, an argument represented in a
 29209 format wider than its semantic type is converted to its semantic type. Then classification is based
 29210 on the type of the argument.

29211 **RETURN VALUE**
 29212 The *fpclassify()* macro shall return the value of the number classification macro appropriate to
 29213 the value of its argument.

29214 **ERRORS**
 29215 No errors are defined.

29216 **EXAMPLES**
 29217 None.

29218 **APPLICATION USAGE**
 29219 None.

29220 **RATIONALE**
 29221 None.

29222 **FUTURE DIRECTIONS**
 29223 None.

29224 **SEE ALSO**
 29225 *isfinite()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

29226 XBD <math.h>

29227 **CHANGE HISTORY**
 29228 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

29229 **NAME**29230 `dprintf, fprintf, printf, snprintf, sprintf` — print formatted output29231 **SYNOPSIS**29232 `#include <stdio.h>`

```

29233 CX int dprintf(int fildes, const char *format, ...);
29234 int fprintf(FILE *restrict stream, const char *restrict format, ...);
29235 int printf(const char *restrict format, ...);
29236 int snprintf(char *restrict s, size_t n,
29237 const char *restrict format, ...);
29238 int sprintf(char *restrict s, const char *restrict format, ...);

```

29239 **DESCRIPTION**

29240 CX Excluding `dprintf()`: The functionality described on this reference page is aligned with the ISO C
 29241 standard. Any conflict between the requirements described here and the ISO C standard is
 29242 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

29243 The `fprintf()` function shall place output on the named output *stream*. The `printf()` function shall
 29244 place output on the standard output stream *stdout*. The `sprintf()` function shall place output
 29245 followed by the null byte, `'\0'`, in consecutive bytes starting at **s*; it is the user's responsibility
 29246 to ensure that enough space is available.

29247 CX The `dprintf()` function shall be equivalent to the `fprintf()` function, except that `dprintf()` shall
 29248 write output to the file associated with the file descriptor specified by the *fildes* argument rather
 29249 than place output on a stream.

29250 The `snprintf()` function shall be equivalent to `sprintf()`, with the addition of the *n* argument
 29251 which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s*
 29252 may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of
 29253 being written to the array, and a null byte is written at the end of the bytes actually written into
 29254 the array.

29255 If copying takes place between objects that overlap as a result of a call to `sprintf()` or `snprintf()`,
 29256 the results are undefined.

29257 Each of these functions converts, formats, and prints its arguments under control of the *format*.
 29258 The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is
 29259 composed of zero or more directives: *ordinary characters*, which are simply copied to the output
 29260 stream, and *conversion specifications*, each of which shall result in the fetching of zero or more
 29261 arguments. The results are undefined if there are insufficient arguments for the *format*. If the
 29262 *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are
 29263 otherwise ignored.

29264 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
 29265 to the next unused argument. In this case, the conversion specifier character `%` (see below) is
 29266 replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range `[1,{NL_ARGMAX}]`,
 29267 giving the position of the argument in the argument list. This feature provides for the definition
 29268 of format strings that select arguments in an order appropriate to specific languages (see the
 29269 EXAMPLES section).

29270 The *format* can contain either numbered argument conversion specifications (that is, `"%n$"` and
 29271 `"*m$"`), or unnumbered argument conversion specifications (that is, `%` and `*`), but not both. The
 29272 only exception to this is that `%%` can be mixed with the `"%n$"` form. The results of mixing
 29273 numbered and unnumbered argument specifications in a *format* string are undefined. When
 29274 numbered argument specifications are used, specifying the *N*th argument requires that all the
 29275 leading arguments, from the first to the $(N-1)$ th, are specified in the format string.

- 29276 In format strings containing the "%n\$" form of conversion specification, numbered arguments
29277 in the argument list can be referenced from the format string as many times as required.
- 29278 In format strings containing the % form of conversion specification, each conversion specification
29279 uses the first unused argument in the argument list.
- 29280 CX All forms of the *fprintf()* functions allow for the insertion of a language-dependent radix
29281 character in the output string. The radix character is defined in the process' locale (category
29282 *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the
29283 radix character shall default to a period ('.').
- 29284 CX Each conversion specification is introduced by the '%' character or by the character sequence
29285 "%n\$", after which the following appear in sequence:
- 29286 • Zero or more *flags* (in any order), which modify the meaning of the conversion
29287 specification.
 - 29288 • An optional minimum *field width*. If the converted value has fewer bytes than the field
29289 width, it shall be padded with spaces by default on the left; it shall be padded on the right
29290 if the left-adjustment flag ('-'), described below, is given to the field width. The field
29291 width takes the form of an asterisk ('*'), described below, or a decimal integer.
 - 29292 • An optional *precision* that gives the minimum number of digits to appear for the d, i, o, u,
29293 x, and X conversion specifiers; the number of digits to appear after the radix character for
29294 the a, A, e, E, f, and F conversion specifiers; the maximum number of significant digits for
29295 the g and G conversion specifiers; or the maximum number of bytes to be printed from a
29296 XSI string in the s and S conversion specifiers. The precision takes the form of a period ('.')
29297 followed either by an asterisk ('*'), described below, or an optional decimal digit string,
29298 where a null digit string is treated as zero. If a precision appears with any other conversion
29299 specifier, the behavior is undefined.
 - 29300 • An optional length modifier that specifies the size of the argument.
 - 29301 • A *conversion specifier* character that indicates the type of conversion to be applied.
- 29302 A field width, or precision, or both, may be indicated by an asterisk ('*'). In this case an
29303 argument of type **int** supplies the field width or precision. Applications shall ensure that
29304 arguments specifying field width, or precision, or both appear in that order before the argument,
29305 if any, to be converted. A negative field width is taken as a '-' flag followed by a positive field
29306 CX width. A negative precision is taken as if the precision were omitted. In *format* strings
29307 containing the "%n\$" form of a conversion specification, a field width or precision may be
29308 indicated by the sequence "*m\$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}]
29309 giving the position in the argument list (after the *format* argument) of an integer argument
29310 containing the field width or precision, for example:
- ```
29311 printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 29312 The flag characters and their meanings are:
- 29313 CX ' The integer portion of the result of a decimal conversion (%i, %d, %u, %f, %F, %g, or %G)  
29314 shall be formatted with thousands' grouping characters. For other conversions the  
29315 behavior is undefined. The non-monetary grouping character is used.
- 29316 - The result of the conversion shall be left-justified within the field. The conversion is  
29317 right-justified if this flag is not specified.
- 29318 + The result of a signed conversion shall always begin with a sign ('+' or '-'). The  
29319 conversion shall begin with a sign only when a negative value is converted if this flag is  
29320 not specified.

|       |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------|----|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 29321 |    | <space>      | If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and '+' flags both appear, the <space> flag shall be ignored.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 29322 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29323 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29324 |    | #            | Specifies that the value is to be converted to an alternative form. For o conversion, it increases the precision (if necessary) to force the first digit of the result to be zero. For x or X conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For a, A, e, E, f, F, g, and G conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For g and G conversion specifiers, trailing zeros shall <i>not</i> be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined. |
| 29325 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29326 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29327 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29328 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29329 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29330 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29331 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29332 |    | 0            | For d, i, o, u, x, X, a, A, e, E, f, F, g, and G conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width; no space padding is performed. If the '0' and '-' flags both appear, the '0' flag is ignored. For d, i, o, u, x, and X conversion specifiers, if a precision is specified, the '0' flag shall be ignored. If the '0' and ' ' flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behavior is undefined.                                                                                                                                                                                                |
| 29333 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29334 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29335 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29336 | CX |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29337 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29338 |    |              | The length modifiers and their meanings are:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 29339 |    | hh           | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>signed char</b> or <b>unsigned char</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>signed char</b> or <b>unsigned char</b> before printing); or that a following n conversion specifier applies to a pointer to a <b>signed char</b> argument.                                                                                                                                                                                                                                                                                                              |
| 29340 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29341 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29342 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29343 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29344 |    | h            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>short</b> or <b>unsigned short</b> argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to <b>short</b> or <b>unsigned short</b> before printing); or that a following n conversion specifier applies to a pointer to a <b>short</b> argument.                                                                                                                                                                                                                                                                                                                              |
| 29345 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29346 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29347 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29348 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29349 |    | l (ell)      | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long</b> or <b>unsigned long</b> argument; that a following n conversion specifier applies to a pointer to a <b>long</b> argument; that a following c conversion specifier applies to a <b>wint_t</b> argument; that a following s conversion specifier applies to a pointer to a <b>wchar_t</b> argument; or has no effect on a following a, A, e, E, f, F, g, or G conversion specifier.                                                                                                                                                                                                                                                  |
| 29350 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29351 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29352 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29353 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29354 |    | ll (ell-ell) | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>long long</b> or <b>unsigned long long</b> argument; or that a following n conversion specifier applies to a pointer to a <b>long long</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 29355 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29356 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29357 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29358 |    | j            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to an <b>intmax_t</b> or <b>uintmax_t</b> argument; or that a following n conversion specifier applies to a pointer to an <b>intmax_t</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29359 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29360 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29361 |    | z            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>size_t</b> or the corresponding signed integer type argument; or that a following n conversion specifier applies to a pointer to a signed integer type corresponding to a <b>size_t</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 29362 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29363 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29364 |    | t            | Specifies that a following d, i, o, u, x, or X conversion specifier applies to a <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type argument; or that a following n conversion specifier applies to a pointer to a <b>ptrdiff_t</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 29365 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29366 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 29367 |    | L            | Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to a <b>long double</b> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 29368 |    |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

- 29369 If a length modifier appears with any conversion specifier other than as specified above, the  
29370 behavior is undefined.
- 29371 The conversion specifiers and their meanings are:
- 29372 **d, i** The **int** argument shall be converted to a signed decimal in the style "`[-]dddd`". The  
29373 precision specifies the minimum number of digits to appear; if the value being  
29374 converted can be represented in fewer digits, it shall be expanded with leading zeros.  
29375 The default precision is 1. The result of converting zero with an explicit precision of  
29376 zero shall be no characters.
- 29377 **o** The **unsigned** argument shall be converted to unsigned octal format in the style  
29378 "`dddd`". The precision specifies the minimum number of digits to appear; if the value  
29379 being converted can be represented in fewer digits, it shall be expanded with leading  
29380 zeros. The default precision is 1. The result of converting zero with an explicit precision  
29381 of zero shall be no characters.
- 29382 **u** The **unsigned** argument shall be converted to unsigned decimal format in the style  
29383 "`dddd`". The precision specifies the minimum number of digits to appear; if the value  
29384 being converted can be represented in fewer digits, it shall be expanded with leading  
29385 zeros. The default precision is 1. The result of converting zero with an explicit precision  
29386 of zero shall be no characters.
- 29387 **x** The **unsigned** argument shall be converted to unsigned hexadecimal format in the style  
29388 "`dddd`"; the letters "abcdef" are used. The precision specifies the minimum number  
29389 of digits to appear; if the value being converted can be represented in fewer digits, it  
29390 shall be expanded with leading zeros. The default precision is 1. The result of  
29391 converting zero with an explicit precision of zero shall be no characters.
- 29392 **X** Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead  
29393 of "abcdef".
- 29394 **f, F** The **double** argument shall be converted to decimal notation in the style  
29395 "`[-]ddd.ddd`", where the number of digits after the radix character is equal to the  
29396 precision specification. If the precision is missing, it shall be taken as 6; if the precision  
29397 is explicitly zero and no '#' flag is present, no radix character shall appear. If a radix  
29398 character appears, at least one digit appears before it. The low-order digit shall be  
29399 rounded in an implementation-defined manner.
- 29400 A **double** argument representing an infinity shall be converted in one of the styles  
29401 "`[-]inf`" or "`[-]infinity`"; which style is implementation-defined. A **double**  
29402 argument representing a NaN shall be converted in one of the styles "`[-]nan(n-char-sequence)`" or "`[-]nan`"; which style, and the meaning of any *n-char-sequence*,  
29403 is implementation-defined. The **F** conversion specifier produces "INF", "INFINITY",  
29404 or "NAN" instead of "inf", "infinity", or "nan", respectively.
- 29406 **e, E** The **double** argument shall be converted in the style "`[-]d.ddde±dd`", where there is  
29407 one digit before the radix character (which is non-zero if the argument is non-zero) and  
29408 the number of digits after it is equal to the precision; if the precision is missing, it shall  
29409 be taken as 6; if the precision is zero and no '#' flag is present, no radix character shall  
29410 appear. The low-order digit shall be rounded in an implementation-defined manner.  
29411 The **E** conversion specifier shall produce a number with 'E' instead of 'e'  
29412 introducing the exponent. The exponent shall always contain at least two digits. If the  
29413 value is zero, the exponent shall be zero.
- 29414 A **double** argument representing an infinity or NaN shall be converted in the style of  
29415 an **f** or **F** conversion specifier.

29416 g, G The **double** argument representing a floating-point number shall be converted in the  
 29417 style `f` or `e` (or in the style `F` or `E` in the case of a `G` conversion specifier), depending on  
 29418 the value converted and the precision. Let `P` equal the precision if non-zero, 6 if the  
 29419 precision is omitted, or 1 if the precision is zero. Then, if a conversion with style `E`  
 29420 would have an exponent of `X`:

- 29421 — If  $P > X \geq -4$ , the conversion shall be with style `f` (or `F`) and precision  $P - (X + 1)$ .
- 29422 — Otherwise, the conversion shall be with style `e` (or `E`) and precision  $P - 1$ .

29423 Finally, unless the `'#'` flag is used, any trailing zeros shall be removed from the  
 29424 fractional portion of the result and the decimal-point character shall be removed if there  
 29425 is no fractional portion remaining.

29426 A **double** argument representing an infinity or NaN shall be converted in the style of  
 29427 an `f` or `F` conversion specifier.

29428 a, A A **double** argument representing a floating-point number shall be converted in the  
 29429 style `"[-]0xh.hhhhp±d"`, where there is one hexadecimal digit (which shall be non-  
 29430 zero if the argument is a normalized floating-point number and is otherwise  
 29431 unspecified) before the decimal-point character and the number of hexadecimal digits  
 29432 after it is equal to the precision; if the precision is missing and `FLT_RADIX` is a power  
 29433 of 2, then the precision shall be sufficient for an exact representation of the value; if the  
 29434 precision is missing and `FLT_RADIX` is not a power of 2, then the precision shall be  
 29435 sufficient to distinguish values of type **double**, except that trailing zeros may be  
 29436 omitted; if the precision is zero and the `'#'` flag is not specified, no decimal-point  
 29437 character shall appear. The letters `"abcdef"` shall be used for a conversion and the  
 29438 letters `"ABCDEF"` for a conversion. The `A` conversion specifier produces a number with  
 29439 `'X'` and `'P'` instead of `'x'` and `'p'`. The exponent shall always contain at least one  
 29440 digit, and only as many more digits as necessary to represent the decimal exponent of  
 29441 2. If the value is zero, the exponent shall be zero.

29442 A **double** argument representing an infinity or NaN shall be converted in the style of  
 29443 an `f` or `F` conversion specifier.

29444 c The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall  
 29445 be written.

29446 If an `l` (ell) qualifier is present, the **wint\_t** argument shall be converted as if by an `ls`  
 29447 conversion specification with no precision and an argument that points to a two-  
 29448 element array of type **wchar\_t**, the first element of which contains the **wint\_t** argument  
 29449 to the `ls` conversion specification and the second element contains a null wide  
 29450 character.

29451 s The argument shall be a pointer to an array of **char**. Bytes from the array shall be  
 29452 written up to (but not including) any terminating null byte. If the precision is specified,  
 29453 no more than that many bytes shall be written. If the precision is not specified or is  
 29454 greater than the size of the array, the application shall ensure that the array contains a  
 29455 null byte.

29456 If an `l` (ell) qualifier is present, the argument shall be a pointer to an array of type  
 29457 **wchar\_t**. Wide characters from the array shall be converted to characters (each as if by  
 29458 a call to the `wcrtomb()` function, with the conversion state described by an **mbstate\_t**  
 29459 object initialized to zero before the first wide character is converted) up to and  
 29460 including a terminating null wide character. The resulting characters shall be written  
 29461 up to (but not including) the terminating null character (byte). If no precision is  
 29462 specified, the application shall ensure that the array contains a null wide character. If a  
 29463 precision is specified, no more than that many characters (bytes) shall be written  
 29464 (including shift sequences, if any), and the array shall contain a null wide character if,

|       |     |   |                                                                                                    |
|-------|-----|---|----------------------------------------------------------------------------------------------------|
| 29465 |     |   | to equal the character sequence length given by the precision, the function would need             |
| 29466 |     |   | to access a wide character one past the end of the array. In no case shall a partial               |
| 29467 |     |   | character be written.                                                                              |
| 29468 | p   |   | The argument shall be a pointer to <b>void</b> . The value of the pointer is converted to a        |
| 29469 |     |   | sequence of printable characters, in an implementation-defined manner.                             |
| 29470 | n   |   | The argument shall be a pointer to an integer into which is written the number of bytes            |
| 29471 |     |   | written to the output so far by this call to one of the <i>fprintf()</i> functions. No argument is |
| 29472 |     |   | converted.                                                                                         |
| 29473 | XSI | C | Equivalent to <code>lc</code> .                                                                    |
| 29474 | XSI | S | Equivalent to <code>ls</code> .                                                                    |
| 29475 | %   |   | Print a ' % ' character; no argument is converted. The complete conversion specification           |
| 29476 |     |   | shall be <code>%%</code> .                                                                         |

29477 If a conversion specification does not match one of the above forms, the behavior is undefined. If  
 29478 any argument is not the correct type for the corresponding conversion specification, the  
 29479 behavior is undefined.

29480 In no case shall a nonexistent or small field width cause truncation of a field; if the result of a  
 29481 conversion is wider than the field width, the field shall be expanded to contain the conversion  
 29482 result. Characters generated by *fprintf()* and *printf()* are printed as if *fputc()* had been called.

29483 For the `a` and `A` conversion specifiers, if `FLT_RADIX` is a power of 2, the value shall be correctly  
 29484 rounded to a hexadecimal floating number with the given precision.

29485 For `a` and `A` conversions, if `FLT_RADIX` is not a power of 2 and the result is not exactly  
 29486 representable in the given precision, the result should be one of the two adjacent numbers in  
 29487 hexadecimal floating style with the given precision, with the extra stipulation that the error  
 29488 should have a correct sign for the current rounding direction.

29489 For the `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, if the number of significant decimal digits is at  
 29490 most `DECIMAL_DIG`, then the result should be correctly rounded. If the number of significant  
 29491 decimal digits is more than `DECIMAL_DIG` but the source value is exactly representable with  
 29492 `DECIMAL_DIG` digits, then the result should be an exact representation with trailing zeros.  
 29493 Otherwise, the source value is bounded by two adjacent decimal strings  $L < U$ , both having  
 29494 `DECIMAL_DIG` significant digits; the value of the resultant decimal string  $D$  should satisfy  $L \leq$   
 29495  $D \leq U$ , with the extra stipulation that the error should have a correct sign for the current  
 29496 rounding direction.

29497 CX The last data modification and last file status change timestamps of the file shall be marked for  
 29498 update between the call to a successful execution of *fprintf()* or *printf()* and the next successful  
 29499 completion of a call to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

## 29500 RETURN VALUE

29501 CX Upon successful completion, the *dprintf()*, *fprintf()*, and *printf()* functions shall return the  
 29502 number of bytes transmitted.

29503 Upon successful completion, the *sprintf()* function shall return the number of bytes written to *s*,  
 29504 excluding the terminating null byte.

29505 Upon successful completion, the *snprintf()* function shall return the number of bytes that would  
 29506 be written to *s* had *n* been sufficiently large excluding the terminating null byte.

29507 CX If an output error was encountered, these functions shall return a negative value and set *errno* to  
 29508 indicate the error.

29509 If the value of *n* is zero on a call to *snprintf()*, nothing shall be written, the number of bytes that  
 29510 would have been written had *n* been sufficiently large excluding the terminating null shall be

29511 returned, and *s* may be a null pointer.

## 29512 ERRORS

29513 CX For the conditions under which *dprintf()*, *fprintf()*, and *printf()* fail and may fail, refer to *fputc()*  
29514 or *fputcw()*.

29515 In addition, all forms of *fprintf()* may fail if:

29516 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been  
29517 detected.

29518 CX [EINVAL] There are insufficient arguments.

29519 The *dprintf()* function may fail if:

29520 [EBADF] The *fildev* argument is not a valid file descriptor.

29521 CX The *dprintf()*, *fprintf()*, and *printf()* functions may fail if:

29522 CX [ENOMEM] Insufficient storage space is available.

29523 The *snprintf()* function shall fail if:

29524 CX [EOVERFLOW] The value of *n* is greater than {INT\_MAX} or the number of bytes needed to  
29525 hold the output excluding the terminating null is greater than {INT\_MAX}.

## 29526 EXAMPLES

### 29527 Printing Language-Independent Date and Time

29528 The following statement can be used to print date and time using a language-independent  
29529 format:

```
29530 printf(format, weekday, month, day, hour, min);
```

29531 For American usage, *format* could be a pointer to the following string:

```
29532 "%s, %s %d, %d:%.2d\n"
```

29533 This example would produce the following message:

```
29534 Sunday, July 3, 10:02
```

29535 For German usage, *format* could be a pointer to the following string:

```
29536 "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

29537 This definition of *format* would produce the following message:

```
29538 Sonntag, 3. Juli, 10:02
```

### 29539 Printing File Information

29540 The following example prints information about the type, permissions, and number of links of a  
29541 specific file in a directory.

29542 The first two calls to *printf()* use data decoded from a previous *stat()* call. The user-defined  
29543 *strperm()* function shall return a string similar to the one at the beginning of the output for the  
29544 following command:

```
29545 ls -l
```

29546 The next call to *printf()* outputs the owner's name if it is found using *getpwuid()*; the *getpwuid()*  
29547 function shall return a **passwd** structure from which the name of the user is extracted. If the user  
29548 name is not found, the program instead prints out the numeric value of the user ID.

29549 The next call prints out the group name if it is found using *getgrgid()*; *getgrgid()* is very similar

29550 to *getpwuid()* except that it shall return group information based on the group number. Once  
 29551 again, if the group is not found, the program prints the numeric value of the group for the entry.  
 29552 The final call to *printf()* prints the size of the file.

```

29553 #include <stdio.h>
29554 #include <sys/types.h>
29555 #include <pwd.h>
29556 #include <grp.h>

29557 char *strperm (mode_t);
29558 ...
29559 struct stat statbuf;
29560 struct passwd *pwd;
29561 struct group *grp;
29562 ...
29563 printf("%10.10s", strperm (statbuf.st_mode));
29564 printf("%4d", statbuf.st_nlink);

29565 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
29566 printf(" %-8.8s", pwd->pw_name);
29567 else
29568 printf(" %-8ld", (long) statbuf.st_uid);
29569 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
29570 printf(" %-8.8s", grp->gr_name);
29571 else
29572 printf(" %-8ld", (long) statbuf.st_gid);
29573 printf("%9jd", (intmax_t) statbuf.st_size);
29574 ...

```

#### 29575 **Printing a Localized Date String**

29576 The following example gets a localized date string. The *nl\_langinfo()* function shall return the  
 29577 localized date string, which specifies the order and layout of the date. The *strftime()* function  
 29578 takes this information and, using the **tm** structure for values, places the date and time  
 29579 information into *datestring*. The *printf()* function then outputs *datestring* and the name of the  
 29580 entry.

```

29581 #include <stdio.h>
29582 #include <time.h>
29583 #include <langinfo.h>
29584 ...
29585 struct dirent *dp;
29586 struct tm *tm;
29587 char datestring[256];
29588 ...
29589 strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
29590 printf(" %s %s\n", datestring, dp->d_name);
29591 ...

```

29592

**Printing Error Information**

29593

The following example uses *fprintf()* to write error information to standard error.

29594

29595

29596

29597

In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If the file already exists, this is an error, as indicated by the **O\_EXCL** flag on the *open()* function. If the call fails, the program assumes that someone else is updating the password file, and the program exits.

29598

29599

The next group of calls saves a new password file as the current password file by creating a link between **LOCKFILE** and the new password file **PASSWDFILE**.

29600

29601

29602

29603

29604

29605

29606

29607

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
 fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
 exit(1);
}
...
if (link(LOCKFILE, PASSWDFILE) == -1) {
 fprintf(stderr, "Link error: %s\n", strerror(errno));
 exit(1);
}
...

```

29625

**Printing Usage Information**

29626

29627

The following example checks to make sure the program has the necessary arguments, and uses *fprintf()* to print usage information if the expected number of arguments is not present.

29628

29629

29630

29631

29632

29633

29634

29635

29636

```
#include <stdio.h>
#include <stdlib.h>
...
char *Options = "hdbt1";
...
if (argc < 2) {
 fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
}
...

```



29637

**Formatting a Decimal String**

29638

The following example prints a key and data pair on *stdout*. Note use of the '\*' (asterisk) in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

29639

29640

29641

```
#include <stdio.h>
```

29642

```
...
```

29643

```
long i;
```

29644

```
char *keyst;
```

29645

```
int elementlen, len;
```

29646

```
...
```

29647

```
while (len < elementlen) {
```

29648

```
...
```

29649

```
 printf("%s Element%0*ld\n", keyst, elementlen, i);
```

29650

```
...
```

29651

```
}
```

29652

**Creating a Filename**

29653

The following example creates a filename using information from a previous *getpwnam()* function that returned the HOME directory of the user.

29654

29655

```
#include <stdio.h>
```

29656

```
#include <sys/types.h>
```

29657

```
#include <unistd.h>
```

29658

```
...
```

29659

```
char filename[PATH_MAX+1];
```

29660

```
struct passwd *pw;
```

29661

```
...
```

29662

```
sprintf(filename, "%s/%d.out", pw->pw_dir, getpid());
```

29663

```
...
```

29664

**Reporting an Event**

29665

The following example loops until an event has timed out. The *pause()* function waits forever unless it receives a signal. The *fprintf()* statement should never occur due to the possible return values of *pause()*.

29666

29667

29668

```
#include <stdio.h>
```

29669

```
#include <unistd.h>
```

29670

```
#include <string.h>
```

29671

```
#include <errno.h>
```

29672

```
...
```

29673

```
while (!event_complete) {
```

29674

```
...
```

29675

```
 if (pause() != -1 || errno != EINTR)
```

29676

```
 fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
```

29677

```
}
```

29678

```
...
```

29679

**Printing Monetary Information**

29680

The following example uses *strfmon()* to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

29681

29682

29683

```
#include <monetary.h>
```

29684

```
#include <stdio.h>
```

29685

```
...
```

29686

```
struct tblfmt {
```

29687

```
 char *format;
```

29688

```
 char *description;
```

29689

```
};
```

29690

```
struct tblfmt table[] = {
```

29691

```
 { "%n", "default formatting" },
```

29692

```
 { "%11n", "right align within an 11 character field" },
```

29693

```
 { "%#5n", "aligned columns for values up to 99999" },
```

29694

```
 { "%=#5n", "specify a fill character" },
```

29695

```
 { "%=0#5n", "fill characters do not use grouping" },
```

29696

```
 { "%^#5n", "disable the grouping separator" },
```

29697

```
 { "%^#5.0n", "round off to whole units" },
```

29698

```
 { "%^#5.4n", "increase the precision" },
```

29699

```
 { "%(#5n", "use an alternative pos/neg style" },
```

29700

```
 { "%!(#5n", "disable the currency symbol" },
```

29701

```
};
```

29702

```
...
```

29703

```
float input[3];
```

29704

```
int i, j;
```

29705

```
char convbuf[100];
```

29706

```
...
```

29707

```
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
```

29708

```
if (j == 0) {
```

29709

```
 printf("%s%s%s\n", table[i].format,
```

29710

```
 convbuf, table[i].description);
```

29711

```
}
```

29712

```
else {
```

29713

```
 printf("%s\n", convbuf);
```

29714

```
}
```

29715

```
...
```

29716

**Printing Wide Characters**

29717

The following example prints a series of wide characters. Suppose that "L'@'" expands to three bytes:

29718

29719

```
wchar_t wz [3] = L"@@"; // Zero-terminated
```

29720

```
wchar_t wn [3] = L"@@"; // Unterminated
```

29721

```
fprintf (stdout, "%ls", wz); // Outputs 6 bytes
```

29722

```
fprintf (stdout, "%ls", wn); // Undefined because wn has no terminator
```

29723

```
fprintf (stdout, "%4ls", wz); // Outputs 3 bytes
```

29724

```
fprintf (stdout, "%4ls", wn); // Outputs 3 bytes; no terminator needed
```

29725

```
fprintf (stdout, "%9ls", wz); // Outputs 6 bytes
```

29726

```
fprintf (stdout, "%9ls", wn); // Outputs 9 bytes; no terminator needed
```

29727

```
fprintf (stdout, "%10ls", wz); // Outputs 6 bytes
```

29728            `fprintf (stdout, "%10ls", wn); // Undefined because wn has no terminator`

29729            In the last line of the example, after processing three characters, nine bytes have been output.  
29730            The fourth character must then be examined to determine whether it converts to one byte or  
29731            more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth  
29732            character in the array, the behavior is undefined.

### 29733 APPLICATION USAGE

29734            If the application calling `fprintf()` has any objects of type `wint_t` or `wchar_t`, it must also include  
29735            the `<wchar.h>` header to have these objects defined.

### 29736 RATIONALE

29737            None.

### 29738 FUTURE DIRECTIONS

29739            None.

### 29740 SEE ALSO

29741            [fputc\(\)](#), [fscanf\(\)](#), [setlocale\(\)](#), [strfmon\(\)](#), [wctomb\(\)](#)

29742            XBD Chapter 7 (on page 121), `<stdio.h>`, `<wchar.h>` |

### 29743 CHANGE HISTORY

29744            First released in Issue 1. Derived from Issue 1 of the SVID.

#### 29745 Issue 5

29746            Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the `l` (ell) qualifier can  
29747            now be used with `c` and `s` conversion specifiers.

29748            The `snprintf()` function is new in Issue 5.

#### 29749 Issue 6

29750            Extensions beyond the ISO C standard are marked.

29751            The normative text is updated to avoid use of the term “must” for application requirements.

29752            The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 29753            • The prototypes for `fprintf()`, `printf()`, `snprintf()`, and `sprintf()` are updated, and the XSI  
29754            shading is removed from `snprintf()`.
- 29755            • The description of `snprintf()` is aligned with the ISO C standard. Note that this supersedes  
29756            the `snprintf()` description in The Open Group Base Resolution bwg98-006, which changed  
29757            the behavior from Issue 5.
- 29758            • The DESCRIPTION is updated.

29759            The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion  
29760            specification” consistently.

29761            ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

29762            An example of printing wide characters is added.

#### 29763 Issue 7

29764            ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #68 (SD5-XSH-ERN-70) is applied,  
29765            revising the description of `g` and `G`.

29766            ISO C TC2 #68 is applied. +

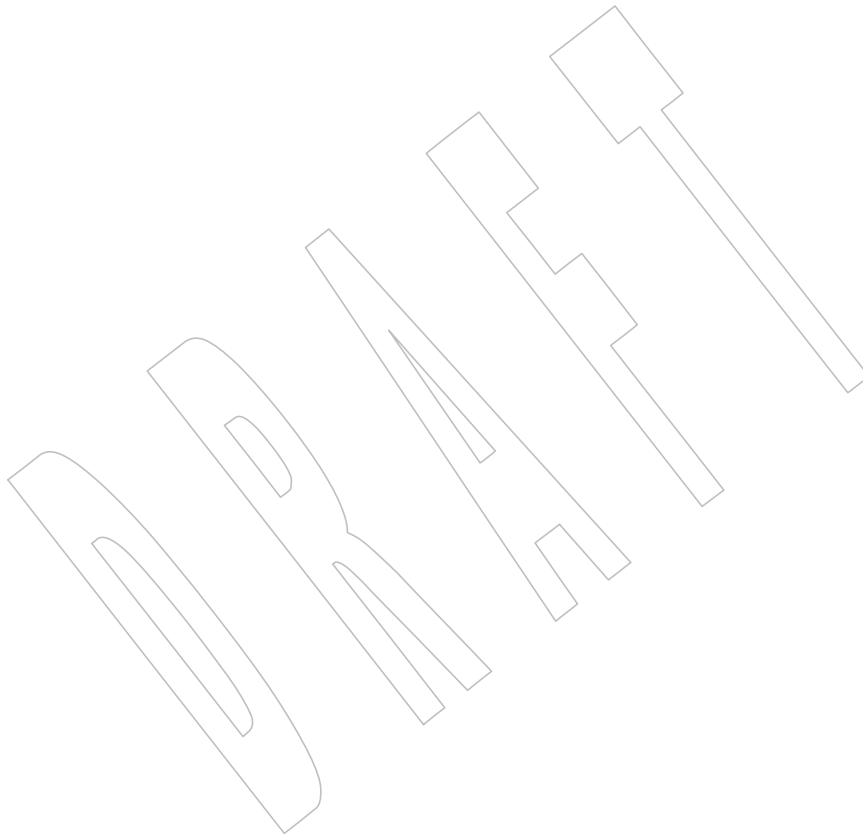
29767            SD5-XSH-ERN-174 is applied. +

29768            The `dprintf()` function is added from The Open Group Technical Standard, 2006, Extended API  
29769            Set Part 1.

29770            Functionality relating to the `%n$` form of conversion specification and the `'` (apostrophe) flag

- 29771 is moved from the XSI option to the Base.
- 29772 Changes are made related to support for finegrained timestamps.

+



29773 **NAME**

29774 fputc — put a byte on a stream

29775 **SYNOPSIS**

29776 #include &lt;stdio.h&gt;

29777 int fputc(int *c*, FILE \**stream*);29778 **DESCRIPTION**

29779 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29780 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29781 volume of POSIX.1-200x defers to the ISO C standard.

29782 The *fputc()* function shall write the byte specified by *c* (converted to an **unsigned char**) to the  
 29783 output stream pointed to by *stream*, at the position indicated by the associated file-position  
 29784 indicator for the stream (if defined), and shall advance the indicator appropriately. If the file  
 29785 cannot support positioning requests, or if the stream was opened with append mode, the byte  
 29786 shall be appended to the output stream.

29787 CX The last data modification and last file status change timestamps of the file shall be marked for  
 29788 update between the successful execution of *fputc()* and the next successful completion of a call  
 29789 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

29790 **RETURN VALUE**

29791 Upon successful completion, *fputc()* shall return the value it has written. Otherwise, it shall  
 29792 CX return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the  
 29793 error.

29794 **ERRORS**

29795 The *fputc()* function shall fail if either the *stream* is unbuffered or the *stream's* buffer needs to be  
 29796 flushed, and:

29797 CX **[EAGAIN]** The O\_NONBLOCK flag is set for the file descriptor underlying *stream* and  
 29798 the thread would be delayed in the write operation.

29799 CX **[EBADF]** The file descriptor underlying *stream* is not a valid file descriptor open for  
 29800 writing.

29801 CX **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size.

29802 XSI **[EFBIG]** An attempt was made to write to a file that exceeds the file size limit of the  
 29803 process.

29804 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the  
 29805 offset maximum.

29806 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data  
 29807 was transferred.

29808 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background  
 29809 process group attempting to write to its controlling terminal, TOSTOP is set,  
 29810 the process is neither ignoring nor blocking SIGTTOU, and the process group  
 29811 of the process is orphaned. This error may also be returned under  
 29812 implementation-defined conditions.

29813 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

29814 CX **[EPIPE]** An attempt is made to write to a pipe or FIFO that is not open for reading by  
 29815 any process. A SIGPIPE signal shall also be sent to the thread.

- 29816 The *fputc()* function may fail if:
- 29817 CX [ENOMEM] Insufficient storage space is available.
- 29818 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
29819 capabilities of the device.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO***error(), fopen(), getrlimit(), putc(), puts(), setbuf(), ulimit*

XBD &lt;stdio.h&gt;

**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

Large File Summit extensions are added.

**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] and [EFBIG] mandatory error conditions are added.
- The [ENOMEM] and [ENXIO] optional error conditions are added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/37 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

**Issue 7**

Changes are made related to support for finegrained timestamps.

29846 **NAME**  
 29847 fputs — put a string on a stream

29848 **SYNOPSIS**  
 29849 #include <stdio.h>

29850 int fputs(const char \*restrict s, FILE \*restrict stream);

#### 29851 DESCRIPTION

29852 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29853 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29854 volume of POSIX.1-200x defers to the ISO C standard.

29855 The *fputs()* function shall write the null-terminated string pointed to by *s* to the stream pointed  
 29856 to by *stream*. The terminating null byte shall not be written.

29857 CX The last data modification and last file status change timestamps of the file shall be marked for  
 29858 update between the successful execution of *fputs()* and the next successful completion of a call  
 29859 to *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

#### 29860 RETURN VALUE

29861 Upon successful completion, *fputs()* shall return a non-negative number. Otherwise, it shall  
 29862 CX return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

#### 29863 ERRORS

29864 Refer to *fputc()*.

#### 29865 EXAMPLES

##### 29866 Printing to Standard Output

29867 The following example gets the current time, converts it to a string using *localtime()* and  
 29868 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to  
 29869 an event for which it is waiting.

```
29870 #include <time.h>
29871 #include <stdio.h>
29872 ...
29873 time_t now;
29874 int minutes_to_event;
29875 ...
29876 time(&now);
29877 printf("The time is ");
29878 fputs(asctime(localtime(&now)), stdout);
29879 printf("There are still %d minutes to the event.\n",
29880 minutes_to_event);
29881 ...
```

#### 29882 APPLICATION USAGE

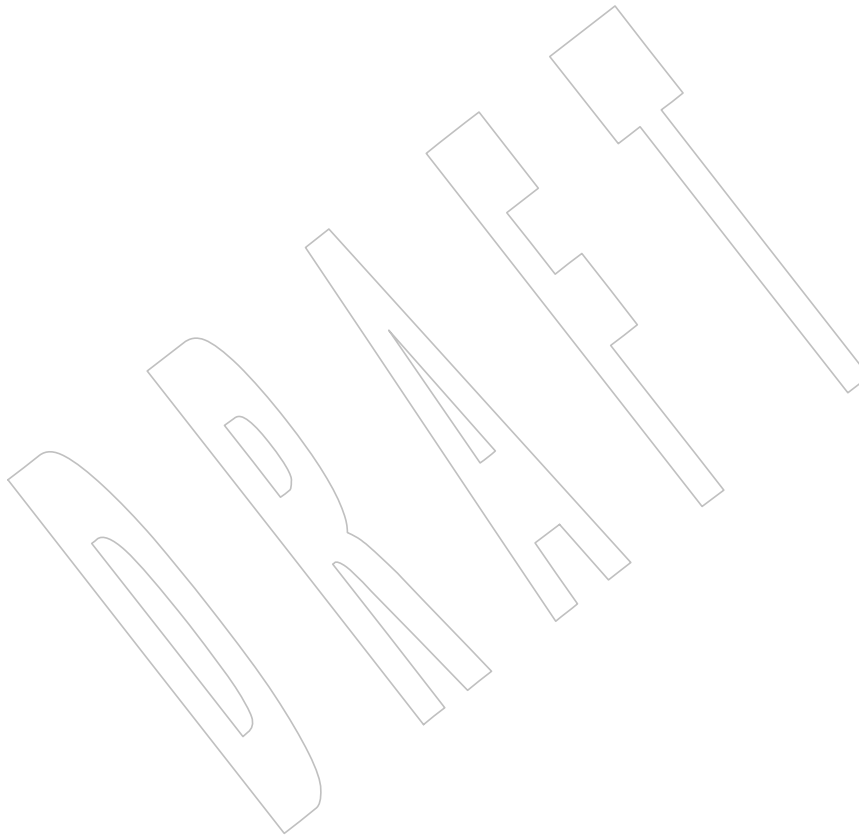
29883 The *puts()* function appends a <newline> while *fputs()* does not.

#### 29884 RATIONALE

29885 None.

**fputs()**

|       |                                                                                            |   |
|-------|--------------------------------------------------------------------------------------------|---|
| 29886 | <b>FUTURE DIRECTIONS</b>                                                                   |   |
| 29887 | None.                                                                                      |   |
| 29888 | <b>SEE ALSO</b>                                                                            |   |
| 29889 | <i>fopen()</i> , <i>putc()</i> , <i>puts()</i>                                             |   |
| 29890 | XBD <stdio.h>                                                                              |   |
| 29891 | <b>CHANGE HISTORY</b>                                                                      |   |
| 29892 | First released in Issue 1. Derived from Issue 1 of the SVID.                               |   |
| 29893 | <b>Issue 6</b>                                                                             |   |
| 29894 | Extensions beyond the ISO C standard are marked.                                           |   |
| 29895 | The <i>fputs()</i> prototype is updated for alignment with the ISO/IEC 9899:1999 standard. |   |
| 29896 | <b>Issue 7</b>                                                                             |   |
| 29897 | Changes are made related to support for finegrained timestamps.                            | + |





29898 **NAME**  
 29899 `fputc` — put a wide-character code on a stream

29900 **SYNOPSIS**

29901 `#include <stdio.h>`  
 29902 `#include <wchar.h>`  
 29903 `wint_t fputc(wchar_t wc, FILE *stream);`

29904 **DESCRIPTION**

29905 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 29906 conflict between the requirements described here and the ISO C standard is unintentional. This  
 29907 volume of POSIX.1-200x defers to the ISO C standard.

29908 The `fputc()` function shall write the character corresponding to the wide-character code `wc` to  
 29909 the output stream pointed to by `stream`, at the position indicated by the associated file-position  
 29910 indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot  
 29911 support positioning requests, or if the stream was opened with append mode, the character is  
 29912 appended to the output stream. If an error occurs while writing the character, the shift state of  
 29913 the output file is left in an undefined state.

29914 CX The last data modification and last file status change timestamps of the file shall be marked for  
 29915 update between the successful execution of `fputc()` and the next successful completion of a call  
 29916 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.

29917 **RETURN VALUE**

29918 Upon successful completion, `fputc()` shall return `wc`. Otherwise, it shall return WEOF, the error  
 29919 CX indicator for the stream shall be set, and `errno` shall be set to indicate the error.

29920 **ERRORS**

29921 The `fputc()` function shall fail if either the stream is unbuffered or data in the `stream`'s buffer  
 29922 needs to be written, and:

29923 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor underlying `stream` and  
 29924 the thread would be delayed in the write operation.

29925 CX **[EBADF]** The file descriptor underlying `stream` is not a valid file descriptor open for  
 29926 writing.

29927 CX **[EFBIG]** An attempt was made to write to a file that exceeds the maximum file size or  
 29928 the file size limit of the process.

29929 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the  
 29930 offset maximum associated with the corresponding stream.

29931 **[EILSEQ]** The wide-character code `wc` does not correspond to a valid character.

29932 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data  
 29933 was transferred.

29934 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background  
 29935 process group attempting to write to its controlling terminal, `TOSTOP` is set,  
 29936 the process is neither ignoring nor blocking `SIGTTOU`, and the process group  
 29937 of the process is orphaned. This error may also be returned under  
 29938 implementation-defined conditions.

29939 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

**fputwc()**

29940 CX [EPIPE] An attempt is made to write to a pipe or FIFO that is not open for reading by  
29941 any process. A SIGPIPE signal shall also be sent to the thread.

29942 The *fputwc()* function may fail if:

29943 CX [ENOMEM] Insufficient storage space is available.

29944 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
29945 capabilities of the device.

**EXAMPLES**

29946 None.  
29947

**APPLICATION USAGE**

29948 None.  
29949

**RATIONALE**

29950 None.  
29951

**FUTURE DIRECTIONS**

29952 None.  
29953

**SEE ALSO**

29954 *ferror()*, *fopen()*, *setbuf()*, *ulimit*  
29955

29956 XBD [<stdio.h>](#), [<wchar.h>](#) |

**CHANGE HISTORY**

29957 First released in Issue 4. Derived from the MSE working draft.  
29958

**Issue 5**

29959 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*  
29960 is changed from **wint\_t** to **wchar\_t**.  
29961

29962 The Optional Header (OH) marking is removed from **<stdio.h>**.

29963 Large File Summit extensions are added.

**Issue 6**

29964 Extensions beyond the ISO C standard are marked.  
29965

29966 The following new requirements on POSIX implementations derive from alignment with the  
29967 Single UNIX Specification:

- 29968 • The [EFBIG] and [EIO] mandatory error conditions are added.
- 29969 • The [ENOMEM] and [ENXIO] optional error conditions are added.

29970 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/38 is applied, updating the [EAGAIN]  
29971 error in the ERRORS section from “the process would be delayed” to “the thread would be  
29972 delayed”.

**Issue 7**

29973 Changes are made related to support for finegrained timestamps.  
29974 +

29975 **NAME**29976 `fputws` — put a wide-character string on a stream29977 **SYNOPSIS**29978 `#include <stdio.h>`29979 `#include <wchar.h>`29980 `int fputws(const wchar_t *restrict ws, FILE *restrict stream);`29981 **DESCRIPTION**29982 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
29983 conflict between the requirements described here and the ISO C standard is unintentional. This  
29984 volume of POSIX.1-200x defers to the ISO C standard.29985 The `fputws()` function shall write a character string corresponding to the (null-terminated) wide-  
29986 character string pointed to by `ws` to the stream pointed to by `stream`. No character corresponding  
29987 to the terminating null wide-character code shall be written.29988 CX The last data modification and last file status change timestamps of the file shall be marked for  
29989 update between the successful execution of `fputws()` and the next successful completion of a call  
29990 to `fflush()` or `fclose()` on the same stream or a call to `exit()` or `abort()`.29991 **RETURN VALUE**29992 Upon successful completion, `fputws()` shall return a non-negative number. Otherwise, it shall  
29993 CX return `-1`, set an error indicator for the stream, and set `errno` to indicate the error.29994 **ERRORS**29995 Refer to `fputwc()`.29996 **EXAMPLES**

29997 None.

29998 **APPLICATION USAGE**29999 The `fputws()` function does not append a <newline>.30000 **RATIONALE**

30001 None.

30002 **FUTURE DIRECTIONS**

30003 None.

30004 **SEE ALSO**30005 `fopen()`30006 XBD `<stdio.h>`, `<wchar.h>`30007 **CHANGE HISTORY**

30008 First released in Issue 4. Derived from the MSE working draft.

30009 **Issue 5**30010 The Optional Header (OH) marking is removed from `<stdio.h>`.30011 **Issue 6**

30012 Extensions beyond the ISO C standard are marked.

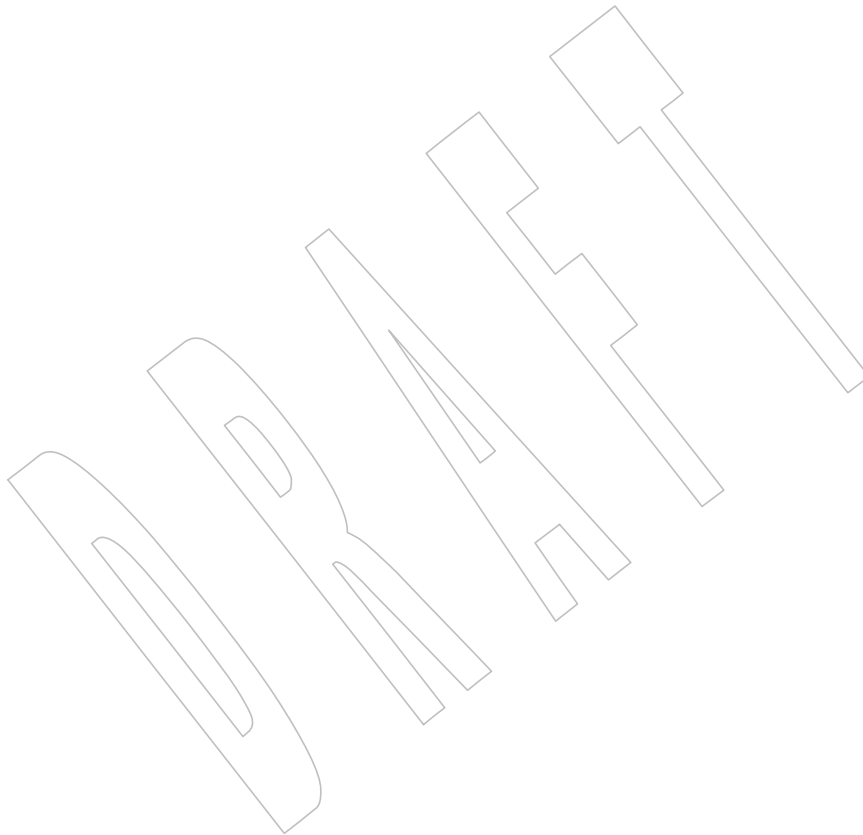
30013 The `fputws()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

30014  
30015

**Issue 7**

Changes are made related to support for finegrained timestamps.

+



30016 **NAME**

30017 fread — binary input

30018 **SYNOPSIS**

30019 #include &lt;stdio.h&gt;

30020 size\_t fread(void \*restrict ptr, size\_t size, size\_t nitems,  
30021 FILE \*restrict stream);30022 **DESCRIPTION**30023 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30024 conflict between the requirements described here and the ISO C standard is unintentional. This  
30025 volume of POSIX.1-200x defers to the ISO C standard.30026 The *fread()* function shall read into the array pointed to by *ptr* up to *nitems* elements whose size  
30027 is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall  
30028 be made to the *fgetc()* function and the results stored, in the order read, in an array of **unsigned**  
30029 **char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be  
30030 advanced by the number of bytes successfully read. If an error occurs, the resulting value of the  
30031 file position indicator for the stream is unspecified. If a partial element is read, its value is  
30032 unspecified.30033 CX The *fread()* function may mark the last data access timestamp of the file associated with *stream* |  
30034 for update. The last data access timestamp shall be marked for update by the first successful |  
30035 execution of *fgetc()*, *fgets()*, *fgetwc()*, *fgetws()*, *fread()*, *fscanf()*, *getc()*, *getchar()*, *gets()*, or *scanf()*  
30036 using *stream* that returns data not supplied by a prior call to *ungetc()* or *ungetwc()*.30037 **RETURN VALUE**30038 Upon successful completion, *fread()* shall return the number of elements successfully read which  
30039 is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread()*  
30040 shall return 0 and the contents of the array and the state of the stream remain unchanged.30041 CX Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall  
30042 be set to indicate the error.30043 **ERRORS**30044 Refer to *fgetc()*.30045 **EXAMPLES**30046 **Reading from a Stream**30047 The following example reads a single element from the *fp* stream into the array pointed to by  
30048 *buf*.30049 #include <stdio.h>  
30050 ...  
30051 size\_t bytes\_read;  
30052 char buf[100];  
30053 FILE \*fp;  
30054 ...  
30055 bytes\_read = fread(buf, sizeof(buf), 1, fp);  
30056 ...

30057

**APPLICATION USAGE**

30058

The *ferror()* or *feof()* functions must be used to distinguish between an error condition and an end-of-file condition.

30059

30060

Because of possible differences in element length and byte ordering, files written using *fwrite()* are application-dependent, and possibly cannot be read using *fread()* by a different application or by the same application on a different processor.

30061

30062

30063

**RATIONALE**

30064

None.

30065

**FUTURE DIRECTIONS**

30066

None.

30067

**SEE ALSO**

30068

*feof()*, *ferror()*, *fgetc()*, *fopen()*, *fscanf()*, *getc()*, *gets()*

30069

XBD [<stdio.h>](#)

30070

**CHANGE HISTORY**

30071

First released in Issue 1. Derived from Issue 1 of the SVID.

30072

**Issue 6**

30073

Extensions beyond the ISO C standard are marked.

30074

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

30075

- The *fread()* prototype is updated.

30076

- The DESCRIPTION is updated to describe how the bytes from a call to *fgetc()* are stored.

30077

**Issue 7**

30078

Changes are made related to support for finegrained timestamps.

30079 **NAME**

30080 free — free allocated memory

30081 **SYNOPSIS**

30082 #include &lt;stdlib.h&gt;

30083 void free(void \*ptr);

30084 **DESCRIPTION**

30085 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30086 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30087 volume of POSIX.1-200x defers to the ISO C standard.

30088 The *free()* function shall cause the space pointed to by *ptr* to be deallocated; that is, made  
 30089 available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the  
 30090 ADV argument does not match a pointer earlier returned by the *calloc()*, *malloc()*, *posix\_memalign()*,  
 30091 *realloc()*, or *strdup()* function, or if the space has been deallocated by a call to *free()* or *realloc()*,  
 30092 the behavior is undefined.

30093 Any use of a pointer that refers to freed space results in undefined behavior.

30094 **RETURN VALUE**30095 The *free()* function shall not return a value.30096 **ERRORS**

30097 No errors are defined.

30098 **EXAMPLES**

30099 None.

30100 **APPLICATION USAGE**

30101 There is now no requirement for the implementation to support the inclusion of &lt;malloc.h&gt;.

30102 **RATIONALE**

30103 None.

30104 **FUTURE DIRECTIONS**

30105 None.

30106 **SEE ALSO**30107 *calloc()*, *getdelim()*, *malloc()*, *open\_memstream()*, *realloc()*, *strdup()*, *wcsdup()*

30108 XBD &lt;stdlib.h&gt;

30109 **CHANGE HISTORY**

30110 First released in Issue 1. Derived from Issue 1 of the SVID.

30111 **Issue 6**30112 Reference to the *valloc()* function is removed.

30113 **NAME**  
 30114 freeaddrinfo, getaddrinfo — get address information

30115 **SYNOPSIS**

```
30116 #include <sys/socket.h>
30117 #include <netdb.h>

30118 void freeaddrinfo(struct addrinfo *ai);
30119 int getaddrinfo(const char *restrict nodename,
30120 const char *restrict servname,
30121 const struct addrinfo *restrict hints,
30122 struct addrinfo **restrict res);
```

30123 **DESCRIPTION**

30124 The *freeaddrinfo()* function shall free one or more **addrinfo** structures returned by *getaddrinfo()*,  
 30125 along with any additional storage associated with those structures. If the *ai\_next* field of the  
 30126 structure is not null, the entire list of structures shall be freed. The *freeaddrinfo()* function shall  
 30127 support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo()*.

30128 The *getaddrinfo()* function shall translate the name of a service location (for example, a host  
 30129 name) and/or a service name and shall return a set of socket addresses and associated  
 30130 information to be used in creating a socket with which to address the specified service.

30131 **Note:** In many cases it is implemented by the Domain Name System, as documented in RFC 1034,  
 30132 RFC 1035, and RFC 1886.

30133 The *freeaddrinfo()* and *getaddrinfo()* functions shall be thread-safe.

30134 The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated  
 30135 strings. One or both of these two arguments shall be supplied by the application as a non-null  
 30136 pointer.

30137 The format of a valid name depends on the address family or families. If a specific family is not  
 30138 given and the name could be interpreted as valid within multiple supported families, the  
 30139 implementation shall attempt to resolve the name in all supported families and, in absence of  
 30140 errors, one or more results shall be returned.

30141 If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If  
 30142 IP6 the specified address family is AF\_INET, AF\_INET6, or AF\_UNSPEC, valid descriptive names  
 30143 include host names. If the specified address family is AF\_INET or AF\_UNSPEC, address strings  
 30144 using Internet standard dot notation as specified in *inet\_addr()* are valid.

30145 IP6 If the specified address family is AF\_INET6 or AF\_UNSPEC, standard IPv6 text forms described  
 30146 in *inet\_ntop()* are valid.

30147 If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the  
 30148 requested service location is local to the caller.

30149 If *servname* is null, the call shall return network-level addresses for the specified *nodename*. If  
 30150 *servname* is not null, it is a null-terminated character string identifying the requested service.  
 30151 This can be either a descriptive name or a numeric representation suitable for use with the  
 30152 IP6 address family or families. If the specified address family is AF\_INET, AF\_INET6, or  
 30153 AF\_UNSPEC, the service can be specified as a string specifying a decimal port number.

30154 If the *hints* argument is not null, it refers to a structure containing input values that may direct  
 30155 the operation by providing options and by limiting the returned information to a specific socket  
 30156 type, address family, and/or protocol. In this *hints* structure every member other than *ai\_flags*,  
 30157 *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be set to zero or a null pointer. A value of  
 30158 AF\_UNSPEC for *ai\_family* means that the caller shall accept any address family. A value of zero



30159 for *ai\_socktype* means that the caller shall accept any socket type. A value of zero for *ai\_protocol*  
 30160 means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if  
 30161 it referred to a structure containing the value zero for the *ai\_flags*, *ai\_socktype*, and *ai\_protocol*  
 30162 fields, and AF\_UNSPEC for the *ai\_family* field.

30163 The *ai\_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-  
 30164 inclusive OR of one or more of the values AI\_PASSIVE, AI\_CANONNAME,  
 30165 AI\_NUMERICHOST, AI\_NUMERICSERV, AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG.

30166 If the AI\_PASSIVE flag is specified, the returned address information shall be suitable for use in  
 30167 binding a socket for accepting incoming connections for the specified service. In this case, if the  
 30168 *nodename* argument is null, then the IP address portion of the socket address structure shall be  
 30169 set to INADDR\_ANY for an IPv4 address or IN6ADDR\_ANY\_INIT for an IPv6 address. If the  
 30170 AI\_PASSIVE flag is not specified, the returned address information shall be suitable for a call to  
 30171 *connect()* (for a connection-mode protocol) or for a call to *connect()*, *sendto()*, or *sendmsg()* (for a  
 30172 connectionless protocol). In this case, if the *nodename* argument is null, then the IP address  
 30173 portion of the socket address structure shall be set to the loopback address. The AI\_PASSIVE  
 30174 flag shall be ignored if the *nodename* argument is not null.

30175 If the AI\_CANONNAME flag is specified and the *nodename* argument is not null, the function  
 30176 shall attempt to determine the canonical name corresponding to *nodename* (for example, if  
 30177 *nodename* is an alias or shorthand notation for a complete name).

30178 **Note:** Since different implementations use different conceptual models, the terms “canonical name”  
 30179 and “alias” cannot be precisely defined for the general case. However, Domain Name System  
 30180 implementations are expected to interpret them as they are used in RFC 1034.

30181 A numeric host address string is not a “name”, and thus does not have a “canonical name”  
 30182 form; no address to host name translation is performed. See below for handling of the case  
 30183 where a canonical name cannot be obtained.

30184 If the AI\_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a  
 30185 numeric host address string. Otherwise, an [EAI\_NONAME] error is returned. This flag shall  
 30186 prevent any type of name resolution service (for example, the DNS) from being invoked.

30187 If the AI\_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a  
 30188 numeric port string. Otherwise, an [EAI\_NONAME] error shall be returned. This flag shall  
 30189 prevent any type of name resolution service (for example, NIS+) from being invoked.

30190 IP6 If the AI\_V4MAPPED flag is specified along with an *ai\_family* of AF\_INET6, then *getaddrinfo()*  
 30191 shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai\_addrlen*  
 30192 shall be 16). The AI\_V4MAPPED flag shall be ignored unless *ai\_family* equals AF\_INET6. If the  
 30193 AI\_ALL flag is used with the AI\_V4MAPPED flag, then *getaddrinfo()* shall return all matching  
 30194 IPv6 and IPv4 addresses. The AI\_ALL flag without the AI\_V4MAPPED flag is ignored.

30195 IP6 If the AI\_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4  
 30196 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6  
 30197 address is configured on the local system.

30198 The *ai\_socktype* field to which argument *hints* points specifies the socket type for the service, as  
 30199 defined in *socket()*. If a specific socket type is not given (for example, a value of zero) and the  
 30200 service name could be interpreted as valid with multiple supported socket types, the  
 30201 implementation shall attempt to resolve the service name for all supported socket types and, in  
 30202 the absence of errors, all possible results shall be returned. A non-zero socket type value shall  
 30203 limit the returned information to values with the specified socket type.

30204 If the *ai\_family* field to which *hints* points has the value AF\_UNSPEC, addresses shall be  
 30205 returned for use with any address family that can be used with the specified *nodename* and/or  
 30206 *servname*. Otherwise, addresses shall be returned for use only with the specified address family.  
 30207 If *ai\_family* is not AF\_UNSPEC and *ai\_protocol* is not zero, then addresses are returned for use

**freeaddrinfo()**

System Interfaces

30208 only with the specified address family and protocol; the value of *ai\_protocol* shall be interpreted  
30209 as in a call to the *socket()* function with the corresponding values of *ai\_family* and *ai\_protocol*.

**RETURN VALUE**

30210  
30211 A zero return value for *getaddrinfo()* indicates successful completion; a non-zero return value  
30212 indicates failure. The possible values for the failures are listed in the ERRORS section.

30213 Upon successful return of *getaddrinfo()*, the location to which *res* points shall refer to a linked list  
30214 of **addrinfo** structures, each of which shall specify a socket address and information for use in  
30215 creating a socket with which to use that socket address. The list shall include at least one  
30216 **addrinfo** structure. The *ai\_next* field of each structure contains a pointer to the next structure on  
30217 the list, or a null pointer if it is the last structure on the list. Each structure on the list shall  
30218 include values for use with a call to the *socket()* function, and a socket address for use with the  
30219 *connect()* function or, if the AI\_PASSIVE flag was specified, for use with the *bind()* function. The  
30220 fields *ai\_family*, *ai\_socktype*, and *ai\_protocol* shall be usable as the arguments to the *socket()*  
30221 function to create a socket suitable for use with the returned address. The fields *ai\_addr* and  
30222 *ai\_addrlen* are usable as the arguments to the *connect()* or *bind()* functions with such a socket,  
30223 according to the AI\_PASSIVE flag.

30224 If *nodename* is not null, and if requested by the AI\_CANONNAME flag, the *ai\_canonname* field of  
30225 the first returned **addrinfo** structure shall point to a null-terminated string containing the  
30226 canonical name corresponding to the input *nodename*; if the canonical name is not available, then  
30227 *ai\_canonname* shall refer to the *nodename* argument or a string with the same contents. The  
30228 contents of the *ai\_flags* field of the returned structures are undefined.

30229 All fields in socket address structures returned by *getaddrinfo()* that are not filled in through an  
30230 explicit argument (for example, *sin6\_flowinfo*) shall be set to zero.

30231 **Note:** This makes it easier to compare socket address structures.

**ERRORS**

30232 The *getaddrinfo()* function shall fail and return the corresponding error value if:

30233 [EAI\_AGAIN] The name could not be resolved at this time. Future attempts may succeed.

30234 [EAI\_BADFLAGS] The *flags* parameter had an invalid value.

30235 [EAI\_FAIL] A non-recoverable error occurred when attempting to resolve the name.

30236 [EAI\_FAMILY] The address family was not recognized.

30237 [EAI\_MEMORY] There was a memory allocation failure when trying to allocate storage for the  
30238 return value.

30239 [EAI\_NONAME] The name does not resolve for the supplied parameters.

30240 Neither *nodename* nor *servname* were supplied. At least one of these shall be  
30241 supplied.

30242 [EAI\_SERVICE] The service passed was not recognized for the specified socket type.

30243 [EAI\_SOCKTYPE] The intended socket type was not recognized.

30244 [EAI\_SYSTEM] A system error occurred; the error code can be found in *errno*.

**EXAMPLES**

30248  
30249 None.

**APPLICATION USAGE**

30250  
30251 If the caller handles only TCP and not UDP, for example, then the *ai\_protocol* member of the *hints*  
30252 structure should be set to IPPROTO\_TCP when *getaddrinfo()* is called.

30253  
30254 If the caller handles only IPv4 and not IPv6, then the *ai\_family* member of the *hints* structure  
should be set to AF\_INET when *getaddrinfo()* is called.

30255  
30256 The term “canonical name” is misleading; it is taken from the Domain Name System (RFC 2181).  
30257 It should be noted that the canonical name is a result of alias processing, and not necessarily a  
30258 unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this  
in the Domain Name System context.

**RATIONALE**

30259  
30260 None.

**FUTURE DIRECTIONS**

30261  
30262 None.

**SEE ALSO**

30263  
30264 *connect()*, *endservent()*, *gai\_strerror()*, *getnameinfo()*, *socket()*

30265  
30266 XBD <netdb.h>, <sys/socket.h>

**CHANGE HISTORY**

30267  
30268 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

30269  
30270 The **restrict** keyword is added to the *getaddrinfo()* prototype for alignment with the  
30271 ISO/IEC 9899:1999 standard.

30272  
30273 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/19 is applied, adding three notes to the  
30274 DESCRIPTION and adding text to the APPLICATION USAGE related to the term “canonical  
30275 name”. A reference to RFC 2181 is also added to the Informative References.

30276  
30277 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/20 is applied, making changes for  
30278 alignment with IPv6. These include the following:

- 30275 • Adding AI\_V4MAPPED, AI\_ALL, and AI\_ADDRCONFIG to the allowed values for the  
30276 *ai\_flags* field
- 30277 • Adding a description of AI\_ADDRCONFIG
- 30278 • Adding a description of the consequences of ignoring the AI\_PASSIVE flag.

30279  
30280 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/39 is applied, changing “corresponding  
value” to “corresponding error value” in the ERRORS section.

**Issue 7**

30281  
30282 Austin Group Interpretation 1003.1-2001 #013 is applied.

30283 **NAME**  
 30284 `freelocale` — free resources allocated for a locale object

30285 **SYNOPSIS**

30286 CX `#include <locale.h>`  
 30287 `void freelocale(locale_t locobj);`

30288 **DESCRIPTION**

30289 The `freelocale()` function shall cause the resources allocated for a locale object returned by a call  
 30290 to the `newlocale()` or `duplocale()` functions to be released.

30291 Any use of a locale object that has been freed results in undefined behavior.

30292 **RETURN VALUE**

30293 None.

30294 **ERRORS**

30295 None.

30296 **EXAMPLES**

30297 **Freeing Up a Locale Object**

30298 The following example shows a code fragment to free a locale object created by `newlocale()`:

```
30299 #include <locale.h>
30300 ...
30301 /* Every locale object allocated with newlocale() should be
30302 * freed using freelocale():
30303 */
30304 locale_t loc;
30305 /* Get the locale. */
30306 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
30307 /* ... Use the locale object ... */
30308 ...
30309 /* Free the locale object resources. */
30310 freelocale (loc);
```

30311 **APPLICATION USAGE**

30312 None.

30313 **RATIONALE**

30314 None.

30315 **FUTURE DIRECTIONS**

30316 None.

30317 **SEE ALSO**

30318 [duplocale\(\)](#), [newlocale\(\)](#), [uselocale\(\)](#)

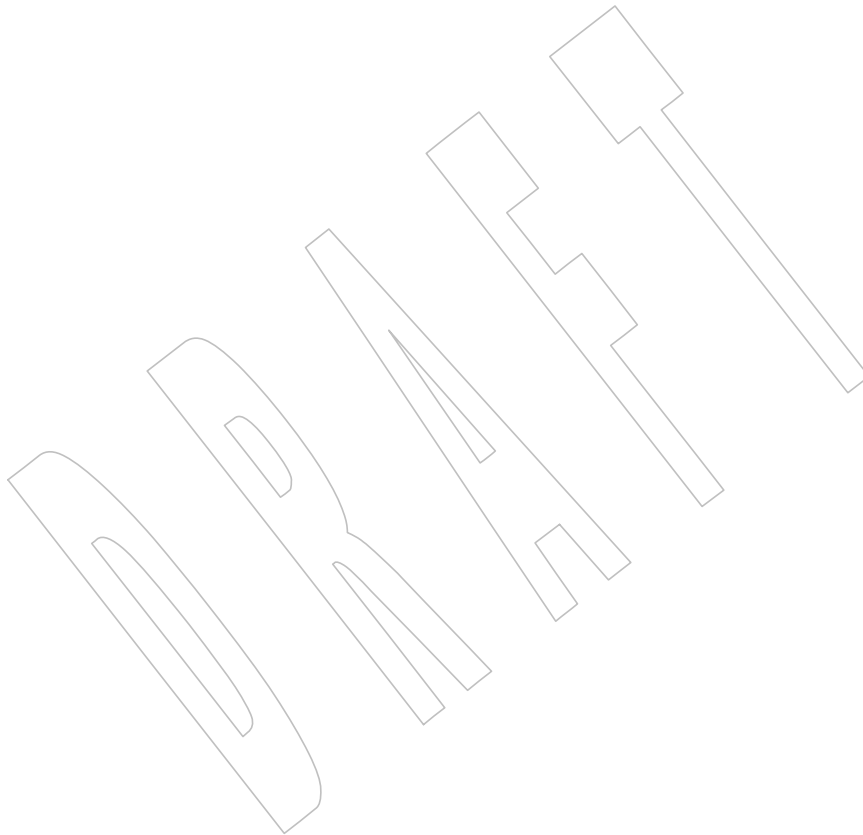
30319 XBD [<locale.h>](#)

30320

30321

**CHANGE HISTORY**

First released in Issue 7.



## 30322 NAME

30323 freopen — open a stream

## 30324 SYNOPSIS

30325 #include &lt;stdio.h&gt;

30326 FILE \*freopen(const char \*restrict filename, const char \*restrict mode,  
30327 FILE \*restrict stream);

## 30328 DESCRIPTION

30329 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30330 conflict between the requirements described here and the ISO C standard is unintentional. This  
30331 volume of POSIX.1-200x defers to the ISO C standard.30332 The *freopen()* function shall first attempt to flush the stream associated with *stream* as if by a call  
30333 to *fflush(stream)*. Failure to flush the stream successfully shall be ignored. If *filename* is not a null  
30334 pointer, *freopen()* shall close any file descriptor associated with *stream*. Failure to close the file  
30335 descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall  
30336 be cleared.30337 The *freopen()* function shall open the file whose pathname is the string pointed to by *filename*  
30338 and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in  
30339 *fopen()*. The *freopen()* function shall allocate a file descriptor in the same way as *open()*.

30340 The original stream shall be closed regardless of whether the subsequent open succeeds.

30341 If *filename* is a null pointer, the *freopen()* function shall attempt to change the mode of the stream  
30342 to that specified by *mode*, as if the name of the file currently associated with the stream had been  
30343 used. In this case, the file descriptor associated with the stream need not be closed if the call to  
30344 *freopen()* succeeds. It is implementation-defined which changes of mode are permitted (if any),  
30345 and under what circumstances.30346 XSI After a successful call to the *freopen()* function, the orientation of the stream shall be cleared, the  
30347 encoding rule shall be cleared, and the associated **mbstate\_t** object shall be set to describe an  
30348 initial conversion state.30349 CX The largest value that can be represented correctly in an object of type **off\_t** shall be established  
30350 as the offset maximum in the open file description.

## 30351 RETURN VALUE

30352 Upon successful completion, *freopen()* shall return the value of *stream*. Otherwise, a null pointer  
30353 CX shall be returned, and *errno* shall be set to indicate the error.

## 30354 ERRORS

30355 The *freopen()* function shall fail if:30356 CX [EACCES] Search permission is denied on a component of the path prefix, or the file  
30357 exists and the permissions specified by *mode* are denied, or the file does not  
30358 exist and write permission is denied for the parent directory of the file to be  
30359 created.30360 CX [EBADF] The file descriptor underlying the stream is not a valid file descriptor when  
30361 *filename* is a null pointer.30362 CX [EINTR] A signal was caught during *freopen()*.30363 CX [EISDIR] The named file is a directory and *mode* requires write access.

|       |    |                |                                                                                                                                                      |
|-------|----|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 30364 | CX | [ELOOP]        | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                                                           |
| 30365 |    |                |                                                                                                                                                      |
| 30366 | CX | [EMFILE]       | All file descriptors available to the process are currently open.                                                                                    |
| 30367 | CX | [ENAMETOOLONG] |                                                                                                                                                      |
| 30368 |    |                | The length of the <i>filename</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.                                     |
| 30369 |    |                |                                                                                                                                                      |
| 30370 | CX | [ENFILE]       | The maximum allowable number of files is currently open in the system.                                                                               |
| 30371 | CX | [ENOENT]       | A component of <i>filename</i> does not name an existing file or <i>filename</i> is an empty string.                                                 |
| 30372 |    |                |                                                                                                                                                      |
| 30373 | CX | [ENOSPC]       | The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.                  |
| 30374 |    |                |                                                                                                                                                      |
| 30375 | CX | [ENOTDIR]      | A component of the path prefix is not a directory.                                                                                                   |
| 30376 | CX | [ENXIO]        | The named file is a character special or block special file, and the device associated with this special file does not exist.                        |
| 30377 |    |                |                                                                                                                                                      |
| 30378 | CX | [EOVERFLOW]    | The named file is a regular file and the size of the file cannot be represented correctly in an object of type <b>off_t</b> .                        |
| 30379 |    |                |                                                                                                                                                      |
| 30380 | CX | [EROFS]        | The named file resides on a read-only file system and <i>mode</i> requires write access.                                                             |
| 30381 |    |                |                                                                                                                                                      |
| 30382 |    |                | The <i>freopen()</i> function may fail if:                                                                                                           |
| 30383 | CX | [EBADF]        | The mode with which the file descriptor underlying the stream was opened does not support the requested mode when <i>filename</i> is a null pointer. |
| 30384 |    |                |                                                                                                                                                      |
| 30385 | CX | [EINVAL]       | The value of the <i>mode</i> argument is not valid.                                                                                                  |
| 30386 | CX | [ELOOP]        | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.                                               |
| 30387 |    |                |                                                                                                                                                      |
| 30388 | CX | [ENAMETOOLONG] |                                                                                                                                                      |
| 30389 |    |                | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.                                              |
| 30390 |    |                |                                                                                                                                                      |
| 30391 | CX | [ENOMEM]       | Insufficient storage space is available.                                                                                                             |
| 30392 | CX | [ENXIO]        | A request was made of a nonexistent device, or the request was outside the capabilities of the device.                                               |
| 30393 |    |                |                                                                                                                                                      |
| 30394 | CX | [ETXTBSY]      | The file is a pure procedure (shared text) file that is being executed and <i>mode</i> requires write access.                                        |
| 30395 |    |                |                                                                                                                                                      |

## EXAMPLES

### Directing Standard Output to a File

The following example logs all standard output to the `/tmp/logfile` file.

```

30399 #include <stdio.h>
30400 ...
30401 FILE *fp;
30402 ...
30403 fp = freopen ("/tmp/logfile", "a+", stdout);
30404 ...

```

30405  
30406  
30407  
  
30408  
30409  
  
30410  
30411  
  
30412  
30413  
30414  
  
30415  
30416  
  
30417  
30418  
30419  
30420  
30421  
  
30422  
30423  
30424  
30425  
30426  
30427  
30428  
30429  
30430  
30431  
30432  
30433  
30434  
30435  
30436  
30437  
30438  
30439  
30440  
30441  
30442  
30443

**APPLICATION USAGE**

The *freopen()* function is typically used to attach the preopened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*fclose()*, *fdopen()*, *fflush()*, *fmemopen()*, *fopen()*, *mbsinit()*, *open()*, *open\_memstream()*

XBD **<stdio.h>**

**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

The DESCRIPTION is updated to indicate that the orientation of the stream is cleared and the conversion state of the stream is set to an initial conversion state by a successful call to the *freopen()* function.

Large File Summit extensions are added.

**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files.
- In the ERRORS section, the [Eoverflow] condition is added. This change is to support large files.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [EINVAL], [ENOMEM], [ENXIO], and [ETXTBSY] optional error conditions are added.

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *freopen()* prototype is updated.
- The DESCRIPTION is updated.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

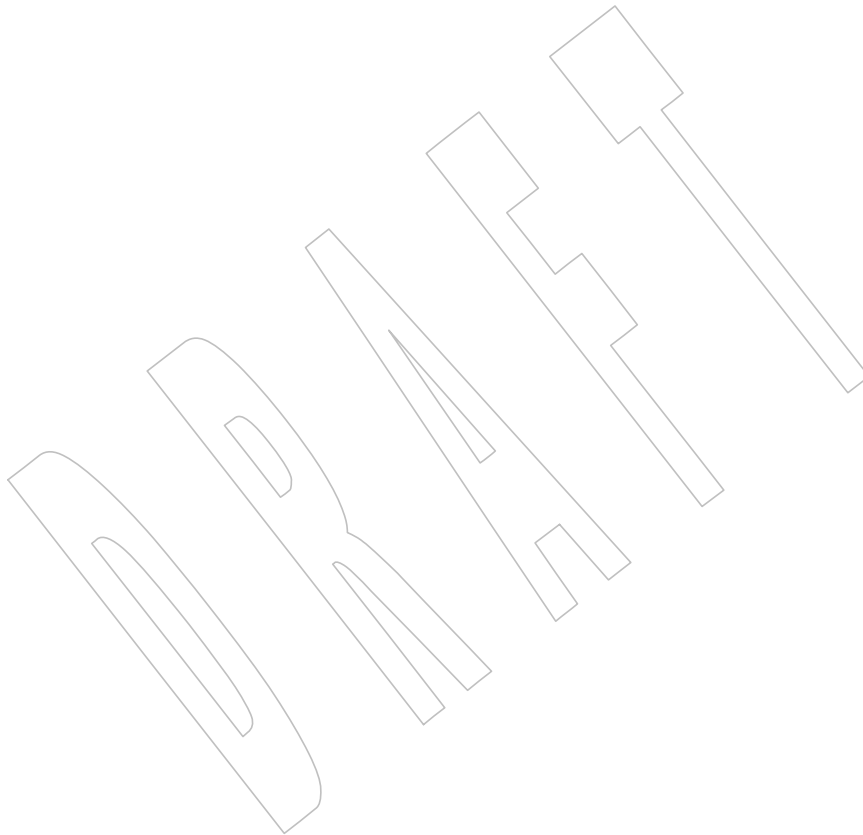
The DESCRIPTION is updated regarding failure to close, changing the “file” to “file descriptor”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/40 is applied, adding the following sentence to the DESCRIPTION: “In this case, the file descriptor associated with the stream need not be closed if the call to *freopen()* succeeds.”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/41 is applied, adding an mandatory [EBADF] error, and an optional [EBADF] error to the ERRORS section.



|       |                                                                                                        |   |
|-------|--------------------------------------------------------------------------------------------------------|---|
| 30444 | <b>Issue 7</b>                                                                                         |   |
| 30445 | Austin Group Interpretation 1003.1-2001 #043 is applied, clarifying that the <i>freopen()</i> function | - |
| 30446 | allocates a file descriptor as per <i>open()</i> .                                                     |   |
| 30447 | SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.                               | + |
| 30448 | SD5-XSH-ERN-150 is applied.                                                                            |   |



30449 **NAME**30450 `frexp`, `frexpf`, `frexpl` — extract mantissa and exponent from a double precision number30451 **SYNOPSIS**30452 `#include <math.h>`30453 `double frexp(double num, int *exp);`30454 `float frexpf(float num, int *exp);`30455 `long double frexpl(long double num, int *exp);`30456 **DESCRIPTION**30457 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any  
30458 conflict between the requirements described here and the ISO C standard is unintentional. This  
30459 volume of POSIX.1-200x defers to the ISO C standard.30460 These functions shall break a floating-point number *num* into a normalized fraction and an  
30461 integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.30462 **RETURN VALUE**30463 For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the  
30464 interval  $[\frac{1}{2}, 1)$  or 0, and *num* equals *x* times 2 raised to the power *\*exp*.30465 **MX** If *num* is NaN, a NaN shall be returned, and the value of *\*exp* is unspecified.30466 If *num* is  $\pm 0$ ,  $\pm 0$  shall be returned, and the value of *\*exp* shall be 0.30467 If *num* is  $\pm \text{Inf}$ , *num* shall be returned, and the value of *\*exp* is unspecified.30468 **ERRORS**

30469 No errors are defined.

30470 **EXAMPLES**

30471 None.

30472 **APPLICATION USAGE**

30473 None.

30474 **RATIONALE**

30475 None.

30476 **FUTURE DIRECTIONS**

30477 None.

30478 **SEE ALSO**30479 [\*isnan\(\)\*](#), [\*ldexp\(\)\*](#), [\*modf\(\)\*](#)30480 XBD [\*\*<math.h>\*\*](#)30481 **CHANGE HISTORY**

30482 First released in Issue 1. Derived from Issue 1 of the SVID.

30483 **Issue 5**30484 The DESCRIPTION is updated to indicate how an application should check for an error. This  
30485 text was previously published in the APPLICATION USAGE section.

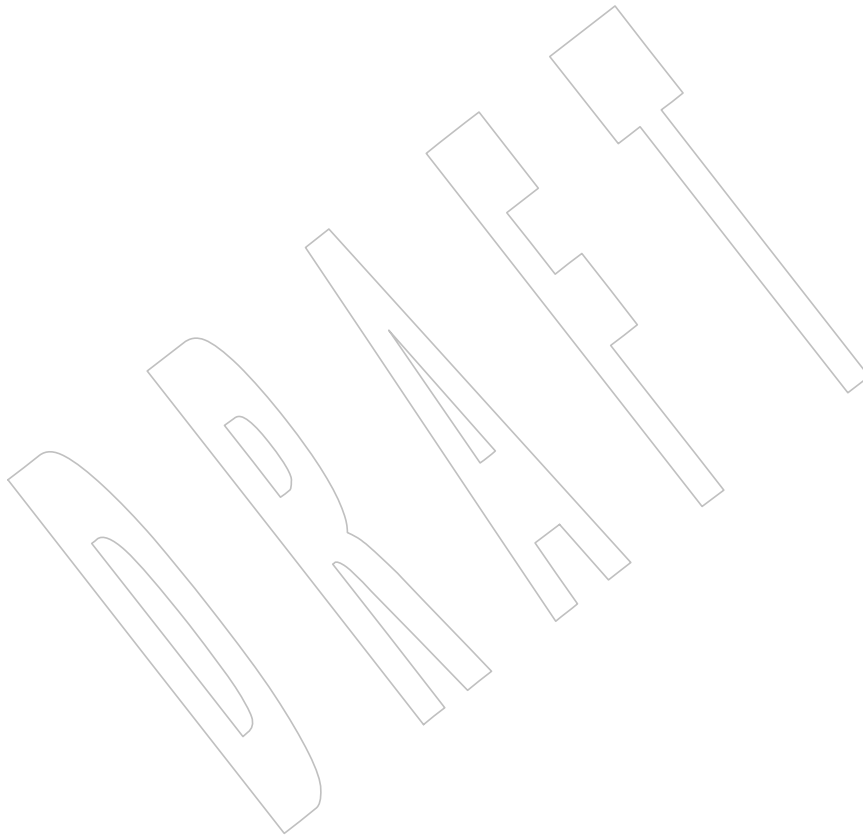
30486  
30487  
30488  
30489  
30490  
30491  
30492

**Issue 6**

The *frexpf()* and *frexpl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.



30493 **NAME**  
 30494 `fscanf, scanf, sscanf` — convert formatted input

30495 **SYNOPSIS**

```
30496 #include <stdio.h>
30497
30497 int fscanf(FILE *restrict stream, const char *restrict format, ...);
30498 int scanf(const char *restrict format, ...);
30499 int sscanf(const char *restrict s, const char *restrict format, ...);
```

30500 **DESCRIPTION**

30501 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30502 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30503 volume of POSIX.1-200x defers to the ISO C standard.

30504 The `fscanf()` function shall read from the named input *stream*. The `scanf()` function shall read  
 30505 from the standard input stream *stdin*. The `sscanf()` function shall read from the string *s*. Each  
 30506 function reads bytes, interprets them according to a *format*, and stores the results in its  
 30507 arguments. Each expects, as arguments, a control string *format* described below, and a set of  
 30508 *pointer* arguments indicating where the converted input should be stored. The result is  
 30509 undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while  
 30510 arguments remain, the excess arguments shall be evaluated but otherwise ignored.

30511 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 30512 to the next unused argument. In this case, the conversion specifier character `%` (see below) is  
 30513 replaced by the sequence `"%n$"`, where *n* is a decimal integer in the range `[1, {NL_ARGMAX}]`.  
 30514 This feature provides for the definition of format strings that select arguments in an order  
 30515 appropriate to specific languages. In format strings containing the `"%n$"` form of conversion  
 30516 specifications, it is unspecified whether numbered arguments in the argument list can be  
 30517 referenced from the format string more than once.

30518 The *format* can contain either form of a conversion specification—that is, `%` or `"%n$"`—but the  
 30519 two forms cannot be mixed within a single *format* string. The only exception to this is that `%%` or  
 30520 `%*` can be mixed with the `"%n$"` form. When numbered argument specifications are used,  
 30521 specifying the *N*th argument requires that all the leading arguments, from the first to the  
 30522 (*N*–1)th, are pointers.

30523 The `fscanf()` function in all its forms shall allow detection of a language-dependent radix  
 30524 character in the input string. The radix character is defined in the locale of the process (category  
 30525 `LC_NUMERIC`). In the POSIX locale, or in a locale where the radix character is not defined, the  
 30526 radix character shall default to a period (`'.'`).

30527 The *format* is a character string, beginning and ending in its initial shift state, if any, composed  
 30528 of zero or more directives. Each directive is composed of one of the following: one or more  
 30529 white-space characters (`<space>`s, `<tab>`s, `<newline>`s, `<vertical-tab>`s, or `<form-feed>`s); an  
 30530 ordinary character (neither `'%'` nor a white-space character); or a conversion specification. Each  
 30531 CX conversion specification is introduced by the character `'%'` or the character sequence `"%n$"`,  
 30532 after which the following appear in sequence:

- 30533 • An optional assignment-suppressing character `'*'`.
- 30534 • An optional non-zero decimal integer that specifies the maximum field width.
- 30535 CX • An optional assignment-allocation character `'m'`.
- 30536 • An option length modifier that specifies the size of the receiving object.

- 30537                   • A *conversion specifier* character that specifies the type of conversion to be applied. The valid  
30538                   conversion specifiers are described below.
- 30539                   The *fscanf()* functions shall execute each directive of the format in turn. If a directive fails, as  
30540                   detailed below, the function shall return. Failures are described as input failures (due to the  
30541                   unavailability of input bytes) or matching failures (due to inappropriate input).
- 30542                   A directive composed of one or more white-space characters shall be executed by reading input  
30543                   until no more valid input can be read, or up to the first byte which is not a white-space character,  
30544                   which remains unread.
- 30545                   A directive that is an ordinary character shall be executed as follows: the next byte shall be read  
30546                   from the input and compared with the byte that comprises the directive; if the comparison  
30547                   shows that they are not equivalent, the directive shall fail, and the differing and subsequent  
30548                   bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a  
30549                   character from being read, the directive shall fail.
- 30550                   A directive that is a conversion specification defines a set of matching input sequences, as  
30551                   described below for each conversion character. A conversion specification shall be executed in  
30552                   the following steps.
- 30553                   Input white-space characters (as specified by *isspace()*) shall be skipped, unless the conversion  
30554                   specification includes a *l*, *c*, *C*, or *n* conversion specifier.
- 30555                   An item shall be read from the input, unless the conversion specification includes an *n*  
30556                   conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to  
30557                   any specified maximum field width, which may be measured in characters or bytes dependent  
30558                   on the conversion specifier) which is an initial subsequence of a matching sequence. The first  
30559                   byte, if any, after the input item shall remain unread. If the length of the input item is 0, the  
30560                   execution of the conversion specification shall fail; this condition is a matching failure, unless  
30561                   end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is  
30562                   an input failure.
- 30563                   Except in the case of a *%* conversion specifier, the input item (or, in the case of a *%n* conversion  
30564                   specification, the count of input bytes) shall be converted to a type appropriate to the conversion  
30565                   character. If the input item is not a matching sequence, the execution of the conversion  
30566                   specification fails; this condition is a matching failure. Unless assignment suppression was  
30567                   indicated by a *'\*'*, the result of the conversion shall be placed in the object pointed to by the  
30568                   first argument following the *format* argument that has not already received a conversion result if  
30569                   the conversion specification is introduced by *%*, or in the *n*th argument if introduced by the  
30570                   character sequence "*%n\$*". If this object does not have an appropriate type, or if the result of the  
30571                   conversion cannot be represented in the space provided, the behavior is undefined.
- 30572                   CX       The *%s*, *%S*, and *%[* conversion specifiers shall accept an optional assignment-allocation  
30573                   character *'m'*, which shall cause a memory buffer to be allocated to hold the string converted  
30574                   including a terminating null character. In such a case, the argument corresponding to the  
30575                   conversion specifier should be a reference to a pointer variable that will receive a pointer to the  
30576                   allocated buffer. The system shall allocate a buffer as if *malloc()* had been called. The application  
30577                   shall be responsible for freeing the memory after usage. If there is insufficient memory to  
30578                   allocate a buffer, the function shall set *errno* to [ENOMEM] and a conversion error shall result. In  
30579                   that case, any memory successfully allocated for other parameters using assignment-allocation  
30580                   character *'m'* by this call shall be freed.                   +
- 30581                   The length modifiers and their meanings are:
- 30582                   hh       Specifies that a following *d*, *i*, *o*, *u*, *x*, *X*, or *n* conversion specifier applies to an  
30583                   argument with type pointer to **signed char** or **unsigned char**.

**fscanf()**

|       |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
|-------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|
| 30584 | h            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>short</b> or <b>unsigned short</b> .                                                                                                                                                                                                                                                                                                                                                                                                  |   |
| 30585 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30586 | l (ell)      | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>long</b> or <b>unsigned long</b> ; that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to <b>double</b> ; or that a following c, s, or [ conversion specifier applies to an argument with type pointer to <b>wchar_t</b> . If the 'm' assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to <b>wchar_t</b> . | + |
| 30587 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30588 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30589 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30590 | CX           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30591 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30592 | ll (ell-ell) | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>long long</b> or <b>unsigned long long</b> .                                                                                                                                                                                                                                                                                                                                                                                          |   |
| 30593 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30594 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30595 | j            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>intmax_t</b> or <b>uintmax_t</b> .                                                                                                                                                                                                                                                                                                                                                                                                    |   |
| 30596 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30597 | z            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>size_t</b> or the corresponding signed integer type.                                                                                                                                                                                                                                                                                                                                                                                  |   |
| 30598 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30599 | t            | Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to <b>ptrdiff_t</b> or the corresponding <b>unsigned</b> type.                                                                                                                                                                                                                                                                                                                                                                              |   |
| 30600 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30601 | L            | Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to <b>long double</b> .                                                                                                                                                                                                                                                                                                                                                                                                                  |   |
| 30602 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30603 |              | If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                        |   |
| 30604 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30605 |              | The following conversion specifiers are valid:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |   |
| 30606 | d            | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                                                                                                                                                                                                                                 |   |
| 30607 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30608 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30609 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30610 | i            | Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>strtol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>int</b> .                                                                                                                                                                                                                                                                    |   |
| 30611 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30612 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30613 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30614 | o            | Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                              |   |
| 30615 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30616 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30617 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30618 | u            | Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                           |   |
| 30619 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30620 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30621 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30622 | x            | Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>strtoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>unsigned</b> .                                                                                                                                                                                                                                       |   |
| 30623 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30624 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30625 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30626 | a, e, f, g   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30627 |              | Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of <i>strtod()</i> . In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to <b>float</b> .                                                                                                                                                                                                                                                                     |   |
| 30628 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30629 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |
| 30630 |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |

|       |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
|-------|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| 30631 |    | If the <i>fprintf()</i> family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the <i>fscanf()</i> family of functions shall recognize them as input.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                 |
| 30632 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30633 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30634 | s  | Matches a sequence of bytes that are not white-space characters. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null character code, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <br> <br> <br>+ |
| 30635 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30636 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30637 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30638 | CX |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30639 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30640 |    | If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .                                                                                                                                                                                                                           | <br> <br> <br>+ |
| 30641 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30642 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30643 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30644 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30645 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30646 | CX |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30647 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30648 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30649 | [  | Matches a non-empty sequence of bytes from a set of expected bytes (the <i>scanset</i> ). The normal skip over white-space characters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> , or <b>unsigned char</b> large enough to accept the sequence and a terminating null byte, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                                                                                                                                                                                                                                                           | <br> <br> <br>+ |
| 30650 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30651 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30652 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30653 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30654 | CX |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30655 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30656 |    | If an l (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence shall be converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> .                                                                                                                                                                                                           | <br> <br> <br>+ |
| 30657 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30658 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30659 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30660 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30661 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30662 | CX |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30663 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | +               |
| 30664 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30665 |    | The conversion specification includes all subsequent bytes in the <i>format</i> string up to and including the matching right square bracket (']'). The bytes between the square brackets (the <i>scanlist</i> ) comprise the <i>scanset</i> , unless the byte after the left square bracket is a circumflex ('^'), in which case the <i>scanset</i> contains all bytes that do not appear in the <i>scanlist</i> between the circumflex and the right square bracket. If the conversion specification begins with "[ ]" or "[^]", the right square bracket is included in the <i>scanlist</i> and the next right square bracket is the matching right square bracket that ends the conversion specification; otherwise, the first right square bracket is the one that ends the conversion specification. If a '-' is in the <i>scanlist</i> and is not the first character, nor the second where the first character is a '^', nor the last character, the behavior is implementation-defined. |                 |
| 30666 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30667 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30668 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30669 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30670 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30671 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30672 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30673 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30674 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30675 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30676 | c  | Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space characters shall be suppressed in this case. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of <b>char</b> , <b>signed char</b> ,                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <br> <br> <br>  |
| 30677 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30678 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30679 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |
| 30680 |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                 |

|       |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 30681 | CX  | or <b>unsigned char</b> large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>char</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 30682 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30683 |     | If an <b>l</b> (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the <i>mbrtowc()</i> function, with the conversion state described by an <b>mbstate_t</b> object initialized to zero before the first character is converted. No null wide character is added. If the 'm' assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of <b>wchar_t</b> large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a <b>wchar_t</b> . |
| 30684 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30685 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30686 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30687 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30688 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30689 | CX  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30690 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30691 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30692 | p   | Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the %p conversion specification of the corresponding <i>fprintf()</i> functions. The application shall ensure that the corresponding argument is a pointer to a pointer to <b>void</b> . The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the %p conversion specification is undefined.                                                                                                                                                                         |
| 30693 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30694 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30695 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30696 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30697 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30698 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30699 | n   | No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the <i>fscanf()</i> functions. Execution of a %n conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.                                                                                                                                                                                                      |
| 30700 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30701 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30702 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30703 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30704 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30705 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30706 | XSI | <b>C</b> Equivalent to <b>lc</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 30707 | XSI | <b>S</b> Equivalent to <b>ls</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 30708 | %   | Matches a single '%' character; no conversion or assignment occurs. The complete conversion specification shall be %%.<br>If a conversion specification is invalid, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 30709 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30710 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30711 |     | The conversion specifiers <b>A</b> , <b>E</b> , <b>F</b> , <b>G</b> , and <b>X</b> are also valid and shall be equivalent to <b>a</b> , <b>e</b> , <b>f</b> , <b>g</b> , and <b>x</b> , respectively.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 30712 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30713 |     | If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for %n) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.                                                                                                                                                                                                       |
| 30714 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30715 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30716 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30717 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30718 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30719 |     | Reaching the end of the string in <i>sscanf()</i> shall be equivalent to encountering end-of-file for <i>fscanf()</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 30720 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30721 |     | If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including <newline>s) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the %n conversion specification.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 30722 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30723 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30724 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30725 | CX  | The <i>fscanf()</i> and <i>scanf()</i> functions may mark the last data access timestamp of the file associated with <i>stream</i> for update. The last data access timestamp shall be marked for update by the first successful execution of <i>fgetc()</i> , <i>fgets()</i> , <i>fread()</i> , <i>getc()</i> , <i>getchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>scanf()</i> using <i>stream</i> that returns data not supplied by a prior call to <i>ungetc()</i> .                                                                                                                                                                                                                                                                                                 |
| 30726 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30727 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 30728 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |



**RETURN VALUE**

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first matching failure or conversion, EOF shall be returned. If a read error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be set to indicate the error.

**ERRORS**

For the conditions under which the *fscanf()* functions fail and may fail, refer to *fgetc()* or *fgetwc()*.

In addition, the *fscanf()* functions shall fail if:

[ENOMEM]      Insufficient storage space is available.

The *fscanf()* functions may fail if:

[EILSEQ]      Input byte sequence does not form a valid character.

[EINVAL]      There are insufficient arguments.

**EXAMPLES**

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string "Hamster".

The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to *getchar()* shall return the character 'a'.

**Reading Data into an Array**

The following call uses *fscanf()* to read three floating-point numbers from standard input into the *input* array.

```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

**APPLICATION USAGE**

If the application calling *fscanf()* has any objects of type **wint\_t** or **wchar\_t**, it must also include the **<wchar.h>** header to have these objects defined.

**RATIONALE**

This function is aligned with the ISO/IEC 9899:1999 standard, and in doing so a few "obvious" things were not included. Specifically, the set of characters allowed in a scanset is limited to single-byte characters. In other similar places, multi-byte characters have been permitted, but for alignment with the ISO/IEC 9899:1999 standard, it has not been done here. Applications needing this could use the corresponding wide-character functions to achieve the desired results.

|       |                                                                                                                                 |   |
|-------|---------------------------------------------------------------------------------------------------------------------------------|---|
| 30772 | <b>FUTURE DIRECTIONS</b>                                                                                                        |   |
| 30773 | None.                                                                                                                           |   |
| 30774 | <b>SEE ALSO</b>                                                                                                                 |   |
| 30775 | <i>fprintf()</i> , <i>getc()</i> , <i>setlocale()</i> , <i>strtod()</i> , <i>strtol()</i> , <i>strtoul()</i> , <i>wcrtomb()</i> | - |
| 30776 | XBD Chapter 7 (on page 121), <i>&lt;langinfo.h&gt;</i> , <i>&lt;stdio.h&gt;</i> , <i>&lt;wchar.h&gt;</i>                        |   |
| 30777 | <b>CHANGE HISTORY</b>                                                                                                           |   |
| 30778 | First released in Issue 1. Derived from Issue 1 of the SVID.                                                                    |   |
| 30779 | <b>Issue 5</b>                                                                                                                  |   |
| 30780 | Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the l (ell) qualifier is                                     |   |
| 30781 | now defined for the c, s, and [ conversion specifiers.                                                                          |   |
| 30782 | The DESCRIPTION is updated to indicate that if infinity and NaN can be generated by the                                         |   |
| 30783 | <i>fprintf()</i> family of functions, then they are recognized by the <i>fscanf()</i> family.                                   |   |
| 30784 | <b>Issue 6</b>                                                                                                                  |   |
| 30785 | The Open Group Corrigenda U021/7 and U028/10 are applied. These correct several                                                 |   |
| 30786 | occurrences of “characters” in the text which have been replaced with the term “bytes”.                                         |   |
| 30787 | The normative text is updated to avoid use of the term “must” for application requirements.                                     |   |
| 30788 | The following changes are made for alignment with the ISO/IEC 9899:1999 standard:                                               |   |
| 30789 | • The prototypes for <i>fscanf()</i> , <i>scanf()</i> , and <i>sscanf()</i> are updated.                                        |   |
| 30790 | • The DESCRIPTION is updated.                                                                                                   |   |
| 30791 | • The hh, ll, j, t, and z length modifiers are added.                                                                           |   |
| 30792 | • The a, A, and F conversion characters are added.                                                                              |   |
| 30793 | The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion                                              |   |
| 30794 | specification” consistently.                                                                                                    |   |
| 30795 | <b>Issue 7</b>                                                                                                                  |   |
| 30796 | SD5-XSH-ERN-9 is applied, correcting <i>fscanf()</i> to <i>scanf()</i> in the DESCRIPTION.                                      |   |
| 30797 | SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ‘m’.                                                     | + |
| 30798 | Functionality relating to the %n\$ form of conversion specification is moved from the XSI option                                |   |
| 30799 | to the Base.                                                                                                                    |   |
| 30800 | Changes are made related to support for finegrained timestamps.                                                                 | + |

30801 **NAME**  
 30802 `fseek, fseeko` — reposition a file-position indicator in a stream

30803 **SYNOPSIS**  
 30804 `#include <stdio.h>`  
 30805 `int fseek(FILE *stream, long offset, int whence);`  
 30806 CX `int fseeko(FILE *stream, off_t offset, int whence);`

### 30807 DESCRIPTION

30808 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 30809 conflict between the requirements described here and the ISO C standard is unintentional. This  
 30810 volume of POSIX.1-200x defers to the ISO C standard.

30811 The `fseek()` function shall set the file-position indicator for the stream pointed to by `stream`. If a  
 30812 read or write error occurs, the error indicator for the stream shall be set and `fseek()` fails.

30813 The new position, measured in bytes from the beginning of the file, shall be obtained by adding  
 30814 `offset` to the position specified by `whence`. The specified point is the beginning of the file for  
 30815 `SEEK_SET`, the current value of the file-position indicator for `SEEK_CUR`, or end-of-file for  
 30816 `SEEK_END`.

30817 If the stream is to be used with wide-character input/output functions, the application shall  
 30818 ensure that `offset` is either 0 or a value returned by an earlier call to `ftell()` on the same stream and  
 30819 `whence` is `SEEK_SET`.

30820 A successful call to `fseek()` shall clear the end-of-file indicator for the stream and undo any effects  
 30821 of `ungetc()` and `ungetwc()` on the same stream. After an `fseek()` call, the next operation on an  
 30822 update stream may be either input or output.

30823 CX If the most recent operation, other than `ftell()`, on a given stream is `fflush()`, the file offset in the  
 30824 underlying open file description shall be adjusted to reflect the location specified by `fseek()`.

30825 The `fseek()` function shall allow the file-position indicator to be set beyond the end of existing  
 30826 data in the file. If data is later written at this point, subsequent reads of data in the gap shall  
 30827 return bytes with the value 0 until data is actually written into the gap.

30828 The behavior of `fseek()` on devices which are incapable of seeking is implementation-defined.  
 30829 The value of the file offset associated with such a device is undefined.

30830 If the stream is writable and buffered data had not been written to the underlying file, `fseek()`  
 30831 shall cause the unwritten data to be written to the file and shall mark the last data modification  
 30832 and last file status change timestamps of the file for update.

30833 In a locale with state-dependent encoding, whether `fseek()` restores the stream's shift state is  
 30834 implementation-defined.

30835 The `fseeko()` function shall be equivalent to the `fseek()` function except that the `offset` argument is  
 30836 of type `off_t`.

### 30837 RETURN VALUE

30838 CX The `fseek()` and `fseeko()` functions shall return 0 if they succeed.

30839 CX Otherwise, they shall return `-1` and set `errno` to indicate the error.

**fseek()****ERRORS**

- 30840
- 30841 CX The *fseek()* and *fseeko()* functions shall fail if, either the *stream* is unbuffered or the *stream's*
- 30842 buffer needed to be flushed, and the call to *fseek()* or *fseeko()* causes an underlying *lseek()* or
- 30843 *write()* to be invoked, and:
- 30844 CX [EAGAIN] The O\_NONBLOCK flag is set for the file descriptor and the thread would be
- 30845 delayed in the write operation.
- 30846 CX [EBADF] The file descriptor underlying the stream file is not open for writing or the
- 30847 stream's buffer needed to be flushed and the file is not open.
- 30848 CX [EFBIG] An attempt was made to write a file that exceeds the maximum file size.
- 30849 XSI [EFBIG] An attempt was made to write a file that exceeds the file size limit of the
- 30850 process.
- 30851 CX [EFBIG] The file is a regular file and an attempt was made to write at or beyond the
- 30852 offset maximum associated with the corresponding stream.
- 30853 CX [EINTR] The write operation was terminated due to the receipt of a signal, and no data
- 30854 was transferred.
- 30855 CX [EINVAL] The *whence* argument is invalid. The resulting file-position indicator would be
- 30856 set to a negative value.
- 30857 CX [EIO] A physical I/O error has occurred, or the process is a member of a background
- 30858 process group attempting to perform a *write()* to its controlling terminal,
- 30859 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the
- 30860 process group of the process is orphaned. This error may also be returned
- 30861 under implementation-defined conditions.
- 30862 CX [ENOSPC] There was no free space remaining on the device containing the file.
- 30863 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the
- 30864 capabilities of the device.
- 30865 CX [EOVERFLOW] For *fseek()*, the resulting file offset would be a value which cannot be
- 30866 represented correctly in an object of type **long**.
- 30867 CX [EOVERFLOW] For *fseeko()*, the resulting file offset would be a value which cannot be
- 30868 represented correctly in an object of type **off\_t**.
- 30869 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading
- 30870 by any process; a SIGPIPE signal shall also be sent to the thread.
- 30871 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO***fdopen()*, *fsetpos()*, *ftell()*, *getrlimit()*, *lseek()*, *rewind()*, *ulimit*, *ungetc()*, *write*

XBD &lt;stdio.h&gt;

30883  
30884  
30885  
30886  
30887  
30888  
30889  
30890  
30891  
30892  
30893  
30894  
30895  
30896  
30897  
30898  
30899  
30900  
30901  
30902  
30903  
30904  
30905  
30906

**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Large File Summit extensions are added.

**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *fseeko()* function is added.
- The [EFBIG], [EOVERFLOW], and [ENXIO] mandatory error conditions are added.

The following change is incorporated for alignment with the FIPS requirements:

- The [EINTR] error is no longer an indication that the implementation does not report partial transfers.

The normative text is updated to avoid use of the term “must” for application requirements.

The DESCRIPTION is updated to explicitly state that *fseek()* sets the file-position indicator, and then on error the error indicator is set and *fseek()* fails. This is for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/42 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

**Issue 7**

Changes are made related to support for finegrained timestamps.

+

30907 **NAME**30908 `fsetpos` — set current file position30909 **SYNOPSIS**30910 `#include <stdio.h>`30911 `int fsetpos(FILE *stream, const fpos_t *pos);`30912 **DESCRIPTION**30913 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
30914 conflict between the requirements described here and the ISO C standard is unintentional. This  
30915 volume of POSIX.1-200x defers to the ISO C standard.30916 The `fsetpos()` function shall set the file position and state indicators for the stream pointed to by  
30917 `stream` according to the value of the object pointed to by `pos`, which the application shall ensure is  
30918 a value obtained from an earlier call to `fgetpos()` on the same stream. If a read or write error  
30919 occurs, the error indicator for the stream shall be set and `fsetpos()` fails.30920 A successful call to the `fsetpos()` function shall clear the end-of-file indicator for the stream and  
30921 undo any effects of `ungetc()` on the same stream. After an `fsetpos()` call, the next operation on an  
30922 update stream may be either input or output.30923 CX The behavior of `fsetpos()` on devices which are incapable of seeking is implementation-defined.  
30924 The value of the file offset associated with such a device is undefined.30925 **RETURN VALUE**30926 The `fsetpos()` function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and  
30927 set `errno` to indicate the error.30928 **ERRORS**30929 CX The `fsetpos()` function shall fail if, either the `stream` is unbuffered or the `stream`'s buffer needed to  
30930 be flushed, and the call to `fsetpos()` causes an underlying `lseek()` or `write()` to be invoked, and:30931 CX **[EAGAIN]** The `O_NONBLOCK` flag is set for the file descriptor and the thread would be  
30932 delayed in the write operation.30933 CX **[EBADF]** The file descriptor underlying the stream file is not open for writing or the  
30934 stream's buffer needed to be flushed and the file is not open.30935 CX **[EFBIG]** An attempt was made to write a file that exceeds the maximum file size.30936 XSI **[EFBIG]** An attempt was made to write a file that exceeds the file size limit of the  
30937 process.30938 CX **[EFBIG]** The file is a regular file and an attempt was made to write at or beyond the  
30939 offset maximum associated with the corresponding stream.30940 CX **[EINTR]** The write operation was terminated due to the receipt of a signal, and no data  
30941 was transferred.30942 CX **[EIO]** A physical I/O error has occurred, or the process is a member of a background  
30943 process group attempting to perform a `write()` to its controlling terminal,  
30944 TOSTOP is set, the process is neither ignoring nor blocking SIGTTOU, and the  
30945 process group of the process is orphaned. This error may also be returned  
30946 under implementation-defined conditions.30947 CX **[ENOSPC]** There was no free space remaining on the device containing the file.

- 30948 CX [ENXIO] A request was made of a nonexistent device, or the request was outside the  
30949 capabilities of the device.
- 30950 CX [EPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.
- 30951 CX [EPIPE] An attempt was made to write to a pipe or FIFO that is not open for reading  
30952 by any process; a SIGPIPE signal shall also be sent to the thread.

**EXAMPLES**

None.

**APPLICATION USAGE**

None.

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO***fopen()*, *ftell()*, *lseek()*, *rewind()*, *ungetc()*, *write*

XBD &lt;stdio.h&gt;

**CHANGE HISTORY**

First released in Issue 4. Derived from the ISO C standard.

**Issue 6**

Extensions beyond the ISO C standard are marked.

An additional [ESPIPE] error condition is added for sockets.

The normative text is updated to avoid use of the term “must” for application requirements.

The DESCRIPTION is updated to clarify that the error indicator is set for the stream on a read or write error. This is for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/21 is applied, deleting an erroneous [EINVAL] error case from the ERRORS section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/43 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

30977 **NAME**30978 `fstat` — get file status30979 **SYNOPSIS**30980 `#include <sys/stat.h>`30981 `int fstat(int fildev, struct stat *buf);`30982 **DESCRIPTION**30983 The `fstat()` function shall obtain information about an open file associated with the file  
30984 descriptor *fildev*, and shall write it to the area pointed to by *buf*.30985 SHM If *fildev* references a shared memory object, the implementation shall update in the **stat** structure  
30986 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
30987 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
30988 valid. The implementation may update other fields and flags.30989 TYM If *fildev* references a typed memory object, the implementation shall update in the **stat** structure  
30990 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
30991 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
30992 valid. The implementation may update other fields and flags.30993 The *buf* argument is a pointer to a **stat** structure, as defined in `<sys/stat.h>`, into which  
30994 information is placed concerning the file.30995 For all other file types defined in this volume of POSIX.1-200x, the structure members *st\_mode*,  
30996 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
30997 value of the *st\_nlink* member shall be set to the number of links to the file.30998 An implementation that provides additional or alternative file access control mechanisms may,  
30999 under implementation-defined conditions, cause `fstat()` to fail.31000 The `fstat()` function shall update any time-related fields (as described in XBD [Section 4.8](#), on  
31001 page 97), before writing into the **stat** structure.31002 **RETURN VALUE**31003 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to  
31004 indicate the error.31005 **ERRORS**31006 The `fstat()` function shall fail if:31007 [EBADF] The *fildev* argument is not a valid file descriptor.

31008 [EIO] An I/O error occurred while reading from the file system.

31009 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
31010 serial number cannot be represented correctly in the structure pointed to by  
31011 *buf*.31012 The `fstat()` function may fail if:31013 [EOVERFLOW] One of the values is too large to store into the structure pointed to by the *buf*  
31014 argument.



31015 **EXAMPLES**31016 **Obtaining File Status Information**

31017 The following example shows how to obtain file status information for a file named  
 31018 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The  
 31019 **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file  
 31020 descriptor *filides*.

```
31021 #include <sys/types.h>
31022 #include <sys/stat.h>
31023 #include <fcntl.h>

31024 struct stat buffer;
31025 int status;
31026 ...
31027 filides = open("/home/cnd/mod1", O_RDWR);
31028 status = fstat(filides, &buffer);
```

31029 **APPLICATION USAGE**

31030 None.

31031 **RATIONALE**

31032 None.

31033 **FUTURE DIRECTIONS**

31034 None.

31035 **SEE ALSO**

31036 [fstatat\(\)](#)

31037 XBD Section 4.8 (on page 97), [<sys/stat.h>](#), [<sys/types.h>](#)

31038 **CHANGE HISTORY**

31039 First released in Issue 1. Derived from Issue 1 of the SVID.

31040 **Issue 5**

31041 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

31042 Large File Summit extensions are added.

31043 **Issue 6**

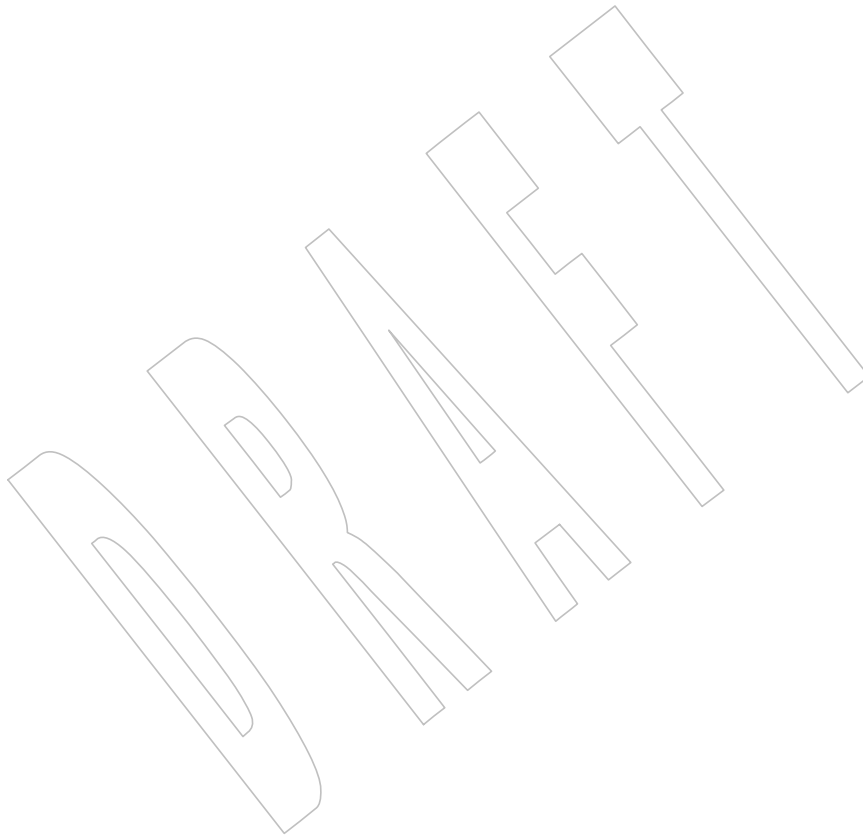
31044 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

31045 The following new requirements on POSIX implementations derive from alignment with the  
 31046 Single UNIX Specification:

- 31047 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was  
 31048 required for conforming implementations of previous POSIX specifications, it was not  
 31049 required for UNIX applications.
- 31050 • The [EIO] mandatory error condition is added.
- 31051 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
 31052 files.
- 31053 • The [EOVERFLOW] optional error condition is added.

31054 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that  
 31055 shared memory object semantics apply to typed memory objects.

- 31056 **Issue 7**
- 31057 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify which file types *st\_nlink*
- 31058 applies to.
- 31059 Changes are made related to support for finegrained timestamps. +



31060 **NAME**  
 31061 fstatat, lstat, stat — get file status

31062 **SYNOPSIS**  
 31063 #include <sys/stat.h>  
 31064 int fstatat(int fd, const char \*restrict path,  
 31065 struct stat \*restrict buf, int flag);  
 31066 int lstat(const char \*restrict path, struct stat \*restrict buf);  
 31067 int stat(const char \*restrict path, struct stat \*restrict buf);

31068 **DESCRIPTION**  
 31069 The *stat()* function shall obtain information about the named file and write it to the area pointed  
 31070 to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or  
 31071 execute permission of the named file is not required. An implementation that provides  
 31072 additional or alternate file access control mechanisms may, under implementation-defined  
 31073 conditions, cause *stat()* to fail. In particular, the system may deny the existence of the file  
 31074 specified by *path*.

31075 If the named file is a symbolic link, the *stat()* function shall continue pathname resolution using  
 31076 the contents of the symbolic link, and shall return information pertaining to the resulting file if  
 31077 the file exists.

31078 The *buf* argument is a pointer to a **stat** structure, as defined in the <sys/stat.h> header, into  
 31079 which information is placed concerning the file.

31080 The *stat()* function shall update any time-related fields (as described in XBD Section 4.8, on page  
 31081 97), before writing into the **stat** structure.

31082 SHM If the named file is a shared memory object, the implementation shall update in the **stat** structure  
 31083 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 31084 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 31085 valid. The implementation may update other fields and flags.

31086 TYM If the named file is a typed memory object, the implementation shall update in the **stat** structure  
 31087 pointed to by the *buf* argument the *st\_uid*, *st\_gid*, *st\_size*, and *st\_mode* fields, and only the  
 31088 S\_IRUSR, S\_IWUSR, S\_IRGRP, S\_IWGRP, S\_IROTH, and S\_IWOTH file permission bits need be  
 31089 valid. The implementation may update other fields and flags.

31090 For all other file types defined in this volume of POSIX.1-200x, the structure members *st\_mode*,  
 31091 *st\_ino*, *st\_dev*, *st\_uid*, *st\_gid*, *st\_atim*, *st\_ctim*, and *st\_mtim* shall have meaningful values and the  
 31092 value of the member *st\_nlink* shall be set to the number of links to the file.

31093 The *lstat()* function shall be equivalent to *stat()*, except when *path* refers to a symbolic link. In  
 31094 that case *lstat()* shall return information about the link, while *stat()* shall return information  
 31095 about the file the link references.

31096 For symbolic links, the *st\_mode* member shall contain meaningful information when used with  
 31097 the file type macros, and the *st\_size* member shall contain the length of the pathname contained  
 31098 in the symbolic link. File mode bits and the contents of the remaining members of the **stat**  
 31099 structure are unspecified. The value returned in the *st\_size* member is the length of the contents  
 31100 of the symbolic link, and does not count any trailing null.

31101 The *fstatat()* function shall be equivalent to the *stat()* or *lstat()* function, depending on the value  
 31102 of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status  
 31103 shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead  
 31104 of the current working directory. It is unspecified whether directory searches are permitted  
 31105 based on whether the file was opened with search permission or on the current permissions of

- 31106 the directory underlying the file descriptor.
- 31107 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined  
31108 in `<fcntl.h>`:
- 31109 AT\_SYMLINK\_NOFOLLOW  
31110 If *path* names a symbolic link, the status of the symbolic link is returned.
- 31111 If *fstatat()* is passed the special value AT\_FDCWD in the *fd* parameter, the current working  
31112 directory is used and the behavior shall be identical to a call to *stat()* or *lstat()* respectively,  
31113 depending on whether or not the AT\_SYMLINK\_NOFOLLOW bit is set in *flag*.
- 31114 **RETURN VALUE**
- 31115 Upon successful completion, these functions shall return 0. Otherwise, these functions shall  
31116 return -1 and set *errno* to indicate the error.
- 31117 **ERRORS**
- 31118 These functions shall fail if:
- 31119 [EACCES] Search permission is denied for a component of the path prefix.
- 31120 [EIO] An error occurred while reading from the file system.
- 31121 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
31122 argument.
- 31123 [ENAMETOOLONG]  
31124 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
31125 component is longer than {NAME\_MAX}.
- 31126 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 31127 [ENOTDIR] A component of the path prefix is not a directory.
- 31128 [EOVERFLOW] The file size in bytes or the number of blocks allocated to the file or the file  
31129 serial number cannot be represented correctly in the structure pointed to by  
31130 *buf*.
- 31131 The *fstatat()* function shall fail if:
- 31132 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is  
31133 neither AT\_FDCWD nor a valid file descriptor open for reading.
- 31134 These functions may fail if:
- 31135 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
31136 resolution of the *path* argument.
- 31137 [ENAMETOOLONG]  
31138 As a result of encountering a symbolic link in resolution of the *path* argument,  
31139 the length of the substituted pathname string exceeded {PATH\_MAX}.
- 31140 [EOVERFLOW] A value to be stored would overflow one of the members of the **stat** structure.
- 31141 The *fstatat()* function may fail if:
- 31142 [EINVAL] The value of the *flag* argument is not valid.
- 31143 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
31144 file descriptor associated with a directory.

31145 **EXAMPLES**31146 **Obtaining File Status Information**

31147 The following example shows how to obtain file status information for a file named  
31148 **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
31149 #include <sys/types.h>
31150 #include <sys/stat.h>
31151 #include <fcntl.h>
31152
31153 struct stat buffer;
31154 int status;
31155 ...
31156 status = stat("/home/cnd/mod1", &buffer);
```

31156 **Getting Directory Information**

31157 The following example fragment gets status information for each entry in a directory. The call to  
31158 the *stat()* function stores file information in the **stat** structure pointed to by *statbuf*. The lines  
31159 that follow the *stat()* call format the fields in the **stat** structure for presentation to the user of the  
31160 program.

```
31161 #include <sys/types.h>
31162 #include <sys/stat.h>
31163 #include <dirent.h>
31164 #include <pwd.h>
31165 #include <grp.h>
31166 #include <time.h>
31167 #include <locale.h>
31168 #include <langinfo.h>
31169 #include <stdio.h>
31170 #include <stdint.h>
31171
31172 struct dirent *dp;
31173 struct stat statbuf;
31174 struct passwd *pwd;
31175 struct group *grp;
31176 struct tm *tm;
31177 char datestring[256];
31178 ...
31179 /* Loop through directory entries. */
31180 while ((dp = readdir(dir)) != NULL) {
31181 /* Get entry's information. */
31182 if (stat(dp->d_name, &statbuf) == -1)
31183 continue;
31184
31185 /* Print out type, permissions, and number of links. */
31186 printf("%10.10s", sperm (statbuf.st_mode));
31187 printf("%4d", statbuf.st_nlink);
31188
31189 /* Print out owner's name if it is found using getpwuid(). */
31190 if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
31191 printf(" %-8.8s", pwd->pw_name);
31192 else
31193 printf(" %-8d", statbuf.st_uid);
31194
31195 /* Print out group name if it is found using getgrgid(). */
```

```

31192 if ((grp = getgrgid(statbuf.st_gid)) != NULL)
31193 printf(" %-8.8s", grp->gr_name);
31194 else
31195 printf(" %-8d", statbuf.st_gid);
31196
31196 /* Print size of file. */
31197 printf(" %9jd", (intmax_t)statbuf.st_size);
31198
31198 tm = localtime(&statbuf.st_mtime);
31199
31199 /* Get localized date string. */
31200 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
31201 printf(" %s %s\n", datestring, dp->d_name);
31202 }

```

### 31203 Obtaining Symbolic Link Status Information

31204 The following example shows how to obtain status information for a symbolic link named  
31205 **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path*  
31206 argument specified the filename for the file pointed to by the symbolic link (**/home/cnd/mod1**),  
31207 the results of calling the function would be the same as those returned by a call to the *stat()*  
31208 function.

```

31209 #include <sys/stat.h>
31210 struct stat buffer;
31211 int status;
31212 ...
31213 status = lstat("/modules/pass1", &buffer);

```

### 31214 APPLICATION USAGE

31215 None.

### 31216 RATIONALE

31217 The intent of the paragraph describing “additional or alternate file access control mechanisms”  
31218 is to allow a secure implementation where a process with a label that does not dominate the  
31219 file’s label cannot perform a *stat()* function. This is not related to read permission; a process with  
31220 a label that dominates the file’s label does not need read permission. An implementation that  
31221 supports write-up operations could fail *fstat()* function calls even though it has a valid file  
31222 descriptor open for writing.

31223 The *lstat()* function is not required to update the time-related fields if the named file is not a  
31224 symbolic link. While the *st\_uid*, *st\_gid*, *st\_atim*, *st\_mtim*, and *st\_ctim* members of the **stat** structure  
31225 may apply to a symbolic link, they are not required to do so. No functions in POSIX.1-200x are  
31226 required to maintain any of these time fields.

31227 The purpose of the *fstatat()* function is to obtain the status of files in directories other than the  
31228 current working directory without exposure to race conditions. Any part of the path of a file  
31229 could be changed in parallel to a call to *stat()*, resulting in unspecified behavior. By opening a  
31230 file descriptor for the target directory and using the *fstatat()* function it can be guaranteed that  
31231 the file for which status is returned is located relative to the desired directory.

### 31232 FUTURE DIRECTIONS

31233 None.

31234 **SEE ALSO**31235 *access()*, *chmod*, *fdopendir()*, *fstat()*, *mknod()*, *readlink()*, *symlink()*31236 XBD Section 4.8 (on page 97), *<fcntl.h>*, *<sys/stat.h>*, *<sys/types.h>* |31237 **CHANGE HISTORY**

31238 First released in Issue 1. Derived from Issue 1 of the SVID.

31239 **Issue 5**

31240 Large File Summit extensions are added.

31241 **Issue 6**31242 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.31243 The following new requirements on POSIX implementations derive from alignment with the  
31244 Single UNIX Specification:

- 31245 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was  
31246 required for conforming implementations of previous POSIX specifications, it was not  
31247 required for UNIX applications.
- 31248 • The [EIO] mandatory error condition is added.
- 31249 • The [ELOOP] mandatory error condition is added.
- 31250 • The [EOVERFLOW] mandatory error condition is added. This change is to support large  
31251 files.
- 31252 • The [ENAMETOOLONG] and the second [EOVERFLOW] optional error conditions are  
31253 added.

31254 The following changes were made to align with the IEEE P1003.1a draft standard:

- 31255 • Details are added regarding the treatment of symbolic links.
- 31256 • The [ELOOP] optional error condition is added.

31257 The normative text is updated to avoid use of the term “must” for application requirements.

31258 The **restrict** keyword is added to the *stat()* prototype for alignment with the ISO/IEC 9899:1999  
31259 standard.31260 **Issue 7**31261 XSH-SD5-ERN-161 is applied, updating the DESCRIPTION to clarify which file types *st\_nlink* +  
31262 applies to. +31263 The *fstatat()* function is added from The Open Group Technical Standard, 2006, Extended API  
31264 Set Part 2.

31265 Changes are made related to support for finegrained timestamps. |

31266 **NAME**31267 `fstatvfs, statvfs` — get file system information31268 **SYNOPSIS**31269 `#include <sys/statvfs.h>`31270 `int fstatvfs(int fildev, struct statvfs *buf);`31271 `int statvfs(const char *restrict path, struct statvfs *restrict buf);`31272 **DESCRIPTION**31273 The `fstatvfs()` function shall obtain information about the file system containing the file  
31274 referenced by *fildev*.31275 The `statvfs()` function shall obtain information about the file system containing the file named by  
31276 *path*.31277 For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read,  
31278 write, or execute permission of the named file is not required.31279 The following flags can be returned in the *f\_flag* member:31280 **ST\_RDONLY** Read-only file system.31281 **ST\_NOSUID** Setuid/setgid bits ignored by *exec*.31282 It is unspecified whether all members of the **statvfs** structure have meaningful values on all file  
31283 systems.31284 **RETURN VALUE**31285 Upon successful completion, `statvfs()` shall return 0. Otherwise, it shall return -1 and set *errno* to  
31286 indicate the error.31287 **ERRORS**31288 The `fstatvfs()` and `statvfs()` functions shall fail if:31289 **[EIO]** An I/O error occurred while reading the file system.31290 **[EINTR]** A signal was caught during execution of the function.31291 **[EOVERFLOW]** One of the values to be returned cannot be represented correctly in the  
31292 structure pointed to by *buf*.31293 The `fstatvfs()` function shall fail if:31294 **[EBADF]** The *fildev* argument is not an open file descriptor.31295 The `statvfs()` function shall fail if:31296 **[EACCES]** Search permission is denied on a component of the path prefix.31297 **[ELOOP]** A loop exists in symbolic links encountered during resolution of the *path*  
31298 argument.31299 **[ENAMETOOLONG]**31300 The length of a pathname exceeds `{PATH_MAX}` or a pathname component is  
31301 longer than `{NAME_MAX}`.31302 **[ENOENT]** A component of *path* does not name an existing file or *path* is an empty string.31303 **[ENOTDIR]** A component of the path prefix of *path* is not a directory.



- 31304 The *statvfs()* function may fail if:
- 31305 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
31306 resolution of the *path* argument.
- 31307 [ENAMETOOLONG]  
31308 Pathname resolution of a symbolic link produced an intermediate result  
31309 whose length exceeds {PATH\_MAX}.

**EXAMPLES****Obtaining File System Information Using *fstatvfs()***

The following example shows how to obtain file system information for the file system upon which the file named */home/cnd/mod1* resides, using the *fstatvfs()* function. The */home/cnd/mod1* file is opened with read/write privileges and the open file descriptor is passed to the *fstatvfs()* function.

```
31316 #include <sys/statvfs.h>
31317 #include <fcntl.h>
31318 struct statvfs buffer;
31319 int status;
31320 ...
31321 fildes = open("/home/cnd/mod1", O_RDWR);
31322 status = fstatvfs(fildes, &buffer);
```

**Obtaining File System Information Using *statvfs()***

The following example shows how to obtain file system information for the file system upon which the file named */home/cnd/mod1* resides, using the *statvfs()* function.

```
31326 #include <sys/statvfs.h>
31327 struct statvfs buffer;
31328 int status;
31329 ...
31330 status = statvfs("/home/cnd/mod1", &buffer);
```

**APPLICATION USAGE**

31331 None.

**RATIONALE**

31333 None.

**FUTURE DIRECTIONS**

31335 None.

**SEE ALSO**

31337 *chmod*, *chown*, *creat()*, *dup()*, *exec*, *fcntl()*, *link*, *mknod()*, *open()*, *pipe()*, *read*, *time*, *unlink*, *utime()*,  
31338 *write*

31339 XBD [<sys/statvfs.h>](#)

**CHANGE HISTORY**

31341 First released in Issue 4, Version 2.

**Issue 5**

31342 Moved from X/OPEN UNIX extension to BASE.

31343 Large File Summit extensions are added.

**fstatvfs()**

31346

**Issue 6**

31347

The normative text is updated to avoid use of the term “must” for application requirements.

31348

31349

The **restrict** keyword is added to the *statvfs()* prototype for alignment with the ISO/IEC 9899:1999 standard.

31350

31351

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

31352

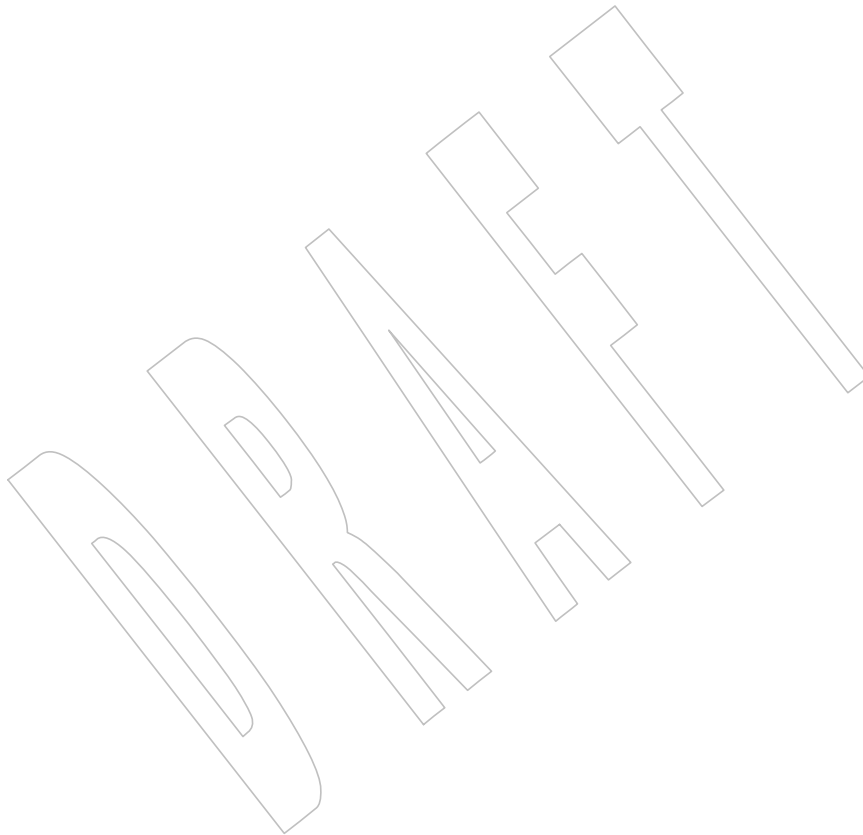
**Issue 7**

31353

The *fstatvfs()* and *statvfs()* functions are moved from the XSI option to the Base.

31354

SD5-XSH-ERN-68 is applied, correcting the EXAMPLES section.



31355 **NAME**

31356 fsync — synchronize changes to a file

31357 **SYNOPSIS**

```
31358 FSC #include <unistd.h>
31359 int fsync(int fildev);
```

31360 **DESCRIPTION**

31361 The *fsync()* function shall request that all data for the open file descriptor named by *fildev* is to be  
 31362 transferred to the storage device associated with the file described by *fildev*. The nature of the  
 31363 transfer is implementation-defined. The *fsync()* function shall not return until the system has  
 31364 completed that action or until an error is detected.

31365 SIO If `_POSIX_SYNCHRONIZED_IO` is defined, the *fsync()* function shall force all currently queued  
 31366 I/O operations associated with the file indicated by file descriptor *fildev* to the synchronized I/O  
 31367 completion state. All I/O operations shall be completed as defined for synchronized I/O file  
 31368 integrity completion.

31369 **RETURN VALUE**

31370 Upon successful completion, *fsync()* shall return 0. Otherwise, `-1` shall be returned and *errno* set  
 31371 to indicate the error. If the *fsync()* function fails, outstanding I/O operations are not guaranteed  
 31372 to have been completed.

31373 **ERRORS**31374 The *fsync()* function shall fail if:

- |       |          |                                                                                          |
|-------|----------|------------------------------------------------------------------------------------------|
| 31375 | [EBADF]  | The <i>fildev</i> argument is not a valid descriptor.                                    |
| 31376 | [EINTR]  | The <i>fsync()</i> function was interrupted by a signal.                                 |
| 31377 | [EINVAL] | The <i>fildev</i> argument does not refer to a file on which this operation is possible. |
| 31378 | [EIO]    | An I/O error occurred while reading from or writing to the file system.                  |

31379 In the event that any of the queued I/O operations fail, *fsync()* shall return the error conditions  
 31380 defined for *read()* and *write()*.

31381 **EXAMPLES**

31382 None.

31383 **APPLICATION USAGE**

31384 The *fsync()* function should be used by programs which require modifications to a file to be  
 31385 completed before continuing; for example, a program which contains a simple transaction  
 31386 facility might use it to ensure that all modifications to a file or files caused by a transaction are  
 31387 recorded.

31388 **RATIONALE**

31389 The *fsync()* function is intended to force a physical write of data from the buffer cache, and to  
 31390 assure that after a system crash or other failure that all data up to the time of the *fsync()* call is  
 31391 recorded on the disk. Since the concepts of “buffer cache”, “system crash”, “physical write”, and  
 31392 “non-volatile storage” are not defined here, the wording has to be more abstract.

31393 If `_POSIX_SYNCHRONIZED_IO` is not defined, the wording relies heavily on the conformance  
 31394 document to tell the user what can be expected from the system. It is explicitly intended that a  
 31395 null implementation is permitted. This could be valid in the case where the system cannot assure  
 31396 non-volatile storage under any circumstances or when the system is highly fault-tolerant and the  
 31397 functionality is not required. In the middle ground between these extremes, *fsync()* might or

31398 might not actually cause data to be written where it is safe from a power failure. The  
 31399 conformance document should identify at least that one configuration exists (and how to obtain  
 31400 that configuration) where this can be assured for at least some files that the user can select to use  
 31401 for critical data. It is not intended that an exhaustive list is required, but rather sufficient  
 31402 information is provided so that if critical data needs to be saved, the user can determine how the  
 31403 system is to be configured to allow the data to be written to non-volatile storage.

31404 It is reasonable to assert that the key aspects of *fsync()* are unreasonable to test in a test suite.  
 31405 That does not make the function any less valuable, just more difficult to test. A formal  
 31406 conformance test should probably force a system crash (power shutdown) during the test for  
 31407 this condition, but it needs to be done in such a way that automated testing does not require this  
 31408 to be done except when a formal record of the results is being made. It would also not be  
 31409 unreasonable to omit testing for *fsync()*, allowing it to be treated as a quality-of-implementation  
 31410 issue.

#### 31411 **FUTURE DIRECTIONS**

31412 None.

#### 31413 **SEE ALSO**

31414 *sync()*

31415 XBD <[unistd.h](#)>

#### 31416 **CHANGE HISTORY**

31417 First released in Issue 3.

#### 31418 **Issue 5**

31419 Aligned with *fsync()* in the POSIX Realtime Extension. Specifically, the DESCRIPTION and  
 31420 RETURN VALUE sections are much expanded, and the ERRORS section is updated to indicate  
 31421 that *fsync()* can return the error conditions defined for *read()* and *write()*.

#### 31422 **Issue 6**

31423 This function is marked as part of the File Synchronization option.

31424 The following new requirements on POSIX implementations derive from alignment with the  
 31425 Single UNIX Specification:

- 31426 • The [EINVAL] and [EIO] mandatory error conditions are added.

31427 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/44 is applied, applying an editorial  
 31428 rewording of the DESCRIPTION. No change in meaning is intended.

31429 **NAME**

31430 ftell, ftello — return a file offset in a stream

31431 **SYNOPSIS**

31432 #include &lt;stdio.h&gt;

31433 long ftell(FILE \*stream);

31434 CX off\_t ftello(FILE \*stream);

31435 **DESCRIPTION**31436 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31437 conflict between the requirements described here and the ISO C standard is unintentional. This  
31438 volume of POSIX.1-200x defers to the ISO C standard.31439 The *ftell()* function shall obtain the current value of the file-position indicator for the stream  
31440 pointed to by *stream*.31441 CX The *ftello()* function shall be equivalent to *ftell()*, except that the return value is of type **off\_t**.31442 **RETURN VALUE**31443 CX Upon successful completion, *ftell()* and *ftello()* shall return the current value of the file-position  
31444 indicator for the stream measured in bytes from the beginning of the file.31445 CX Otherwise, *ftell()* and *ftello()* shall return  $-1$ , cast to **long** and **off\_t** respectively, and set *errno* to  
31446 indicate the error.31447 **ERRORS**31448 CX The *ftell()* and *ftello()* functions shall fail if:31449 CX [EBADF] The file descriptor underlying *stream* is not an open file descriptor.31450 CX [EOVERFLOW] For *ftell()*, the current file offset cannot be represented correctly in an object of  
31451 type **long**.31452 CX [EOVERFLOW] For *ftello()*, the current file offset cannot be represented correctly in an object  
31453 of type **off\_t**.31454 CX [ESPIPE] The file descriptor underlying *stream* is associated with a pipe or FIFO.31455 The *ftell()* function may fail if:31456 CX [ESPIPE] The file descriptor underlying *stream* is associated with a socket.31457 **EXAMPLES**

31458 None.

31459 **APPLICATION USAGE**

31460 None.

31461 **RATIONALE**

31462 None.

31463 **FUTURE DIRECTIONS**

31464 None.

31465 **SEE ALSO**31466 *fgetpos()*, *fopen()*, *fseek()*, *lseek()*

31467 XBD &lt;stdio.h&gt;

31468  
31469  
31470  
31471  
31472  
31473  
31474  
31475  
31476  
31477  
31478

**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

Large File Summit extensions are added.

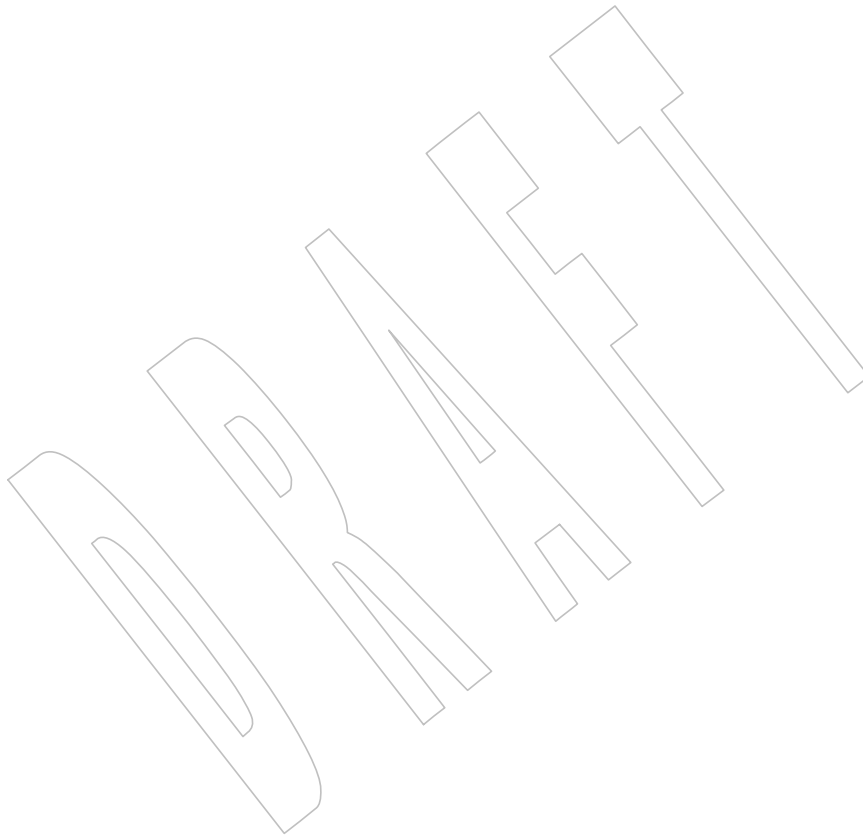
**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *ftello()* function is added.
- The [Eoverflow] error conditions are added.

An additional [ESPIPE] error condition is added for sockets.



31479 **NAME**

31480 ftok — generate an IPC key

31481 **SYNOPSIS**

```
31482 xSI #include <sys/ipc.h>
31483 key_t ftok(const char *path, int id);
```

31484 **DESCRIPTION**

31485 The *ftok()* function shall return a key based on *path* and *id* that is usable in subsequent calls to  
 31486 *msgget()*, *semget()*, and *shmget()*. The application shall ensure that the *path* argument is the  
 31487 pathname of an existing file that the process is able to *stat()*.

31488 The *ftok()* function shall return the same key value for all paths that name the same file, when  
 31489 called with the same *id* value, and return different key values when called with different *id*  
 31490 values or with paths that name different files existing on the same file system at the same time. It  
 31491 is unspecified whether *ftok()* shall return the same key value when called again after the file  
 31492 named by *path* is removed and recreated with the same name.

31493 Only the low-order 8-bits of *id* are significant. The behavior of *ftok()* is unspecified if these bits  
 31494 are 0.

31495 **RETURN VALUE**

31496 Upon successful completion, *ftok()* shall return a key. Otherwise, *ftok()* shall return (**key\_t**)-1  
 31497 and set *errno* to indicate the error.

31498 **ERRORS**

31499 The *ftok()* function shall fail if:

- |                         |                                         |                                                                                                              |
|-------------------------|-----------------------------------------|--------------------------------------------------------------------------------------------------------------|
| 31500                   | [EACCES]                                | Search permission is denied for a component of the path prefix.                                              |
| 31501<br>31502          | [ELOOP]                                 | A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.                   |
| 31503<br>31504<br>31505 | [ENAMETOOLONG]                          | The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}. |
| 31506                   | [ENOENT]                                | A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.                 |
| 31507                   | [ENOTDIR]                               | A component of the path prefix is not a directory.                                                           |
| 31508                   | The <i>ftok()</i> function may fail if: |                                                                                                              |
| 31509<br>31510          | [ELOOP]                                 | More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.       |
| 31511<br>31512<br>31513 | [ENAMETOOLONG]                          | Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.      |

31514 **EXAMPLES**31515 **Getting an IPC Key**

31516 The following example gets a unique key that can be used by the IPC functions *semget()*,  
 31517 *msgget()*, and *shmget()*. The key returned by *ftok()* for this example is based on the ID value *S*  
 31518 and the pathname */tmp*.

```
31519 #include <sys/ipc.h>
31520 ...
31521 key_t key;
31522 char *path = "/tmp";
31523 int id = 'S';
31524 key = ftok(path, id);
```

31525 **Saving an IPC Key**

31526 The following example gets a unique key based on the pathname */tmp* and the ID value *a*. It  
 31527 also assigns the value of the resulting key to the *semkey* variable so that it will be available to a  
 31528 later call to *semget()*, *msgget()*, or *shmget()*.

```
31529 #include <sys/ipc.h>
31530 ...
31531 key_t semkey;
31532 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
31533 perror("IPC error: ftok"); exit(1);
31534 }
```

31535 **APPLICATION USAGE**

31536 For maximum portability, *id* should be a single-byte character.

31537 **RATIONALE**

31538 None.

31539 **FUTURE DIRECTIONS**

31540 None.

31541 **SEE ALSO**

31542 *msgget()*, *semget()*, *shmget()*

31543 XBD [<sys/ipc.h>](#)

31544 **CHANGE HISTORY**

31545 First released in Issue 4, Version 2.

31546 **Issue 5**

31547 Moved from X/OPEN UNIX extension to BASE.

31548 **Issue 6**

31549 The normative text is updated to avoid use of the term “must” for application requirements.

31550 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
 31551 [ELOOP] error condition is added.



31552 **NAME**31553 `ftruncate` — truncate a file to a specified length31554 **SYNOPSIS**31555 `#include <unistd.h>`31556 `int ftruncate(int fildev, off_t length);`31557 **DESCRIPTION**31558 If *fildev* is not a valid file descriptor open for writing, the `ftruncate()` function shall fail.

31559 If *fildev* refers to a regular file, the `ftruncate()` function shall cause the size of the file to be  
 31560 truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no  
 31561 longer be available to reads on the file. If the file previously was smaller than this size,  
 31562 `ftruncate()` shall increase the size of the file. If the file size is increased, the extended area shall  
 31563 appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to  
 31564 `ftruncate()`.

31565 Upon successful completion, if *fildev* refers to a regular file, `ftruncate()` shall mark for update the  
 31566 last data modification and last file status change timestamps of the file and the `S_ISUID` and  
 31567 `S_ISGID` bits of the file mode may be cleared. If the `ftruncate()` function is unsuccessful, the file is  
 31568 unaffected.

31569 XSI If the request would cause the file size to exceed the soft file size limit for the process, the  
 31570 request shall fail and the implementation shall generate the `SIGXFSZ` signal for the thread.

31571 If *fildev* refers to a directory, `ftruncate()` shall fail.31572 If *fildev* refers to any other file type, except a shared memory object, the result is unspecified.

31573 SHM If *fildev* refers to a shared memory object, `ftruncate()` shall set the size of the shared memory  
 31574 object to *length*.

31575 SHM If the effect of `ftruncate()` is to decrease the size of a memory mapped file or a shared memory  
 31576 object and whole pages beyond the new end were previously mapped, then the whole pages  
 31577 beyond the new end shall be discarded.

31578 References to discarded pages shall result in the generation of a `SIGBUS` signal.

31579 If the effect of `ftruncate()` is to increase the size of a memory object, it is unspecified whether the  
 31580 contents of any mapped pages between the old end-of-file and the new are flushed to the  
 31581 underlying object.

31582 **RETURN VALUE**

31583 Upon successful completion, `ftruncate()` shall return 0; otherwise, `-1` shall be returned and `errno`  
 31584 set to indicate the error.

31585 **ERRORS**31586 The `ftruncate()` function shall fail if:

31587 [EINTR] A signal was caught during execution.

31588 [EINVAL] The *length* argument was less than 0.

31589 [EFBIG] or [EINVAL]

31590 The *length* argument was greater than the maximum file size.

31591 [EFBIG] The file is a regular file and *length* is greater than the offset maximum  
 31592 established in the open file description associated with *fildev*.

**ftruncate()**

31593 [EIO] An I/O error occurred while reading from or writing to a file system.  
 31594 [EBADF] or [EINVAL]  
 31595 The *fildest* argument is not a file descriptor open for writing.

**EXAMPLES**

31596 None.  
 31597

**APPLICATION USAGE**

31598 None.  
 31599

**RATIONALE**

31600 None.  
 31601

**FUTURE DIRECTIONS**

31602 None.  
 31603

**SEE ALSO**

31604 *open()*, *truncate()*  
 31605

31606 XBD <[unistd.h](#)>

**CHANGE HISTORY**

31607 First released in Issue 4, Version 2.  
 31608

**Issue 5**

31609 Moved from X/OPEN UNIX extension to BASE and aligned with *ftruncate()* in the POSIX  
 31610 Realtime Extension. Specifically, the DESCRIPTION is extensively reworded and [EROFS] is  
 31611 added to the list of mandatory errors that can be returned by *ftruncate()*.  
 31612

31613 Large File Summit extensions are added.

**Issue 6**

31614 The *truncate()* function is split out into a separate reference page.  
 31615

31616 The following new requirements on POSIX implementations derive from alignment with the  
 31617 Single UNIX Specification:

- The DESCRIPTION is changed to indicate that if the file size is changed, and if the file is a regular file, the S\_ISUID and S\_ISGID bits in the file mode may be cleared.

31620 The following changes were made to align with the IEEE P1003.1a draft standard:

- The DESCRIPTION text is updated.

31622 XSI-conformant systems are required to increase the size of the file if the file was previously  
 31623 smaller than the size requested.

**Issue 7**

31624 Austin Group Interpretation 1003.1-2001 #056 is applied, revising the ERRORS section (although  
 31625 the [EINVAL] “may fail” error was subsequently removed during review of the XSI option).  
 31626

31627 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to  
 31628 the Base.

31629 The DESCRIPTION is updated so that a call to *ftruncate()* when the file is smaller than the size  
 31630 requested will increase the size of the file. Previously, non-XSI-conforming implementations  
 31631 were allowed to increase the size of the file or fail.

31632 Changes are made related to support for finegrained timestamps.

31633 **NAME**  
31634 ftrylockfile — stdio locking functions

31635 **SYNOPSIS**

31636 CX `#include <stdio.h>`  
31637 `int ftrylockfile(FILE *file);`

31638 **DESCRIPTION**

31639 Refer to *flockfile()*.

31640 **NAME**

31641 ftw — traverse (walk) a file tree

31642 **SYNOPSIS**

```
31643 OB XSI #include <ftw.h>
31644
31644 int ftw(const char *path, int (*fn)(const char *,
31645 const struct stat *ptr, int flag), int ndirs);
```

31646 **DESCRIPTION**

31647 The *ftw()* function shall recursively descend the directory hierarchy rooted in *path*. For each  
 31648 object in the hierarchy, *ftw()* shall call the function pointed to by *fn*, passing it a pointer to a null-  
 31649 terminated character string containing the name of the object, a pointer to a **stat** structure  
 31650 containing information about the object, filled in as if *stat()* or *lstat()* had been called to retrieve  
 31651 the information. Possible values of the integer, defined in the **<ftw.h>** header, are:

31652 FTW\_D For a directory.

31653 FTW\_DNR For a directory that cannot be read.

31654 FTW\_F For a file.

31655 FTW\_SL For a symbolic link (but see also FTW\_NS below).

31656 FTW\_NS For an object other than a symbolic link on which *stat()* could not successfully be  
 31657 executed. If the object is a symbolic link and *stat()* failed, it is unspecified whether  
 31658 *ftw()* passes FTW\_SL or FTW\_NS to the user-supplied function.

31659 If the integer is FTW\_DNR, descendants of that directory shall not be processed. If the integer is  
 31660 FTW\_NS, the **stat** structure contains undefined values. An example of an object that would  
 31661 cause FTW\_NS to be passed to the function pointed to by *fn* would be a file in a directory with  
 31662 read but without execute (search) permission.

31663 The *ftw()* function shall visit a directory before visiting any of its descendants.31664 The *ftw()* function shall use at most one file descriptor for each level in the tree.31665 The argument *ndirs* should be in the range [1,{OPEN\_MAX}].

31666 The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a  
 31667 non-zero value, or some error, other than [EACCES], is detected within *ftw()*.

31668 The *ndirs* argument shall specify the maximum number of directory streams or file descriptors  
 31669 or both available for use by *ftw()* while traversing the tree. When *ftw()* returns it shall close any  
 31670 directory streams and file descriptors it uses not counting any opened by the application-  
 31671 supplied *fn* function.

31672 The results are unspecified if the application-supplied *fn* function does not preserve the current  
 31673 working directory.

31674 The *ftw()* function need not be thread-safe. A function that is not required to be thread-safe is  
 31675 not required to be reentrant.

31676 **RETURN VALUE**

31677 If the tree is exhausted, *ftw()* shall return 0. If the function pointed to by *fn* returns a non-zero  
 31678 value, *ftw()* shall stop its tree traversal and return whatever value was returned by the function  
 31679 pointed to by *fn*. If *ftw()* detects an error, it shall return -1 and set *errno* to indicate the error.

31680 If *ftw()* encounters an error other than [EACCES] (see FTW\_DNR and FTW\_NS above), it shall  
 31681 return -1 and set *errno* to indicate the error. The external variable *errno* may contain any error

31682 value that is possible when a directory is opened or when one of the *stat* functions is executed on  
31683 a directory or file.

**ERRORS**

31684 The *ftw()* function shall fail if:  
31685

31686 [EACCES] Search permission is denied for any component of *path* or read permission is  
31687 denied for *path*.

31688 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*  
31689 argument.

31690 [ENAMETOOLONG]  
31691 The length of the *path* argument exceeds {PATH\_MAX} or a pathname  
31692 component is longer than {NAME\_MAX}.

31693 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

31694 [ENOTDIR] A component of *path* is not a directory.

31695 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current  
31696 programming environment for one or more files found in the file hierarchy.

31697 The *ftw()* function may fail if:

31698 [EINVAL] The value of the *ndirs* argument is invalid.

31699 [ELOOP] More than {SYMLOOP\_MAX} symbolic links were encountered during  
31700 resolution of the *path* argument.

31701 [ENAMETOOLONG]  
31702 Pathname resolution of a symbolic link produced an intermediate result  
31703 whose length exceeds {PATH\_MAX}.

31704 In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set  
31705 accordingly.

**EXAMPLES****Walking a Directory Structure**

31707 The following example walks the current directory structure, calling the *fn* function for every  
31708 directory entry, using at most 10 file descriptors:  
31709

```
31710 #include <ftw.h>
31711 ...
31712 if (ftw(".", fn, 10) != 0) {
31713 perror("ftw"); exit(2);
31714 }
```

**APPLICATION USAGE**

31715 The *ftw()* function may allocate dynamic storage during its operation. If *ftw()* is forcibly  
31716 terminated, such as by *longjmp()* or *siglongjmp()* being executed by the function pointed to by *fn*  
31717 or an interrupt routine, *ftw()* does not have a chance to free that storage, so it remains  
31718 permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has  
31719 occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next  
31720 invocation.  
31721

31722 Applications should use the *nftw()* function instead of the obsolescent *ftw()* function.

31723  
31724  
31725  
31726  
31727  
31728  
31729  
31730  
31731  
31732  
31733  
31734  
31735  
31736  
31737  
31738  
31739  
31740  
31741  
31742  
31743  
31744**RATIONALE**

None.

**FUTURE DIRECTIONS**The *ftw()* function may be removed in a future version.**SEE ALSO***fdopendir()*, *fstatat()*, *longjmp()*, *malloc()*, *nftw()*, *siglongjmp()*XBD <*ftw.h*>, <*sys/stat.h*>**CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

**Issue 5**

UX codings in the DESCRIPTION, RETURN VALUE, and ERRORS sections are changed to EX.

**Issue 6**

The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [Eoverflow] mandatory error condition is added.

A note is added to the DESCRIPTION indicating that this function need not be reentrant, and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

**Issue 7**

SD5-XBD-ERN-61 is applied.

The *ftw()* function is marked obsolescent.

31745 **NAME**  
31746 funlockfile — stdio locking functions

31747 **SYNOPSIS**

31748 CX `#include <stdio.h>`  
31749 `void funlockfile(FILE *file);`

31750 **DESCRIPTION**

31751 Refer to *flockfile()*.

## 31752 NAME

31753 futimens, utimensat, utimes — set file access and modification times |

## 31754 SYNOPSIS

31755 #include &lt;sys/stat.h&gt; +

31756 int futimens(int *fd*, const struct timespec *times*[2]); +31757 int utimensat(int *fd*, const char \**path*, const struct timespec *times*[2], +31758 int *flag*); +

31759 XSI #include &lt;sys/time.h&gt;

31760 int utimes(const char \**path*, const struct timeval *times*[2]); |

## 31761 DESCRIPTION

31762 The *futimens()* and *utimensat()* functions shall set the access and modification times of a file to |  
 31763 the values of the *times* argument. The *futimens()* function changes the times of the file associated |  
 31764 with the file descriptor *fd*. The *utimensat()* function changes the times of the file pointed to by |  
 31765 the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions |  
 31766 allow time specifications accurate to the nanosecond. |

31767 For *futimens()* and *utimensat()*, the *times* argument is an array of two **timespec** structures. The |  
 31768 first array member represents the date and time of last access, and the second member |  
 31769 represents the date and time of last modification. The times in the **timespec** structure are |  
 31770 measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set |  
 31771 to the greatest value supported by the file system that is not greater than the specified time. |

31772 If the *tv\_nsec* field of a **timespec** structure has the special value **UTIME\_NOW**, the file's relevant |  
 31773 timestamp shall be set to the greatest value supported by the file system that is not greater than |  
 31774 the current time. If the *tv\_nsec* field has the special value **UTIME\_OMIT**, the file's relevant |  
 31775 timestamp shall not be changed. In either case, the *tv\_sec* field shall be ignored. |

31776 If the *times* argument is a null pointer, both the access and modification timestamps shall be set |  
 31777 to the greatest value supported by the file system that is not greater than the current time. If |  
 31778 *utimensat()* is passed a relative path in the *path* argument, the file to be used shall be relative to |  
 31779 the directory associated with the file descriptor *fd* instead of the current working directory. If |  
 31780 *utimensat()* is passed the special value **AT\_FDCWD** in the *fd* parameter, the current working |  
 31781 directory shall be used. |

31782 Only a process with the effective user ID equal to the user ID of the file, or with write access to |  
 31783 the file, or with appropriate privileges may use *utimensat()* with a null pointer as the *times* |  
 31784 argument or with both *tv\_nsec* fields set to the special value **UTIME\_NOW**. Only a process with |  
 31785 the effective user ID equal to the user ID of the file or with appropriate privileges may use |  
 31786 *utimensat()* with a non-null *times* argument that does not have both *tv\_nsec* fields set to |  
 31787 **UTIME\_NOW** and does not have both *tv\_nsec* fields set to **UTIME\_OMIT**. If both *tv\_nsec* fields |  
 31788 are set to **UTIME\_OMIT**, no ownership or permissions check shall be performed for the file, but |  
 31789 other error conditions may still be detected (including [EACCES] errors related to the path |  
 31790 prefix). |

31791 Values for the *flag* argument of *utimensat()* are constructed by a bitwise-inclusive OR of flags |  
 31792 from the following list, defined in <fcntl.h>: |

31793 AT\_SYMLINK\_NOFOLLOW |

31794 If *path* names a symbolic link, then the access and modification times of the symbolic link |  
 31795 are changed. |

31796 Upon completion, *futimens()* and *utimensat()* shall mark the last file status change timestamp for |



31797

update.

31798

31799

31800

31801

The *utimes()* function shall be equivalent to the *utimensat()* function with the special value *AT\_FDCWD* as the *fd* argument and the *flag* argument set to zero, except that the *times* argument is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond, not nanosecond, and rounding towards the nearest second may occur.

31802

**RETURN VALUE**

31803

31804

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return  $-1$  and set *errno* to indicate the error. If  $-1$  is returned, the file times shall not be affected.

31805

**ERRORS**

31806

These functions shall fail if:

31807

31808

[EACCES] The *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.

31809

31810

31811

[EINVAL] Either of the *times* argument structures specified a *tv\_nsec* value that was neither *UTIME\_NOW* nor *UTIME\_OMIT*, and was a value less than zero or greater than or equal to 1 000 million.

31812

31813

[EINVAL] A new file timestamp would be a value whose *tv\_sec* component is not a value supported by the file system.

31814

31815

31816

31817

31818

[EPERM] The *times* argument is not a null pointer, does not have both *tv\_nsec* fields set to *UTIME\_NOW*, does not have both *tv\_nsec* fields set to *UTIME\_OMIT*, the calling process' effective user ID has write access to the file but does not match the owner of the file, and the calling process does not have the appropriate privileges.

31819

[EROFS] The file system containing the file is read-only.

31820

The *futimens()* function shall fail if:

31821

[EBADF] The *fd* argument is not a valid file descriptor.

31822

The *utimensat()* function shall fail if:

31823

31824

[EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither *AT\_FDCWD* nor a valid file descriptor open for reading.

31825

The *utimensat()* and *utimes()* functions shall fail if:

31826

[EACCES] Search permission is denied by a component of the path prefix.

31827

31828

[ELOOP] A loop exists in symbolic links encountered during resolution of the *path* argument.

31829

31830

31831

[ENAMETOOLONG] The length of the *path* argument exceeds  $\{\text{PATH\_MAX}\}$  or a pathname component is longer than  $\{\text{NAME\_MAX}\}$ .

31832

[ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

31833

[ENOTDIR] A component of the path prefix is not a directory.

31834

These functions may fail if:

31835

31836

[ELOOP] More than  $\{\text{SYMLOOP\_MAX}\}$  symbolic links were encountered during resolution of the *path* argument.

31837

31838

31839

[ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result whose length exceeds  $\{\text{PATH\_MAX}\}$ .

31840 The *utimensat*() function may fail if:

31841 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT\_FDCWD nor a  
31842 file descriptor associated with a directory.

#### EXAMPLES

31843 None.  
31844

#### APPLICATION USAGE

31845 None.  
31846

#### RATIONALE

31847 The purpose of the *utimensat*() function is to set the access and modification time of files in  
31848 directories other than the current working directory without exposure to race conditions. Any  
31849 part of the path of a file could be changed in parallel to a call to *utimes*(), resulting in unspecified  
31850 behavior. By opening a file descriptor for the target directory and using the *utimensat*() function  
31851 it can be guaranteed that the changed file is located relative to the desired directory.  
31852

31853 The standard developers considered including a special case for the permissions required by  
31854 *utimensat*() when one *tv\_nsec* field is *UTIME\_NOW* and the other is *UTIME\_OMIT*. One  
31855 possibility would be to include this case in with the cases where *times* is a null pointer or both  
31856 fields are *UTIME\_NOW*, where the call is allowed if the process has write permission for the file.  
31857 However, associating write permission with an update to just the last data access timestamp  
31858 (which is normally updated by *read*()) did not seem appropriate. The other possibility would be  
31859 to specify that this one case is allowed if the process has read permission, but this was felt to be  
31860 too great a departure from the *utime*() and *utimes*() functions on which *utimensat*() is based. If  
31861 an application needs to set the last data access timestamp to the current time for a file on which  
31862 it has read permission but is not the owner, it can do so by opening the file, reading one or more  
31863 bytes (or reading a directory entry, if the file is a directory), and then closing it.

#### FUTURE DIRECTIONS

31864 None.  
31865

#### SEE ALSO

31866 *read*, *utime*()

31867 XBD <fcntl.h>, <sys/stat.h>, <sys/time.h>

#### CHANGE HISTORY

31869 First released in Issue 4, Version 2.  
31870

##### Issue 5

31871 Moved from X/OPEN UNIX extension to BASE.  
31872

##### Issue 6

31873 This function is marked LEGACY.  
31874

31875 The normative text is updated to avoid use of the term “must” for application requirements.

31876 The wording of the mandatory [ELOOP] error condition is updated, and a second optional  
31877 [ELOOP] error condition is added.

##### Issue 7

31878 The LEGACY marking is removed.  
31879

31880 The *utimensat*() function (renamed from *futimesat*()) is added from The Open Group Technical  
31881 Standard, 2006, Extended API Set Part 2, and changed to allow modifying a symbolic link by  
31882 adding a *flag* argument.

31883 The *futimens*() function is added.

31884 Changes are made related to support for finegrained timestamps.  
31885

31886 **NAME**31887 `fwide` — set stream orientation31888 **SYNOPSIS**31889 `#include <stdio.h>`31890 `#include <wchar.h>`31891 `int fwide(FILE *stream, int mode);`31892 **DESCRIPTION**31893 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
31894 conflict between the requirements described here and the ISO C standard is unintentional. This  
31895 volume of POSIX.1-200x defers to the ISO C standard.31896 The `fwide()` function shall determine the orientation of the stream pointed to by `stream`. If `mode` is  
31897 greater than zero, the function first attempts to make the stream wide-oriented. If `mode` is less  
31898 than zero, the function first attempts to make the stream byte-oriented. Otherwise, `mode` is zero  
31899 and the function does not alter the orientation of the stream.31900 If the orientation of the stream has already been determined, `fwide()` shall not change it.31901 CX Since no return value is reserved to indicate an error, an application wishing to check for error  
31902 situations should set `errno` to 0, then call `fwide()`, then check `errno`, and if it is non-zero, assume  
31903 an error has occurred.31904 **RETURN VALUE**31905 The `fwide()` function shall return a value greater than zero if, after the call, the stream has wide-  
31906 orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no  
31907 orientation.31908 **ERRORS**31909 The `fwide()` function may fail if:31910 CX [EBADF] The `stream` argument is not a valid stream.31911 **EXAMPLES**

31912 None.

31913 **APPLICATION USAGE**31914 A call to `fwide()` with `mode` set to zero can be used to determine the current orientation of a  
31915 stream.31916 **RATIONALE**

31917 None.

31918 **FUTURE DIRECTIONS**

31919 None.

31920 **SEE ALSO**31921 XBD [<wchar.h>](#)31922 **CHANGE HISTORY**31923 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995  
31924 (E).31925 **Issue 6**

31926 Extensions beyond the ISO C standard are marked.

31927 **NAME**

31928 fwprintf, swprintf, wprintf — print formatted wide-character output

31929 **SYNOPSIS**

31930 #include &lt;stdio.h&gt;

31931 #include &lt;wchar.h&gt;

31932 int fwprintf(FILE \*restrict stream, const wchar\_t \*restrict format, ...);

31933 int swprintf(wchar\_t \*restrict ws, size\_t n,

31934 const wchar\_t \*restrict format, ...);

31935 int wprintf(const wchar\_t \*restrict format, ...);

31936 **DESCRIPTION**

31937 CX The functionality described on this reference page is aligned with the ISO C standard. Any  
 31938 conflict between the requirements described here and the ISO C standard is unintentional. This  
 31939 volume of POSIX.1-200x defers to the ISO C standard.

31940 The *fwprintf()* function shall place output on the named output *stream*. The *wprintf()* function  
 31941 shall place output on the standard output stream *stdout*. The *swprintf()* function shall place  
 31942 output followed by the null wide character in consecutive wide characters starting at *\*ws*; no  
 31943 more than *n* wide characters shall be written, including a terminating null wide character, which  
 31944 is always added (unless *n* is zero).

31945 Each of these functions shall convert, format, and print its arguments under control of the *format*  
 31946 wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*,  
 31947 which are simply copied to the output stream, and *conversion specifications*, each of which results  
 31948 in the fetching of zero or more arguments. The results are undefined if there are insufficient  
 31949 arguments for the *format*. If the *format* is exhausted while arguments remain, the excess  
 31950 arguments are evaluated but are otherwise ignored.

31951 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than  
 31952 to the next unused argument. In this case, the conversion specifier wide character % (see below)  
 31953 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range  
 31954 [1,{NL\_ARGMAX}], giving the position of the argument in the argument list. This feature  
 31955 provides for the definition of *format* wide-character strings that select arguments in an order  
 31956 appropriate to specific languages (see the EXAMPLES section).

31957 The *format* can contain either numbered argument specifications (that is, "%n\$" and "\*m\$"), or  
 31958 unnumbered argument conversion specifications (that is, % and \*), but not both. The only  
 31959 exception to this is that %% can be mixed with the "%n\$" form. The results of mixing numbered  
 31960 and unnumbered argument specifications in a *format* wide-character string are undefined. When  
 31961 numbered argument specifications are used, specifying the *N*th argument requires that all the  
 31962 leading arguments, from the first to the (*N*-1)th, are specified in the *format* wide-character string.

31963 In *format* wide-character strings containing the "%n\$" form of conversion specification,  
 31964 numbered arguments in the argument list can be referenced from the *format* wide-character  
 31965 string as many times as required.

31966 In *format* wide-character strings containing the % form of conversion specification, each  
 31967 argument in the argument list shall be used exactly once.

31968 CX All forms of the *fwprintf()* function allow for the insertion of a locale-dependent radix character  
 31969 in the output string, output as a wide-character value. The radix character is defined in the  
 31970 locale of the process (category *LC\_NUMERIC*). In the POSIX locale, or in a locale where the  
 31971 radix character is not defined, the radix character shall default to a period ('.').

- 31972 CX Each conversion specification is introduced by the '`%`' wide character or by the wide-character  
31973 sequence "`%n$`", after which the following appear in sequence:
- 31974 • Zero or more *flags* (in any order), which modify the meaning of the conversion  
31975 specification.
  - 31976 • An optional minimum *field width*. If the converted value has fewer wide characters than  
31977 the field width, it shall be padded with spaces by default on the left; it shall be padded on  
31978 the right, if the left-adjustment flag ('`-`'), described below, is given to the field width. The  
31979 field width takes the form of an asterisk ('`*`'), described below, or a decimal integer.
  - 31980 • An optional *precision* that gives the minimum number of digits to appear for the `d`, `i`, `o`, `u`,  
31981 `x`, and `X` conversion specifiers; the number of digits to appear after the radix character for  
31982 the `a`, `A`, `e`, `E`, `f`, and `F` conversion specifiers; the maximum number of significant digits for  
31983 the `g` and `G` conversion specifiers; or the maximum number of wide characters to be  
31984 printed from a string in the `s` conversion specifiers. The precision takes the form of a  
31985 period ('`.`') followed either by an asterisk ('`*`'), described below, or an optional decimal  
31986 digit string, where a null digit string is treated as 0. If a precision appears with any other  
31987 conversion wide character, the behavior is undefined.
  - 31988 • An optional length modifier that specifies the size of the argument.
  - 31989 • A *conversion specifier* wide character that indicates the type of conversion to be applied.
- 31990 A field width, or precision, or both, may be indicated by an asterisk ('`*`'). In this case an  
31991 argument of type `int` supplies the field width or precision. Applications shall ensure that  
31992 arguments specifying field width, or precision, or both appear in that order before the argument,  
31993 if any, to be converted. A negative field width is taken as a '`-`' flag followed by a positive field  
31994 width. A negative precision is taken as if the precision were omitted. In *format* wide-character  
31995 strings containing the "`%n$`" form of a conversion specification, a field width or precision may  
31996 be indicated by the sequence "`*m$`", where `m` is a decimal integer in the range  
31997 `[1, {NL_ARGMAX}]` giving the position in the argument list (after the *format* argument) of an  
31998 integer argument containing the field width or precision, for example:
- ```
31999 wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```
- 32000 The flag wide characters and their meanings are:
- 32001 CX '`'` The integer portion of the result of a decimal conversion (`%i`, `%d`, `%u`, `%f`, `%F`, `%g`, or `%G`)
32002 shall be formatted with thousands' grouping wide characters. For other conversions,
32003 the behavior is undefined. The numeric grouping wide character is used.
 - 32004 `-` The result of the conversion shall be left-justified within the field. The conversion shall
32005 be right-justified if this flag is not specified.
 - 32006 `+` The result of a signed conversion shall always begin with a sign ('`+`' or '`-`'). The
32007 conversion shall begin with a sign only when a negative value is converted if this flag is
32008 not specified.
 - 32009 `<space>` If the first wide character of a signed conversion is not a sign, or if a signed conversion
32010 results in no wide characters, a `<space>` shall be prefixed to the result. This means that
32011 if the `<space>` and '`+`' flags both appear, the `<space>` flag shall be ignored.
 - 32012 `#` Specifies that the value is to be converted to an alternative form. For `o` conversion, it
32013 increases the precision (if necessary) to force the first digit of the result to be 0. For `x` or
32014 `X` conversion specifiers, a non-zero result shall have `0x` (or `0X`) prefixed to it. For `a`, `A`, `e`,
32015 `E`, `f`, `F`, `g`, and `G` conversion specifiers, the result shall always contain a radix character,
32016 even if no digits follow it. Without this flag, a radix character appears in the result of
32017 these conversions only if a digit follows it. For `g` and `G` conversion specifiers, trailing
32018 zeros shall *not* be removed from the result as they normally are. For other conversion
32019 specifiers, the behavior is undefined.

32020 0 For `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversion specifiers, leading zeros
 32021 (following any indication of sign or base) are used to pad to the field width; no space
 32022 padding is performed. If the `'0'` and `'-'` flags both appear, the `'0'` flag shall be
 32023 ignored. For `d`, `i`, `o`, `u`, `x`, and `X` conversion specifiers, if a precision is specified, the `'0'`
 32024 CX flag shall be ignored. If the `'0'` and `'.'` flags both appear, the grouping wide
 32025 characters are inserted before zero padding. For other conversions, the behavior is
 32026 undefined.

32027 The length modifiers and their meanings are:

32028 hh Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **signed char**
 32029 or **unsigned char** argument (the argument will have been promoted according to the
 32030 integer promotions, but its value shall be converted to **signed char** or **unsigned char**
 32031 before printing); or that a following `n` conversion specifier applies to a pointer to a
 32032 **signed char** argument.

32033 h Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **short** or
 32034 **unsigned short** argument (the argument will have been promoted according to the
 32035 integer promotions, but its value shall be converted to **short** or **unsigned short** before
 32036 printing); or that a following `n` conversion specifier applies to a pointer to a **short**
 32037 argument.

32038 l (ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long** or
 32039 **unsigned long** argument; that a following `n` conversion specifier applies to a pointer to
 32040 a **long** argument; that a following `c` conversion specifier applies to a **wint_t** argument;
 32041 that a following `s` conversion specifier applies to a pointer to a **wchar_t** argument; or
 32042 has no effect on a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier.

32043 ll (ell-ell) Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **long long**
 32044 or **unsigned long long** argument; or that a following `n` conversion specifier applies to a
 32045 pointer to a **long long** argument.
 32046

32047 j Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to an **intmax_t**
 32048 or **uintmax_t** argument; or that a following `n` conversion specifier applies to a pointer
 32049 to an **intmax_t** argument.

32050 z Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **size_t** or the
 32051 corresponding signed integer type argument; or that a following `n` conversion specifier
 32052 applies to a pointer to a signed integer type corresponding to a **size_t** argument.

32053 t Specifies that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a **ptrdiff_t** or
 32054 the corresponding **unsigned** type argument; or that a following `n` conversion specifier
 32055 applies to a pointer to a **ptrdiff_t** argument.

32056 L Specifies that a following `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier applies to a **long**
 32057 **double** argument.

32058 If a length modifier appears with any conversion specifier other than as specified above, the
 32059 behavior is undefined.

32060 The conversion specifiers and their meanings are:

32061 `d`, `i` The **int** argument shall be converted to a signed decimal in the style "`[-]dddd`". The
 32062 precision specifies the minimum number of digits to appear; if the value being
 32063 converted can be represented in fewer digits, it shall be expanded with leading zeros.
 32064 The default precision shall be 1. The result of converting zero with an explicit precision
 32065 of zero shall be no wide characters.

- 32066 o The **unsigned** argument shall be converted to unsigned octal format in the style
32067 "dddd". The precision specifies the minimum number of digits to appear; if the value
32068 being converted can be represented in fewer digits, it shall be expanded with leading
32069 zeros. The default precision shall be 1. The result of converting zero with an explicit
32070 precision of zero shall be no wide characters.
- 32071 u The **unsigned** argument shall be converted to unsigned decimal format in the style
32072 "dddd". The precision specifies the minimum number of digits to appear; if the value
32073 being converted can be represented in fewer digits, it shall be expanded with leading
32074 zeros. The default precision shall be 1. The result of converting zero with an explicit
32075 precision of zero shall be no wide characters.
- 32076 x The **unsigned** argument shall be converted to unsigned hexadecimal format in the style
32077 "dddd"; the letters "abcdef" are used. The precision specifies the minimum number
32078 of digits to appear; if the value being converted can be represented in fewer digits, it
32079 shall be expanded with leading zeros. The default precision shall be 1. The result of
32080 converting zero with an explicit precision of zero shall be no wide characters.
- 32081 X Equivalent to the x conversion specifier, except that letters "ABCDEF" are used instead
32082 of "abcdef".
- 32083 f, F The **double** argument shall be converted to decimal notation in the style
32084 "[-]ddd.d_{ddd}", where the number of digits after the radix character shall be equal to
32085 the precision specification. If the precision is missing, it shall be taken as 6; if the
32086 precision is explicitly zero and no '#' flag is present, no radix character shall appear. If
32087 a radix character appears, at least one digit shall appear before it. The value shall be
32088 rounded in an implementation-defined manner to the appropriate number of digits.
- 32089 A **double** argument representing an infinity shall be converted in one of the styles
32090 "[-]inf" or "[-]infinity"; which style is implementation-defined. A **double**
32091 argument representing a NaN shall be converted in one of the styles "[-]nan" or
32092 "[-]nan(*n-char-sequence*)"; which style, and the meaning of any *n-char-sequence*,
32093 is implementation-defined. The F conversion specifier produces "INF", "INFINITY",
32094 or "NAN" instead of "inf", "infinity", or "nan", respectively.
- 32095 e, E The **double** argument shall be converted in the style "[-]d.ddde±dd", where there
32096 shall be one digit before the radix character (which is non-zero if the argument is non-
32097 zero) and the number of digits after it shall be equal to the precision; if the precision is
32098 missing, it shall be taken as 6; if the precision is zero and no '#' flag is present, no
32099 radix character shall appear. The value shall be rounded in an implementation-defined
32100 manner to the appropriate number of digits. The E conversion wide character shall
32101 produce a number with 'E' instead of 'e' introducing the exponent. The exponent
32102 shall always contain at least two digits. If the value is zero, the exponent shall be zero.
- 32103 A **double** argument representing an infinity or NaN shall be converted in the style of
32104 an f or F conversion specifier.
- 32105 g, G The **double** argument representing a floating-point number shall be converted in the |
32106 style f or e (or in the style F or E in the case of a G conversion specifier), depending on |
32107 the value converted and the precision. Let P equal the precision if non-zero, 6 if the |
32108 precision is omitted, or 1 if the precision is zero. Then, if a conversion with style E |
32109 would have an exponent of X: |
- 32110 — If $P > X \geq -4$, the conversion shall be with style f (or F) and precision $P - (X + 1)$. |
32111 — Otherwise, the conversion shall be with style e (or E) and precision $P - 1$. |
- 32112 Finally, unless the '#' flag is used, any trailing zeros shall be removed from the |
32113 fractional portion of the result and the decimal-point character shall be removed if there |
32114 is no fractional portion remaining. |

32115			A double argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
32116			
32117	a, A		A double argument representing a floating-point number shall be converted in the style " <code>[-]0xh.hhhhp±d</code> ", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and <code>FLT_RADIX</code> is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and <code>FLT_RADIX</code> is not a power of 2, then the precision shall be sufficient to distinguish values of type double , except that trailing zeros may be omitted; if the precision is zero and the <code>'#'</code> flag is not specified, no decimal-point wide character shall appear. The letters "abcdef" are used for a conversion and the letters "ABCDEF" for A conversion. The A conversion specifier produces a number with <code>'X'</code> and <code>'P'</code> instead of <code>'x'</code> and <code>'p'</code> . The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.
32118			
32119			
32120			
32121			
32122			
32123			
32124			
32125			
32126			
32127			
32128			
32129			
32130			
32131			A double argument representing an infinity or NaN shall be converted in the style of an <code>f</code> or <code>F</code> conversion specifier.
32132			
32133	c		If no <code>l</code> (ell) qualifier is present, the int argument shall be converted to a wide character as if by calling the <code>btowc()</code> function and the resulting wide character shall be written. Otherwise, the wint_t argument shall be converted to wchar_t , and written.
32134			
32135			
32136	s		If no <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the <code>mbrtowc()</code> function, with the conversion state described by an mbstate_t object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.
32137			
32138			
32139			
32140			
32141			
32142			
32143			
32144			
32145			If an <code>l</code> (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type wchar_t . Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.
32146			
32147			
32148			
32149			
32150			
32151	p		The application shall ensure that the argument is a pointer to void . The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.
32152			
32153			
32154	n		The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the <code>fwprintf()</code> functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
32155			
32156			
32157			
32158			
32159	XSI	C	Equivalent to <code>lc</code> .
32160	XSI	S	Equivalent to <code>ls</code> .
32161		%	Output a <code>'%'</code> wide character; no argument shall be converted. The entire conversion specification shall be <code>%%</code> .
32162			
32163			If a conversion specification does not match one of the above forms, the behavior is undefined.

32164 In no case does a nonexistent or small field width cause truncation of a field; if the result of a
 32165 conversion is wider than the field width, the field shall be expanded to contain the conversion
 32166 result. Characters generated by *fwprintf()* and *wprintf()* shall be printed as if *fputwc()* had been
 32167 called.

32168 For a and A conversions, if FLT_RADIX is not a power of 2 and the result is not exactly
 32169 representable in the given precision, the result should be one of the two adjacent numbers in
 32170 hexadecimal floating style with the given precision, with the extra stipulation that the error
 32171 should have a correct sign for the current rounding direction.

32172 For e, E, f, F, g, and G conversion specifiers, if the number of significant decimal digits is at
 32173 most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant
 32174 decimal digits is more than DECIMAL_DIG but the source value is exactly representable with
 32175 DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros.
 32176 Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having
 32177 DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L \leq$
 32178 $D \leq U$, with the extra stipulation that the error should have a correct sign for the current
 32179 rounding direction.

32180 CX The last data modification and last file status change timestamps of the file shall be marked for
 32181 update between the call to a successful execution of *fwprintf()* or *wprintf()* and the next
 32182 successful completion of a call to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or
 32183 *abort()*.

32184 RETURN VALUE

32185 Upon successful completion, these functions shall return the number of wide characters
 32186 transmitted, excluding the terminating null wide character in the case of *swprintf()*, or a negative
 32187 value if an output error was encountered, and set *errno* to indicate the error.

32188 If n or more wide characters were requested to be written, *swprintf()* shall return a negative
 32189 value, and set *errno* to indicate the error.

32190 ERRORS

32191 For the conditions under which *fwprintf()* and *wprintf()* fail and may fail, refer to *fputwc()*.

32192 In addition, all forms of *fwprintf()* may fail if:

32193 CX [EILSEQ] A wide-character code that does not correspond to a valid character has been
 32194 detected.

32195 CX [EINVAL] There are insufficient arguments.

32196 In addition, *fwprintf()* and *wprintf()* may fail if:

32197 CX [ENOMEM] Insufficient storage space is available.

32198 The *swprintf()* shall fail if:

32199 CX [EOVERFLOW] The value of n is greater than {INT_MAX} or the number of bytes needed to
 32200 hold the output excluding the terminating null is greater than {INT_MAX}.

32201 EXAMPLES

32202 To print the language-independent date and time format, the following statement could be used:

32203 `wprintf(format, weekday, month, day, hour, min);`

32204 For American usage, *format* could be a pointer to the wide-character string:

32205 `L"%s, %s %d, %d:%.2d\n"`

32206 producing the message:

32207 Sunday, July 3, 10:02

32208 whereas for German usage, *format* could be a pointer to the wide-character string:

32209 L"%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\n"

32210 producing the message:

32211 Sonntag, 3. Juli, 10:02

32212 APPLICATION USAGE

32213 None.

32214 RATIONALE

32215 None.

32216 FUTURE DIRECTIONS

32217 None.

32218 SEE ALSO

32219 *btowc()*, *fputwc()*, *fwscanf()*, *mbrtowc()*, *setlocale()*

32220 XBD Chapter 7 (on page 121), `<stdio.h>`, `<wchar.h>` |

32221 CHANGE HISTORY

32222 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
32223 (E).

32224 Issue 6

32225 The Open Group Corrigendum U040/1 is applied to the RETURN VALUE section, describing
32226 the case if *n* or more wide characters are requested to be written using *swprintf()*.

32227 The normative text is updated to avoid use of the term “must” for application requirements.

32228 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 32229 • The prototypes for *fwprintf()*, *swprintf()*, and *wprintf()* are updated.
- 32230 • The DESCRIPTION is updated.
- 32231 • The *hh*, *ll*, *j*, *t*, and *z* length modifiers are added.
- 32232 • The *a*, *A*, and *F* conversion characters are added.
- 32233 • XSI shading is removed from the description of character string representations of infinity
32234 and NaN floating-point values.

32235 The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion
32236 specification” consistently.

32237 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

32238 Issue 7

32239 Functionality relating to the “%n\$” form of conversion specification and the ‘ ’ (apostrophe)
32240 flag is moved from the XSI option to the Base.

32241 The [Eoverflow] error is added for *swprintf()*.

32242 Changes are made related to support for finegrained timestamps. +

32243 ISO C TC2 #68 is applied.

32244 **NAME**

32245 fwrite — binary output

32246 **SYNOPSIS**

32247 #include <stdio.h>

32248 size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
32249 FILE *restrict stream);32250 **DESCRIPTION**32251 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32252 conflict between the requirements described here and the ISO C standard is unintentional. This
32253 volume of POSIX.1-200x defers to the ISO C standard.32254 The *fwrite()* function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose
32255 size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be
32256 made to the *fputc()* function, taking the values (in order) from an array of **unsigned char** exactly
32257 overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by
32258 the number of bytes successfully written. If an error occurs, the resulting value of the file-
32259 position indicator for the stream is unspecified.32260 CX The last data modification and last file status change timestamps of the file shall be marked for
32261 update between the successful execution of *fwrite()* and the next successful completion of a call
32262 to *fflush()* or *fclose()* on the same stream, or a call to *exit()* or *abort()*.32263 **RETURN VALUE**32264 The *fwrite()* function shall return the number of elements successfully written, which may be
32265 less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite()* shall return 0 and the
32266 state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for
32267 CX the stream shall be set, and *errno* shall be set to indicate the error.32268 **ERRORS**32269 Refer to *fputc()*.32270 **EXAMPLES**

32271 None.

32272 **APPLICATION USAGE**32273 Because of possible differences in element length and byte ordering, files written using *fwrite()*
32274 are application-dependent, and possibly cannot be read using *fread()* by a different application
32275 or by the same application on a different processor.32276 **RATIONALE**

32277 None.

32278 **FUTURE DIRECTIONS**

32279 None.

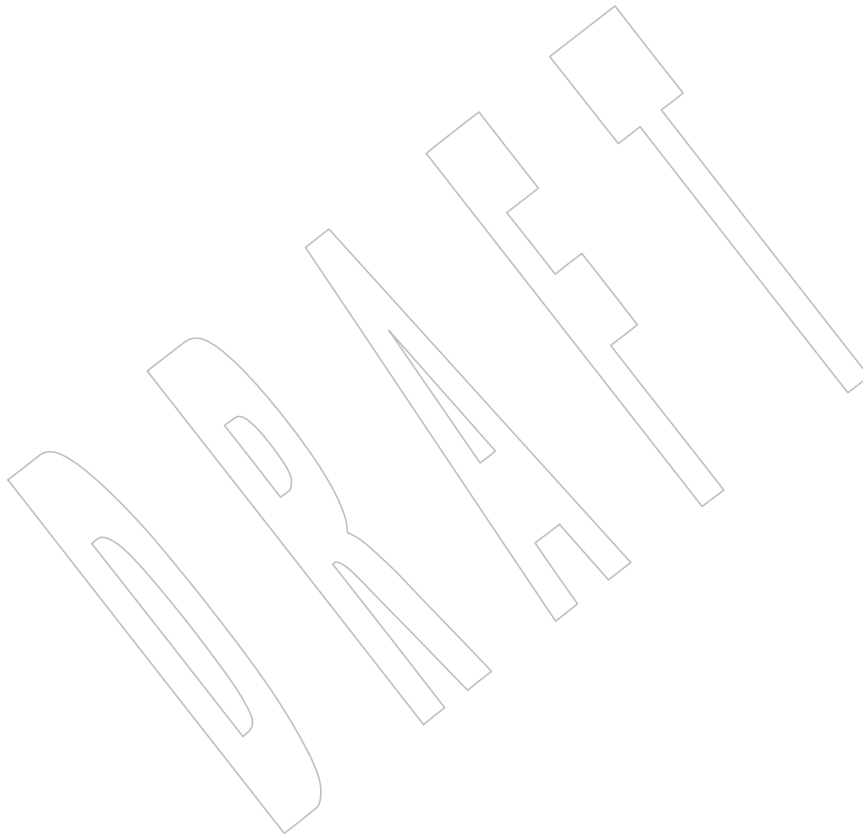
32280 **SEE ALSO**32281 *ferror()*, *fopen()*, *fprintf()*, *putc()*, *puts()*, *write*

32282 XBD <stdio.h>

32283 **CHANGE HISTORY**

32284 First released in Issue 1. Derived from Issue 1 of the SVID.

- 32285 **Issue 6**
- 32286 Extensions beyond the ISO C standard are marked.
- 32287 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:
- 32288 • The *fwrite()* prototype is updated.
 - 32289 • The DESCRIPTION is updated to clarify how the data is written out using *fputc()*.
- 32290 **Issue 7**
- 32291 Changes are made related to support for finegrained timestamps. +



32292 NAME

32293 fwscanf, swscanf, wscanf — convert formatted wide-character input

32294 SYNOPSIS

32295 #include <stdio.h>

32296 #include <wchar.h>

32297 int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ...); |

32298 int swscanf(const wchar_t *restrict ws, |

32299 const wchar_t *restrict format, ...); |

32300 int wscanf(const wchar_t *restrict format, ...); |

32301 DESCRIPTION

32302 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32303 conflict between the requirements described here and the ISO C standard is unintentional. This
32304 volume of POSIX.1-200x defers to the ISO C standard.32305 The *fwscanf()* function shall read from the named input *stream*. The *wscanf()* function shall read
32306 from the standard input stream *stdin*. The *swscanf()* function shall read from the wide-character
32307 string *ws*. Each function reads wide characters, interprets them according to a format, and stores
32308 the results in its arguments. Each expects, as arguments, a control wide-character string *format*
32309 described below, and a set of *pointer* arguments indicating where the converted input should be
32310 stored. The result is undefined if there are insufficient arguments for the format. If the *format* is
32311 exhausted while arguments remain, the excess arguments are evaluated but are otherwise
32312 ignored.32313 CX Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than
32314 to the next unused argument. In this case, the conversion specifier wide character % (see below)
32315 is replaced by the sequence "%n\$", where *n* is a decimal integer in the range
32316 [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that
32317 select arguments in an order appropriate to specific languages. In *format* wide-character strings
32318 containing the "%n\$" form of conversion specifications, it is unspecified whether numbered
32319 arguments in the argument list can be referenced from the *format* wide-character string more
32320 than once.32321 The *format* can contain either form of a conversion specification—that is, % or "%n\$"—but the
32322 two forms cannot normally be mixed within a single *format* wide-character string. The only
32323 exception to this is that %% or %* can be mixed with the "%n\$" form. When numbered argument
32324 specifications are used, specifying the *N*th argument requires that all the leading arguments,
32325 from the first to the (*N*−1)th, are pointers.32326 The *fwscanf()* function in all its forms allows for detection of a language-dependent radix
32327 character in the input string, encoded as a wide-character value. The radix character is defined
32328 in the locale of the process (category *LC_NUMERIC*). In the POSIX locale, or in a locale where
32329 the radix character is not defined, the radix character shall default to a period (' . ').32330 The *format* is a wide-character string composed of zero or more directives. Each directive is
32331 composed of one of the following: one or more white-space wide characters (<space>s, <tab>s,
32332 <newline>s, <vertical-tab>s, or <form-feed>s); an ordinary wide character (neither ' % ' nor a
32333 white-space character); or a conversion specification.32334 CX Each conversion specification is introduced by the ' % ' or by the character sequence "%n\$",
32335 after which the following appear in sequence:

- 32336
- An optional assignment-suppressing character ' * '.

32337		<ul style="list-style-type: none"> • An optional non-zero decimal integer that specifies the maximum field width. 	
32338	CX	<ul style="list-style-type: none"> • An optional assignment-allocation character 'm'. 	+
32339		<ul style="list-style-type: none"> • An optional length modifier that specifies the size of the receiving object. 	
32340		<ul style="list-style-type: none"> • A conversion specifier wide character that specifies the type of conversion to be applied. 	
32341		The valid conversion specifiers are described below.	
32342		The <i>fwscanf()</i> functions shall execute each directive of the format in turn. If a directive fails, as	
32343		detailed below, the function shall return. Failures are described as input failures (due to the	
32344		unavailability of input bytes) or matching failures (due to inappropriate input).	
32345		A directive composed of one or more white-space wide characters is executed by reading input	
32346		until no more valid input can be read, or up to the first wide character which is not a white-	
32347		space wide character, which remains unread.	
32348		A directive that is an ordinary wide character shall be executed as follows. The next wide	
32349		character is read from the input and compared with the wide character that comprises the	
32350		directive; if the comparison shows that they are not equivalent, the directive shall fail, and the	
32351		differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding	
32352		error, or a read error prevents a wide character from being read, the directive shall fail.	
32353		A directive that is a conversion specification defines a set of matching input sequences, as	
32354		described below for each conversion wide character. A conversion specification is executed in	
32355		the following steps.	
32356		Input white-space wide characters (as specified by <i>isspace()</i>) shall be skipped, unless the	
32357		conversion specification includes a <i>l</i> , <i>c</i> , or <i>n</i> conversion specifier.	
32358		An item shall be read from the input, unless the conversion specification includes an <i>n</i>	
32359		conversion specifier wide character. An input item is defined as the longest sequence of input	
32360		wide characters, not exceeding any specified field width, which is an initial subsequence of a	
32361		matching sequence. The first wide character, if any, after the input item shall remain unread. If	
32362		the length of the input item is zero, the execution of the conversion specification shall fail; this	
32363		condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented	
32364		input from the stream, in which case it is an input failure.	
32365		Except in the case of a <i>%</i> conversion specifier, the input item (or, in the case of a <i>%n</i> conversion	
32366		specification, the count of input wide characters) shall be converted to a type appropriate to the	
32367		conversion wide character. If the input item is not a matching sequence, the execution of the	
32368		conversion specification shall fail; this condition is a matching failure. Unless assignment	
32369		suppression was indicated by a <i>'*'</i> , the result of the conversion shall be placed in the object	
32370		pointed to by the first argument following the <i>format</i> argument that has not already received a	
32371	CX	conversion result if the conversion specification is introduced by <i>%</i> , or in the <i>n</i> th argument if	
32372		introduced by the wide-character sequence " <i>%n\$</i> ". If this object does not have an appropriate	
32373		type, or if the result of the conversion cannot be represented in the space provided, the behavior	
32374		is undefined.	
32375	CX	The <i>%s</i> , <i>%S</i> , and <i>%[</i> conversion specifiers shall accept an optional assignment-allocation	+
32376		character 'm', which shall cause a memory buffer to be allocated to hold the wide-character	+
32377		string converted including a terminating null wide character. In such a case, the argument	+
32378		corresponding to the conversion specifier should be a reference to a pointer value that will	+
32379		receive a pointer to the allocated buffer. The system shall allocate a buffer as if <i>malloc()</i> had been	+
32380		called. The application shall be responsible for freeing the memory after usage. If there is	+
32381		insufficient memory to allocate a buffer, the function shall set <i>errno</i> to [ENOMEM] and a	+
32382		conversion error shall result. In that case, any memory successfully allocated for other	+
32383		parameters using assignment-allocation character 'm' by this call shall be freed.	+
32384		The length modifiers and their meanings are:	

32385	hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or unsigned char .
32386		
32387	h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short or unsigned short .
32388		
32389	l (ell)	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long or unsigned long ; that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to double ; or that a following c, s, or [conversion specifier applies to an argument with type pointer to wchar_t . If the 'm' assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to wchar_t .
32390		
32391		
32392		
32393	CX	
32394		
32395	ll (ell-ell)	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long or unsigned long long .
32396		
32397		
32398	j	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to intmax_t or uintmax_t .
32399		
32400	z	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to size_t or the corresponding signed integer type.
32401		
32402	t	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to ptrdiff_t or the corresponding unsigned type.
32403		
32404	L	Specifies that a following a, A, e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to long double .
32405		
32406		If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.
32407		
32408		The following conversion specifier wide characters are valid:
32409	d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>wcstol()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
32410		
32411		
32412		
32413	i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <i>wcstol()</i> with 0 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to int .
32414		
32415		
32416		
32417	o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 8 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
32418		
32419		
32420		
32421	u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 10 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
32422		
32423		
32424		
32425	x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <i>wcstoul()</i> with the value 16 for the <i>base</i> argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to unsigned .
32426		
32427		
32428		
32429	a, e, f, g	Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of <i>wcstod()</i> . In the absence of a size
32430		
32431		

32432		modifier, the application shall ensure that the corresponding argument is a pointer to
32433		float .
32434		If the <i>fwprintf()</i> family of functions generates character string representations for
32435		infinity and NaN (a symbolic entity encoded in floating-point format) to support
32436		IEEE Std 754-1985, the <i>fwscanf()</i> family of functions shall recognize them as input.
32437	s	Matches a sequence of non-white-space wide characters. If no 1 (ell) qualifier is present,
32438		characters from the input field shall be converted as if by repeated calls to the
32439		<i>wcrtomb()</i> function, with the conversion state described by an mbstate_t object
32440		initialized to zero before the first wide character is converted. If the 'm' assignment-
32441		allocation character is not specified, the application shall ensure that the corresponding
32442		argument is a pointer to a character array large enough to accept the sequence and the
32443	CX	terminating null character, which shall be added automatically. Otherwise, the
32444		application shall ensure that the corresponding argument is a pointer to a pointer to a
32445		wchar_t .
32446		If the 1 (ell) qualifier is present and the 'm' assignment-allocation character is not
32447		specified, the application shall ensure that the corresponding argument is a pointer to
32448		an array of wchar_t large enough to accept the sequence and the terminating null wide
32449	CX	character, which shall be added automatically. If the 1 (ell) qualifier is present and the
32450		'm' assignment-allocation character is present, the application shall ensure that the
32451		corresponding argument is a pointer to a pointer to a wchar_t .
32452	[Matches a non-empty sequence of wide characters from a set of expected wide
32453		characters (the <i>scanset</i>). If no 1 (ell) qualifier is present, wide characters from the input
32454		field shall be converted as if by repeated calls to the <i>wcrtomb()</i> function, with the
32455		conversion state described by an mbstate_t object initialized to zero before the first
32456		wide character is converted. If the 'm' assignment-allocation character is not specified,
32457		the application shall ensure that the corresponding argument is a pointer to a character
32458		array large enough to accept the sequence and the terminating null character, which
32459	CX	shall be added automatically. Otherwise, the application shall ensure that the
32460		corresponding argument is a pointer to a pointer to a wchar_t .
32461		If an 1 (ell) qualifier is present and the 'm' assignment-allocation character is not
32462		specified, the application shall ensure that the corresponding argument is a pointer to
32463		an array of wchar_t large enough to accept the sequence and the terminating null wide
32464	CX	character. If an 1 (ell) qualifier is present and the 'm' assignment-allocation character
32465		is specified, the application shall ensure that the corresponding argument is a pointer to
32466		a pointer to a wchar_t .
32467		The conversion specification includes all subsequent wide characters in the <i>format</i>
32468		string up to and including the matching right square bracket (']'). The wide
32469		characters between the square brackets (the <i>scanlist</i>) comprise the scanset, unless the
32470		wide character after the left square bracket is a circumflex ('^'), in which case the
32471		scanset contains all wide characters that do not appear in the scanlist between the
32472		circumflex and the right square bracket. If the conversion specification begins with
32473		"[]" or "[^]", the right square bracket is included in the scanlist and the next right
32474		square bracket is the matching right square bracket that ends the conversion
32475		specification; otherwise, the first right square bracket is the one that ends the
32476		conversion specification. If a '-' is in the scanlist and is not the first wide character,
32477		nor the second where the first wide character is a '^', nor the last wide character, the
32478		behavior is implementation-defined.
32479	c	Matches a sequence of wide characters of exactly the number specified by the field
32480		width (1 if no field width is present in the conversion specification).
32481		If no 1 (ell) length modifier is present, characters from the input field shall be converted

32482			as if by repeated calls to the <i>wcrtomb()</i> function, with the conversion state described by
32483			an mbstate_t object initialized to zero before the first wide character is converted. The
32484			corresponding argument shall be a pointer to the initial element of a character array
32485			large enough to accept the sequence. No null character is added.
32486			No null wide character is added. If an l (ell) length modifier is present and the 'm'
32487			assignment-allocation character is not specified, the application shall ensure that the
32488			corresponding argument shall be a pointer to the initial element of an array of wchar_t
32489	CX		large enough to accept the sequence. If an l (ell) qualifier is present and the 'm'
32490			assignment-allocation character is specified, the application shall ensure that the
32491			corresponding argument is a pointer to a pointer to a wchar_t .
32492		p	Matches an implementation-defined set of sequences, which shall be the same as the set
32493			of sequences that is produced by the %p conversion specification of the corresponding
32494			<i>fwprintf()</i> functions. The application shall ensure that the corresponding argument is a
32495			pointer to a pointer to void . The interpretation of the input item is implementation-
32496			defined. If the input item is a value converted earlier during the same program
32497			execution, the pointer that results shall compare equal to that value; otherwise, the
32498			behavior of the %p conversion is undefined.
32499		n	No input is consumed. The application shall ensure that the corresponding argument is
32500			a pointer to the integer into which is to be written the number of wide characters read
32501			from the input so far by this call to the <i>fwscanf()</i> functions. Execution of a %n
32502			conversion specification shall not increment the assignment count returned at the
32503			completion of execution of the function. No argument shall be converted, but one shall
32504			be consumed. If the conversion specification includes an assignment-suppressing wide
32505			character or a field width, the behavior is undefined.
32506	XSI	C	Equivalent to lc .
32507	XSI	S	Equivalent to ls .
32508		%	Matches a single '%' wide character; no conversion or assignment shall occur. The
32509			complete conversion specification shall be %% .
32510			If a conversion specification is invalid, the behavior is undefined.
32511			The conversion specifiers A , E , F , G , and X are also valid and shall be equivalent to, respectively,
32512			a , e , f , g , and x .
32513			If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before
32514			any wide characters matching the current conversion specification (except for %n) have been
32515			read (other than leading white-space, where permitted), execution of the current conversion
32516			specification shall terminate with an input failure. Otherwise, unless execution of the current
32517			conversion specification is terminated with a matching failure, execution of the following
32518			conversion specification (if any) shall be terminated with an input failure.
32519			Reaching the end of the string in <i>swscanf()</i> shall be equivalent to encountering end-of-file for
32520			<i>fwscanf()</i> .
32521			If conversion terminates on a conflicting input, the offending input shall be left unread in the
32522			input. Any trailing white space (including <newline>) shall be left unread unless matched by a
32523			conversion specification. The success of literal matches and suppressed assignments is only
32524			directly determinable via the %n conversion specification.
32525	CX		The <i>fwscanf()</i> and <i>wscanf()</i> functions may mark the last data access timestamp of the file
32526			associated with <i>stream</i> for update. The last data access timestamp shall be marked for update by
32527			the first successful execution of <i>fgetc()</i> , <i>fgetwc()</i> , <i>fgets()</i> , <i>fgetws()</i> , <i>fread()</i> , <i>getc()</i> , <i>getwc()</i> ,
32528			<i>getchar()</i> , <i>getwchar()</i> , <i>gets()</i> , <i>fscanf()</i> , or <i>fwscanf()</i> using <i>stream</i> that returns data not supplied by a
32529			prior call to <i>ungetc()</i> .

fwscanf()*System Interfaces*32530 **RETURN VALUE**

32531 Upon successful completion, these functions shall return the number of successfully matched
 32532 and assigned input items; this number can be zero in the event of an early matching failure. If
 32533 the input ends before the first matching failure or conversion, EOF shall be returned. If a read
 32534 error occurs, the error indicator for the stream is set, EOF shall be returned, and *errno* shall be
 32535 set to indicate the error.

32536 **ERRORS**

32537 For the conditions under which the *fwscanf()* functions shall fail and may fail, refer to *fgetwc()*.

32538 In addition, the *fwscanf()* functions shall fail if:

32539 [ENOMEM] Insufficient storage space is available.

32540 The *fwscanf()* functions may fail if:

32541 CX [EILSEQ] Input byte sequence does not form a valid character.

32542 CX [EINVAL] There are insufficient arguments.

32543 **EXAMPLES**

32544 The call:

```
32545 int i, n; float x; char name[50];
32546 n = wscanf(L"%d%f%s", &i, &x, name);
```

32547 with the input line:

```
32548 25 54.32E-1 Hamster
```

32549 assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string
 32550 "Hamster".

32551 The call:

```
32552 int i; float x; char name[50];
32553 (void) wscanf(L"%2d%f*d %[0123456789]", &i, &x, name);
```

32554 with input:

```
32555 56789 0123 56a72
```

32556 assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string "56\0" in *name*. The next call to
 32557 *getchar()* shall return the character 'a'.

32558 **APPLICATION USAGE**

32559 In format strings containing the '%' form of conversion specifications, each argument in the
 32560 argument list is used exactly once.

32561 **RATIONALE**

32562 None.

32563 **FUTURE DIRECTIONS**

32564 None.

32565 **SEE ALSO**

32566 *getwc()*, *fwprintf()*, *setlocale()*, *wcstod()*, *wcstol()*, *wcstoul()*, *wcrtomb()*

32567 XBD Chapter 7 (on page 121), <*langinfo.h*>, <*stdio.h*>, <*wchar.h*>

32568 **CHANGE HISTORY**

32569 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 32570 (E).

32571

Issue 6

32572

The normative text is updated to avoid use of the term “must” for application requirements.

32573

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

32574

- The prototypes for *fwscanf()* and *swscanf()* are updated.

32575

- The DESCRIPTION is updated.

32576

- The hh, ll, j, t, and z length modifiers are added.

32577

- The a, A, and F conversion characters are added.

32578

The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” consistently.

32579

32580

Issue 7

32581

SD5-XSH-ERN-132 is applied, adding the assignment-allocation character ‘m’.

+

32582

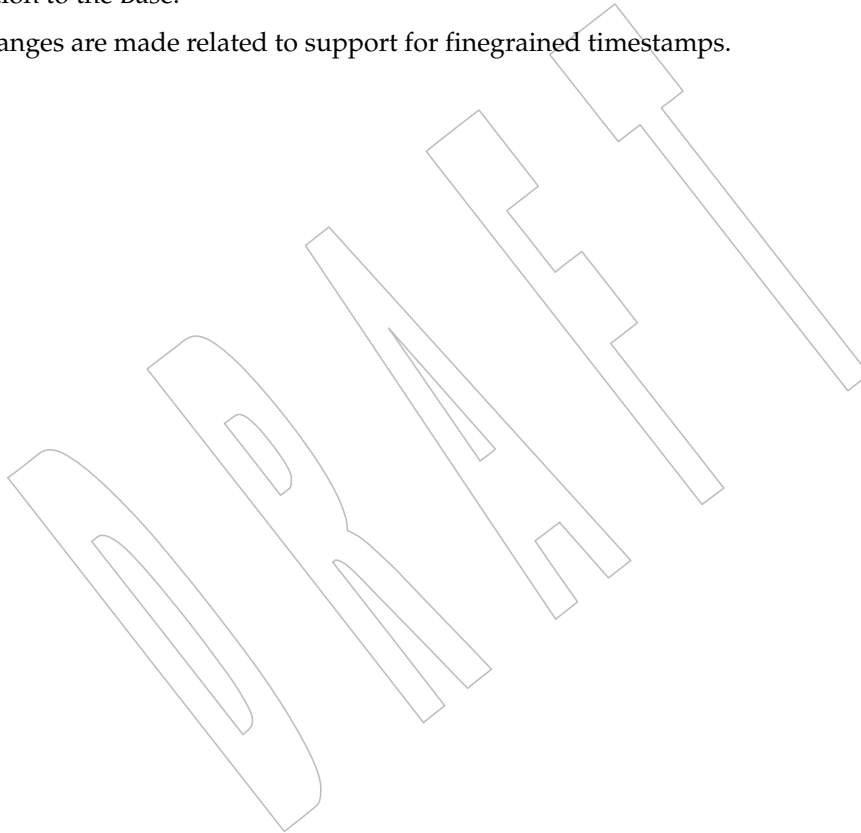
Functionality relating to the “%n\$” form of conversion specification is moved from the XSI option to the Base.

32583

32584

Changes are made related to support for finegrained timestamps.

+



32585 **NAME**32586 `gai_strerror` — address and name information error description32587 **SYNOPSIS**32588 `#include <netdb.h>`32589 `const char *gai_strerror(int ecode);`32590 **DESCRIPTION**32591 The `gai_strerror()` function shall return a text string describing an error value for the `getaddrinfo()`
32592 and `getnameinfo()` functions listed in the `<netdb.h>` header.32593 When the `ecode` argument is one of the following values listed in the `<netdb.h>` header:

32594	[EAI_AGAIN]	[EAI_NONAME]
32595	[EAI_BADFLAGS]	[EAI_OVERFLOW]
32596	[EAI_FAIL]	[EAI_SERVICE]
32597	[EAI_FAMILY]	[EAI_SOCKTYPE]
32598	[EAI_MEMORY]	[EAI_SYSTEM]

32599 the function return value shall point to a string describing the error. If the argument is not one
32600 of those values, the function shall return a pointer to a string whose contents indicate an
32601 unknown error.32602 **RETURN VALUE**32603 Upon successful completion, `gai_strerror()` shall return a pointer to an implementation-defined
32604 string.32605 **ERRORS**

32606 No errors are defined.

32607 **EXAMPLES**

32608 None.

32609 **APPLICATION USAGE**

32610 None.

32611 **RATIONALE**

32612 None.

32613 **FUTURE DIRECTIONS**

32614 None.

32615 **SEE ALSO**32616 [*freeaddrinfo\(\)*](#)32617 XBD `<netdb.h>`32618 **CHANGE HISTORY**

32619 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

32620 The Open Group Base Resolution bwg2001-009 is applied, which changes the return type from
32621 `char *` to `const char *`. This is for coordination with the IPnG Working Group.32622 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/22 is applied, adding the
32623 [EAI_OVERFLOW] error code.

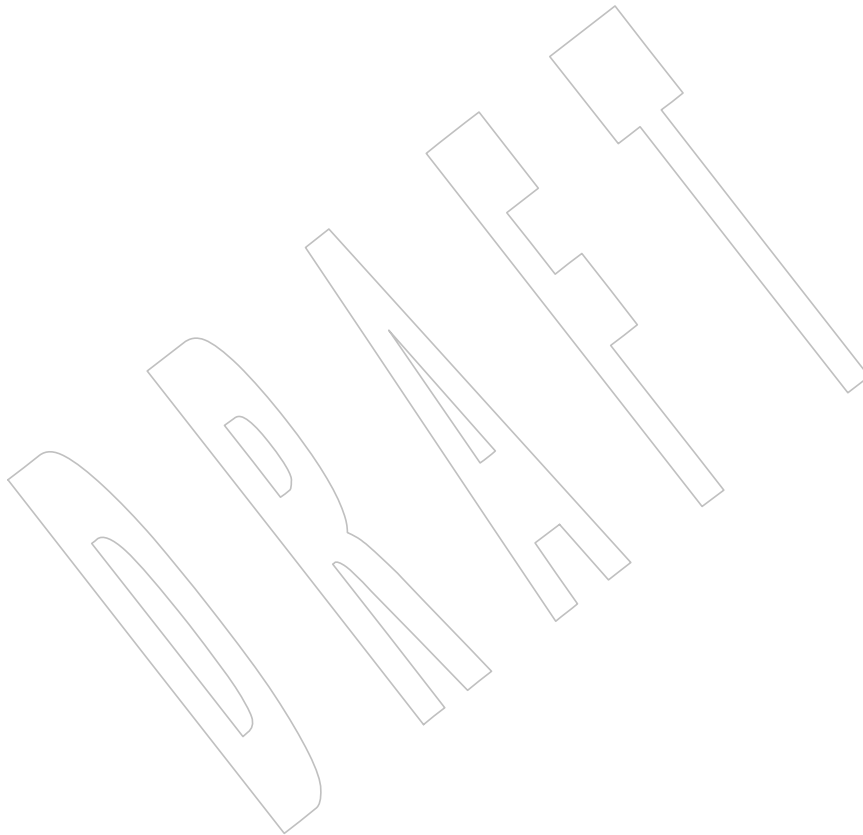
32624 **NAME**
32625 getaddrinfo — get address information

32626 **SYNOPSIS**

```
32627       #include <sys/socket.h>  
32628       #include <netdb.h>  
  
32629       int getaddrinfo(const char *restrict nodename,  
32630                      const char *restrict servname,  
32631                      const struct addrinfo *restrict hints,  
32632                      struct addrinfo **restrict res);
```

32633 **DESCRIPTION**

32634 Refer to *freeaddrinfo()*.



32635 **NAME**

32636 getc — get a byte from a stream

32637 **SYNOPSIS**

32638 #include <stdio.h>

32639 int getc(FILE *stream);

32640 **DESCRIPTION**32641 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32642 conflict between the requirements described here and the ISO C standard is unintentional. This
32643 volume of POSIX.1-200x defers to the ISO C standard.32644 The *getc()* function shall be equivalent to *fgetc()*, except that if it is implemented as a macro it
32645 may evaluate *stream* more than once, so the argument should never be an expression with side
32646 effects.32647 **RETURN VALUE**32648 Refer to *fgetc()*.32649 **ERRORS**32650 Refer to *fgetc()*.32651 **EXAMPLES**

32652 None.

32653 **APPLICATION USAGE**32654 If the integer value returned by *getc()* is stored into a variable of type **char** and then compared
32655 against the integer constant EOF, the comparison may never succeed, because sign-extension of
32656 a variable of type **char** on widening to integer is implementation-defined.32657 Since it may be implemented as a macro, *getc()* may treat incorrectly a *stream* argument with
32658 side effects. In particular, *getc(*f++)* does not necessarily work as expected. Therefore, use of this
32659 function should be preceded by "#undef getc" in such situations; *fgetc()* could also be used.32660 **RATIONALE**

32661 None.

32662 **FUTURE DIRECTIONS**

32663 None.

32664 **SEE ALSO**32665 *fgetc()*

32666 XBD <stdio.h>

32667 **CHANGE HISTORY**

32668 First released in Issue 1. Derived from Issue 1 of the SVID.

32669 **NAME**

32670 getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked — stdio with explicit client
32671 locking

32672 **SYNOPSIS**

```
32673 CX      #include <stdio.h>
32674
32675      int getc_unlocked(FILE *stream);
32676      int getchar_unlocked(void);
32677      int putc_unlocked(int c, FILE *stream);
32678      int putchar_unlocked(int c);
```

32678 **DESCRIPTION**

32679 Versions of the functions *getc()*, *getchar()*, *putc()*, and *putchar()* respectively named
32680 *getc_unlocked()*, *getchar_unlocked()*, *putc_unlocked()*, and *putchar_unlocked()* shall be provided
32681 which are functionally equivalent to the original versions, with the exception that they are not
32682 required to be implemented in a thread-safe manner. They may only safely be used within a
32683 scope protected by *flockfile()* (or *ftrylockfile()*) and *funlockfile()*. These functions may safely be
32684 used in a multi-threaded program if and only if they are called while the invoking thread owns
32685 the (FILE *) object, as is the case after a successful call to the *flockfile()* or *ftrylockfile()* functions.

32686 **RETURN VALUE**

32687 See *getc()*, *getchar()*, *putc()*, and *putchar()*.

32688 **ERRORS**

32689 See *getc()*, *getchar()*, *putc()*, and *putchar()*.

32690 **EXAMPLES**

32691 None.

32692 **APPLICATION USAGE**

32693 Since they may be implemented as macros, *getc_unlocked()* and *putc_unlocked()* may treat
32694 incorrectly a *stream* argument with side effects. In particular, *getc_unlocked>(*f++)* and
32695 *putc_unlocked(*f++)* do not necessarily work as expected. Therefore, use of these functions in
32696 such situations should be preceded by the following statement as appropriate:

```
32697      #undef getc_unlocked
32698      #undef putc_unlocked
```

32699 **RATIONALE**

32700 Some I/O functions are typically implemented as macros for performance reasons (for example,
32701 *putc()* and *getc()*). For safety, they need to be synchronized, but it is often too expensive to
32702 synchronize on every character. Nevertheless, it was felt that the safety concerns were more
32703 important; consequently, the *getc()*, *getchar()*, *putc()*, and *putchar()* functions are required to be
32704 thread-safe. However, unlocked versions are also provided with names that clearly indicate the
32705 unsafe nature of their operation but can be used to exploit their higher performance. These
32706 unlocked versions can be safely used only within explicitly locked program regions, using
32707 exported locking primitives. In particular, a sequence such as:

```
32708      flockfile(fileptr);
32709      putc_unlocked('1', fileptr);
32710      putc_unlocked('\n', fileptr);
32711      fprintf(fileptr, "Line 2\n");
32712      funlockfile(fileptr);
```

32713 is permissible, and results in the text sequence:

32714 1
32715 Line 2

32716 being printed without being interspersed with output from other threads.

32717 It would be wrong to have the standard names such as *getc()*, *putc()*, and so on, map to the
32718 “faster, but unsafe” rather than the “slower, but safe” versions. In either case, you would still
32719 want to inspect all uses of *getc()*, *putc()*, and so on, by hand when converting existing code.
32720 Choosing the safe bindings as the default, at least, results in correct code and maintains the
32721 “atomicity at the function” invariant. To do otherwise would introduce gratuitous
32722 synchronization errors into converted code. Other routines that modify the *stdio* (**FILE ***)
32723 structures or buffers are also safely synchronized.

32724 Note that there is no need for functions of the form *getc_locked()*, *putc_locked()*, and so on, since
32725 this is the functionality of *getc()*, *putc()*, *et al.* It would be inappropriate to use a feature test
32726 macro to switch a macro definition of *getc()* between *getc_locked()* and *getc_unlocked()*, since the
32727 ISO C standard requires an actual function to exist, a function whose behavior could not be
32728 changed by the feature test macro. Also, providing both the *xxx_locked()* and *xxx_unlocked()*
32729 forms leads to the confusion of whether the suffix describes the behavior of the function or the
32730 circumstances under which it should be used.

32731 Three additional routines, *flockfile()*, *ftrylockfile()*, and *funlockfile()* (which may be macros), are
32732 provided to allow the user to delineate a sequence of I/O statements that are executed
32733 synchronously.

32734 The *ungetc()* function is infrequently called relative to the other functions/macros so no
32735 unlocked variation is needed.

32736 FUTURE DIRECTIONS

32737 None.

32738 SEE ALSO

32739 *getc()*, *getchar()*, *putc()*, *putchar()*

32740 XBD [<stdio.h>](#)

32741 CHANGE HISTORY

32742 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

32743 Issue 6

32744 These functions are marked as part of the Thread-Safe Functions option.

32745 The Open Group Corrigendum U030/2 is applied, adding APPLICATION USAGE describing
32746 how applications should be written to avoid the case when the functions are implemented as
32747 macros.

32748 Issue 7

32749 The *getc_unlocked()*, *getchar_unlocked()*, *putc_unlocked()*, and *putchar_unlocked()* functions are
32750 moved from the Thread-Safe Functions option to the Base.

32751 **NAME**

32752 getchar — get a byte from a stdin stream

32753 **SYNOPSIS**

32754 #include <stdio.h>

32755 int getchar(void);

32756 **DESCRIPTION**32757 CX The functionality described on this reference page is aligned with the ISO C standard. Any
32758 conflict between the requirements described here and the ISO C standard is unintentional. This
32759 volume of POSIX.1-200x defers to the ISO C standard.32760 The *getchar()* function shall be equivalent to *getc(stdin)*.32761 **RETURN VALUE**32762 Refer to *fgetc()*.32763 **ERRORS**32764 Refer to *fgetc()*.32765 **EXAMPLES**

32766 None.

32767 **APPLICATION USAGE**32768 If the integer value returned by *getchar()* is stored into a variable of type **char** and then
32769 compared against the integer constant EOF, the comparison may never succeed, because sign-
32770 extension of a variable of type **char** on widening to integer is implementation-defined.32771 **RATIONALE**

32772 None.

32773 **FUTURE DIRECTIONS**

32774 None.

32775 **SEE ALSO**32776 *getc()*

32777 XBD <stdio.h>

32778 **CHANGE HISTORY**

32779 First released in Issue 1. Derived from Issue 1 of the SVID.

getchar_unlocked()*System Interfaces*

32780 **NAME**
32781 getchar_unlocked — stdio with explicit client locking

32782 **SYNOPSIS**

32783 CX #include <stdio.h>
32784 int getchar_unlocked(void);

32785 **DESCRIPTION**

32786 Refer to *getc_unlocked()*.

32787 **NAME**

32788 getcwd — get the pathname of the current working directory

32789 **SYNOPSIS**

32790 #include <unistd.h>

32791 char *getcwd(char *buf, size_t size);

32792 **DESCRIPTION**

32793 The *getcwd()* function shall place an absolute pathname of the current working directory in the
 32794 array pointed to by *buf*, and return *buf*. The pathname copied to the array shall contain no
 32795 components that are symbolic links. The *size* argument is the size in bytes of the character array
 32796 pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd()* is unspecified.

32797 **RETURN VALUE**

32798 Upon successful completion, *getcwd()* shall return the *buf* argument. Otherwise, *getcwd()* shall
 32799 return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by
 32800 *buf* are then undefined.

32801 **ERRORS**32802 The *getcwd()* function shall fail if:32803 [EINVAL] The *size* argument is 0.32804 [ERANGE] The *size* argument is greater than 0, but is smaller than the length of the
32805 pathname +1.32806 The *getcwd()* function may fail if:

32807 [EACCES] Read or search permission was denied for a component of the pathname.

32808 [ENOMEM] Insufficient storage space is available.

32809 **EXAMPLES**32810 **Determining the Absolute Pathname of the Current Working Directory**

32811 The following example returns a pointer to an array that holds the absolute pathname of the
 32812 current working directory. The pointer is returned in the *ptr* variable, which points to the *buf*
 32813 array where the pathname is stored.

```
32814       #include <stdlib.h>
32815       #include <unistd.h>
32816       ...
32817       long size;
32818       char *buf;
32819       char *ptr;

32820       size = pathconf(".", _PC_PATH_MAX);
32821       if ((buf = (char *)malloc((size_t)size)) != NULL)
32822           ptr = getcwd(buf, (size_t)size);
32823       ...
```

32824 **APPLICATION USAGE**

32825 None.

RATIONALE

Since the maximum pathname length is arbitrary unless `{PATH_MAX}` is defined, an application generally cannot supply a *buf* with *size* `{{PATH_MAX}+1}`.

Having `getcwd()` take no arguments and instead use the `malloc()` function to produce space for the returned argument was considered. The advantage is that `getcwd()` knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the `free()` function to free the resulting object, or each use of `getcwd()` would further reduce the available memory. Also, `malloc()` and `free()` are used nowhere else in this volume of POSIX.1-200x. Finally, `getcwd()` is taken from the SVID where it has the two arguments used in this volume of POSIX.1-200x.

The older function `getwd()` was rejected for use in this context because it had only a buffer argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if *buf* is a null pointer, `getcwd()` may obtain *size* bytes of memory using `malloc()`. In this case, the pointer returned by `getcwd()` may be used as the argument in a subsequent call to `free()`. Invoking `getcwd()` with *buf* as a null pointer is not recommended in conforming applications.

If a program is operating in a directory where some (grand)parent directory does not permit reading, `getcwd()` may fail, as in most implementations it must read the directory to determine the name of the file. This can occur if search, but not read, permission is granted in an intermediate directory, or if the program is placed in that directory by some more privileged process (for example, login). Including the `[EACCES]` error condition makes the reporting of the error consistent and warns the application developer that `getcwd()` can fail for reasons beyond the control of the application developer or user. Some implementations can avoid this occurrence (for example, by implementing `getcwd()` using `pwd`, where `pwd` is a set-user-root process), thus the error was made optional. Since this volume of POSIX.1-200x permits the addition of other errors, this would be a common addition and yet one that applications could not be expected to deal with without this addition.

FUTURE DIRECTIONS

None.

SEE ALSO

[*malloc\(\)*](#)

XBD [`<unistd.h>`](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The `[ENOMEM]` optional error condition is added.

32865 **NAME**

32866 getdate — convert user format date and time

32867 **SYNOPSIS**

```
32868 XSI #include <time.h>
32869 struct tm *getdate(const char *string);
```

32870 **DESCRIPTION**

32871 The *getdate()* function shall convert a string representation of a date or time into a broken-down
 32872 time.

32873 The external variable or macro *getdate_err* is used by *getdate()* to return error values.

32874 Templates are used to parse and interpret the input string. The templates are contained in a text
 32875 file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to
 32876 indicate the full pathname of the file that contains the templates. The first line in the template
 32877 that matches the input specification is used for interpretation and conversion into the internal
 32878 time format.

32879 The following conversion specifications shall be supported:

32880	%%	Equivalent to %.
32881	%a	Abbreviated weekday name.
32882	%A	Full weekday name.
32883	%b	Abbreviated month name.
32884	%B	Full month name.
32885	%c	Locale's appropriate date and time representation.
32886	%C	Century number [00,99]; leading zeros are permitted but not required.
32887	%d	Day of month [01,31]; the leading 0 is optional.
32888	%D	Date as %m/%d/%y.
32889	%e	Equivalent to %d.
32890	%h	Abbreviated month name.
32891	%H	Hour [00,23].
32892	%I	Hour [01,12].
32893	%m	Month number [01,12].
32894	%M	Minute [00,59].
32895	%n	Equivalent to <newline>.
32896	%p	Locale's equivalent of either AM or PM.
32897	%r	The locale's appropriate representation of time in AM and PM notation. In the POSIX 32898 locale, this shall be equivalent to %I:%M:%S %p.
32899	%R	Time as %H:%M.
32900	%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap 32901 seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap 32902 second data must come from some external source.

32903	%t	Equivalent to <tab>.
32904	%T	Time as %H:%M:%S.
32905	%w	Weekday number (Sunday = [0,6]).
32906	%x	Locale's appropriate date representation.
32907	%X	Locale's appropriate time representation.
32908	%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.
32911	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
32912		
32913		
32914	%Y	Year as "ccyy" (for example, 2001).
32915	%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by %Z is not the timezone that <i>getdate()</i> expects, an invalid input specification error shall result. The <i>getdate()</i> function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).
32916		
32917		
32918		
32919		The match between the template and input specification performed by <i>getdate()</i> shall be case-insensitive.
32920		
32921		The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the <i>LC_TIME</i> category (see <i>setlocale()</i>).
32922		
32923		
32924		Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra whitespace in either the template file or in <i>string</i> shall be ignored.
32925		
32926		
32927		The results are undefined if the conversion specifications %c, %x, and %X include unsupported conversion specifications.
32928		
32929		The following rules apply for converting the input specification into the internal format:
32930		• If %Z is being scanned, then <i>getdate()</i> shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if <i>localtime()</i> had been called.
32931		
32932		
32933		• If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
32934		
32935		• If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
32936		
32937		
32938		• If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
32939		
32940		• If no date is given, the hour chosen shall be the hour, starting with the current hour and moving into the future, which first matches the named hour.
32941		
32942		If a conversion specification in the DATEMSK file does not correspond to one of the conversion specifications above, the behavior is unspecified.
32943		
32944		The <i>getdate()</i> function need not be thread-safe. A function that is not required to be thread-safe is not required to be reentrant.
32945		

32946
32947
32948**RETURN VALUE**

Upon successful completion, *getdate()* shall return a pointer to a **struct tm**. Otherwise, it shall return a null pointer and set *getdate_err* to indicate the error.

32949
32950
32951**ERRORS**

The *getdate()* function shall fail in the following cases, setting *getdate_err* to the value shown in the list below. Any changes to *errno* are unspecified.

32952
32953
32954
32955
32956
32957
32958
32959
32960

1. The *DATETIME* environment variable is null or undefined.
2. The template file cannot be opened for reading.
3. Failed to get file status information.
4. The template file is not a regular file.
5. An I/O error is encountered while reading the template file.
6. Memory allocation failed (not enough memory available).
7. There is no line in the template that matches the input.
8. Invalid input specification. For example, February 31; or a time is specified that cannot be represented in a **time_t** (representing the time in seconds since the Epoch).

32961

EXAMPLES32962
32963
32964
32965
32966
32967
32968
32969
32970
32971

1. The following example shows the possible contents of a template:

```
%m
%A %B %d, %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d, %m, %Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p, %B %dnd
%A den %d. %B %Y %H.%M Uhr
```

32972
32973
32974
32975
32976
32977
32978

2. The following are examples of valid input specifications for the template in Example 1:

```
getdate("10/1/87 4 PM");
getdate("Friday");
getdate("Friday September 18, 1987, 10:30:30");
getdate("24,9,1986 10:30");
getdate("at monday the 1st of december in 1986");
getdate("run job at 3 PM, december 2nd");
```

32979
32980

If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name and *oktober* as a month name, the following would be valid:

32981

```
getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

32982
32983

3. The following example shows how local date and time specification can be defined in the template:

32984
32985
32986
32987
32988

Invocation	Line in Template
<code>getdate("11/27/86")</code>	<code>%m/%d/%y</code>
<code>getdate("27.11.86")</code>	<code>%d.%m.%y</code>
<code>getdate("86-11-27")</code>	<code>%y-%m-%d</code>
<code>getdate("Friday 12:00:00")</code>	<code>%A %H:%M:%S</code>

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C locale:

Input	Line in Template	Date
Mon	%a	Mon Sep 22 12:19:47 EDT 1986
Sun	%a	Sun Sep 28 12:19:47 EDT 1986
Fri	%a	Fri Sep 26 12:19:47 EDT 1986
September	%B	Mon Sep 1 12:19:47 EDT 1986
January	%B	Thu Jan 1 12:19:47 EST 1987
December	%B	Mon Dec 1 12:19:47 EST 1986
Sep Mon	%b %a	Mon Sep 1 12:19:47 EDT 1986
Jan Fri	%b %a	Fri Jan 2 12:19:47 EST 1987
Dec Mon	%b %a	Mon Dec 1 12:19:47 EST 1986
Jan Wed 1989	%b %a %Y	Wed Jan 4 12:19:47 EST 1989
Fri 9	%a %H	Fri Sep 26 09:00:00 EDT 1986
Feb 10:30	%b %H:%S	Sun Feb 1 10:00:30 EST 1987
10:30	%H:%M	Tue Sep 23 10:30:00 EDT 1986
13:30	%H:%M	Mon Sep 22 13:30:00 EDT 1986

APPLICATION USAGE

Although historical versions of *getdate()* did not require that `<time.h>` declare the external variable *getdate_err*, this volume of POSIX.1-200x does require it. The standard developers encourage applications to remove declarations of *getdate_err* and instead incorporate the declaration by including `<time.h>`.

Applications should use %Y (4-digit years) in preference to %y (2-digit years).

RATIONALE

In standard locales, the conversion specifications %c, %x, and %X do not include unsupported conversion specifiers and so the text regarding results being undefined is not a problem in that case.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime(), *localtime()*, *setlocale()*, *strftime()*, *times()*

XBD `<time.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The last paragraph of the DESCRIPTION is added.

The %C conversion specification is added, and the exact meaning of the %y conversion specification is clarified in the DESCRIPTION.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

The %R conversion specification is changed to follow historical practice.

Issue 6

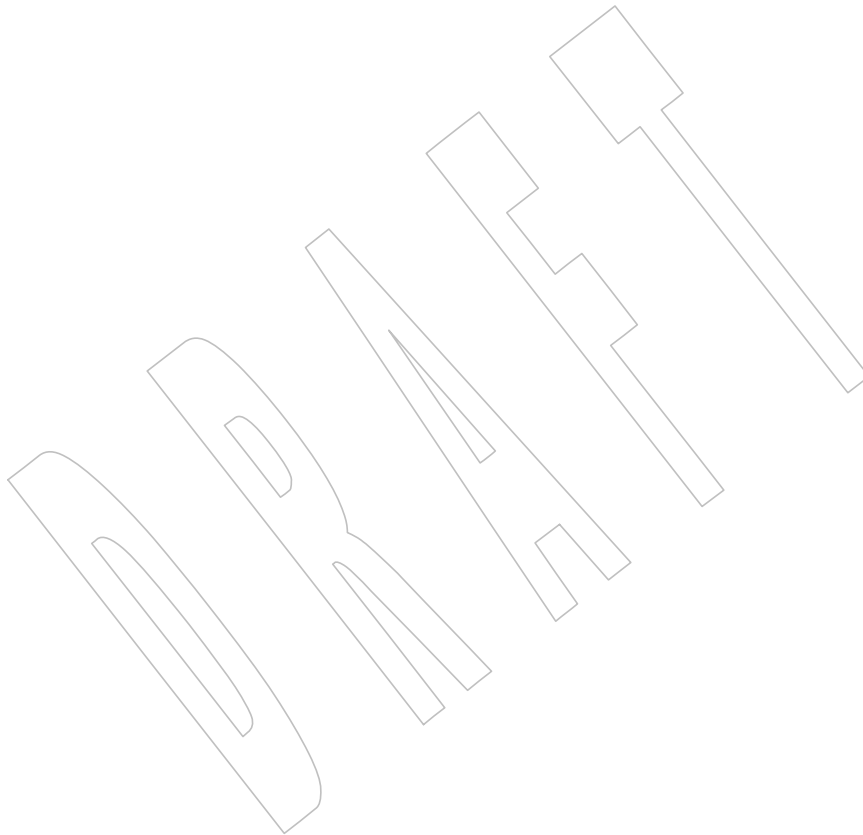
The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time* functions.

The description of %S is updated so that the valid range is [00,60] rather than [00,61].

33035

33036

The DESCRIPTION is updated to refer to conversion specifications instead of field descriptors for consistency with other functions.



33037 **NAME**33038 `getdelim, getline` — read a delimited record from *stream*33039 **SYNOPSIS**

```

33040 CX      #include <stdio.h>
33041         ssize_t getdelim(char **lineptr, size_t *n, int delimiter,
33042         FILE *stream);
33043         ssize_t getline(char **lineptr, size_t *n, FILE *stream);

```

33044 **DESCRIPTION**

33045 The `getdelim()` function shall read from *stream* until it encounters a character matching the
 33046 *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall
 33047 ensure is a character representable as an **unsigned char** of equal value that terminates the read
 33048 process. If the *delimiter* argument has any other value, the behavior is undefined.

33049 The application shall ensure that *lineptr* is a valid argument that could be passed to the `free()`
 33050 function. If **n* is non-zero, the application shall ensure that *lineptr* points to an object of size at
 33051 least **n* bytes.

33052 The size of the object pointed to by *lineptr* shall be increased to fit the incoming line, if it isn't
 33053 already large enough. The characters read shall be stored in the string pointed to by the *lineptr*
 33054 argument.

33055 The `getline()` function shall be equivalent to the `getdelim()` function with the *delimiter* character
 33056 equal to the `<newline>` character.

33057 **RETURN VALUE**

33058 Upon successful completion, the `getdelim()` function shall return the number of characters
 33059 written into the buffer, including the delimiter character if one was encountered before EOF.
 33060 Otherwise, it shall return `-1` and set *errno* to indicate the error.

33061 **ERRORS**

33062 These functions shall fail if:

33063 [EINVAL] When *lineptr* or *n* are a null pointer.

33064 [ENOMEM] Insufficient memory is available.

33065 These functions may fail if:

33066 [EINVAL] *stream* is not a valid file descriptor.33067 [EOVERFLOW] More than {SSIZE_MAX} characters were read without encountering the
33068 *delimiter* character.33069 **EXAMPLES**

```

33070         #include <stdio.h>
33071         #include <stdlib.h>
33072
33073         int main(void)
33074         {
33075             FILE *fp;
33076             char *line = NULL;
33077             size_t len = 0;
33078             ssize_t read;
33079             fp = fopen("/etc/motd", "r");
33079             if (fp == NULL)

```

```

33080         exit(1);
33081     while ((read = getline(&line, &len, fp)) != -1) {
33082         printf("Retrieved line of length %zu :\n", read);
33083         printf("%s", line);
33084     }
33085     free(line);
33086     fclose(fp);
33087     return 0;
33088 }

```

APPLICATION USAGE

Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

RATIONALE

These functions are widely used to solve the problem that the *fgets()* function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

FUTURE DIRECTIONS

None.

SEE ALSO

fgets(), *free()*

XBD <stdio.h>

CHANGE HISTORY

First released in Issue 7.

DRAFT

33103 **NAME**33104 `getegid` — get the effective group ID33105 **SYNOPSIS**33106 `#include <unistd.h>`
33107 `gid_t getegid(void);`33108 **DESCRIPTION**33109 The `getegid()` function shall return the effective group ID of the calling process.33110 **RETURN VALUE**33111 The `getegid()` function shall always be successful and no return value is reserved to indicate an
33112 error.33113 **ERRORS**

33114 No errors are defined.

33115 **EXAMPLES**

33116 None.

33117 **APPLICATION USAGE**

33118 None.

33119 **RATIONALE**

33120 None.

33121 **FUTURE DIRECTIONS**

33122 None.

33123 **SEE ALSO**33124 [*geteuid\(\), getgid\(\), getuid\(\), setegid\(\), seteuid\(\), setgid\(\), setregid\(\), setreuid\(\), setuid\(\)*](#)33125 XBD [*<sys/types.h>*](#), [*<unistd.h>*](#)33126 **CHANGE HISTORY**

33127 First released in Issue 1. Derived from Issue 1 of the SVID.

33128 **Issue 6**33129 In the SYNOPSIS, the optional include of the [*<sys/types.h>*](#) header is removed.33130 The following new requirements on POSIX implementations derive from alignment with the
33131 Single UNIX Specification:

- 33132
- The requirement to include [*<sys/types.h>*](#) has been removed. Although [*<sys/types.h>*](#) was
33133 required for conforming implementations of previous POSIX specifications, it was not
33134 required for UNIX applications.

33135 **NAME**
 33136 `getenv` — get value of an environment variable

33137 **SYNOPSIS**
 33138 `#include <stdlib.h>`

33139 `char *getenv(const char *name);`

33140 DESCRIPTION

33141 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 33142 conflict between the requirements described here and the ISO C standard is unintentional. This
 33143 volume of POSIX.1-200x defers to the ISO C standard.

33144 The `getenv()` function shall search the environment of the calling process (see XBD Chapter 8, on
 33145 page 159) for the environment variable *name* if it exists and return a pointer to the value of the
 33146 environment variable. If the specified environment variable cannot be found, a null pointer shall
 33147 be returned. The application shall ensure that it does not modify the string pointed to by the
 33148 `getenv()` function.

33149 CX The string pointed to may be overwritten by a subsequent call to `getenv()`, `setenv()`, `unsetenv()`,
 33150 XSI or `putenv()` but shall not be overwritten by a call to any other function in this volume of
 33151 POSIX.1-200x.

33152 CX If the application modifies *environ* or the pointers to which it points, the behavior of `getenv()` is
 33153 undefined.

33154 The `getenv()` function need not be thread-safe. A function that is not required to be thread-safe is
 33155 not required to be reentrant.

33156 RETURN VALUE

33157 Upon successful completion, `getenv()` shall return a pointer to a string containing the *value* for
 33158 the specified *name*. If the specified *name* cannot be found in the environment of the calling
 33159 process, a null pointer shall be returned.

33160 ERRORS

33161 No errors are defined.

33162 EXAMPLES

33163 Getting the Value of an Environment Variable

33164 The following example gets the value of the *HOME* environment variable.

```
33165 #include <stdlib.h>
33166 ...
33167 const char *name = "HOME";
33168 char *value;
33169
33170 value = getenv(name);
```

33170 APPLICATION USAGE

33171 None.

33172 RATIONALE

33173 The `clearenv()` function was considered but rejected. The `putenv()` function has now been
 33174 included for alignment with the Single UNIX Specification.

33175 The `getenv()` function is inherently not reentrant because it returns a value pointing to static
 33176 data.

33177 Conforming applications are required not to modify *environ* directly, but to use only the
 33178 functions described here to manipulate the process environment as an abstract object. Thus, the
 33179 implementation of the environment access functions has complete control over the data
 33180 structure used to represent the environment (subject to the requirement that *environ* be
 33181 maintained as a list of strings with embedded equal signs for applications that wish to scan the
 33182 environment). This constraint allows the implementation to properly manage the memory it
 33183 allocates, either by using allocated storage for all variables (copying them on the first invocation
 33184 of *setenv()* or *unsetenv()*), or keeping track of which strings are currently in allocated space and
 33185 which are not, via a separate table or some other means. This enables the implementation to free
 33186 any allocated space used by strings (and perhaps the pointers to them) stored in *environ* when
 33187 *unsetenv()* is called. A C runtime start-up procedure (that which invokes *main()* and perhaps
 33188 initializes *environ*) can also initialize a flag indicating that none of the environment has yet been
 33189 copied to allocated storage, or that the separate table has not yet been initialized.

33190 In fact, for higher performance of *getenv()*, the implementation could also maintain a separate
 33191 copy of the environment in a data structure that could be searched much more quickly (such as
 33192 an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when
 33193 *setenv()* or *unsetenv()* is invoked.

33194 Performance of *getenv()* can be important for applications which have large numbers of
 33195 environment variables. Typically, applications like this use the environment as a resource
 33196 database of user-configurable parameters. The fact that these variables are in the user's shell
 33197 environment usually means that any other program that uses environment variables (such as *ls*,
 33198 which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is
 33199 similarly slowed down by the linear search through the variables.

33200 An implementation that maintains separate data structures, or even one that manages the
 33201 memory it consumes, is not currently required as it was thought it would reduce consensus
 33202 among implementors who do not want to change their historical implementations.

33203 The POSIX Threads Extension states that multi-threaded applications must not modify *environ*
 33204 directly, and that POSIX.1-200x is providing functions which such applications can use in the
 33205 future to manipulate the environment in a thread-safe manner. Thus, moving away from
 33206 application use of *environ* is desirable from that standpoint as well.

33207 FUTURE DIRECTIONS

33208 None.

33209 SEE ALSO

33210 [exec](#), [putenv\(\)](#), [setenv\(\)](#), [unsetenv\(\)](#)

33211 XBD Chapter 8 (on page 159), [<stdlib.h>](#)

33212 CHANGE HISTORY

33213 First released in Issue 1. Derived from Issue 1 of the SVID.

33214 Issue 5

33215 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 33216 VALUE section.

33217 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

33218 Issue 6

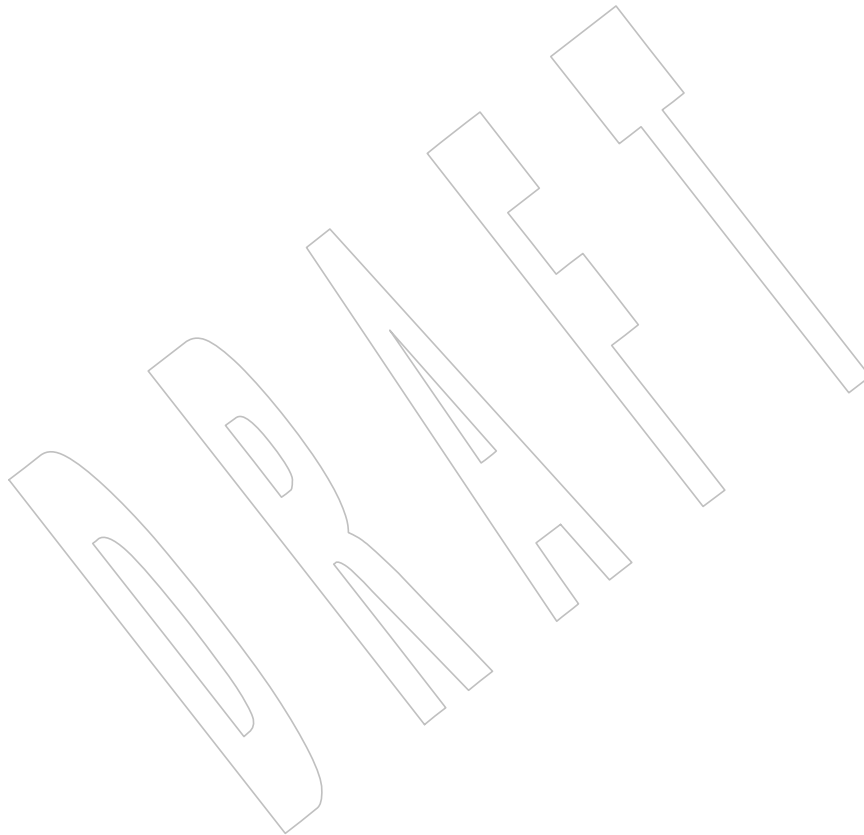
33219 The following changes were made to align with the IEEE P1003.1a draft standard:

- 33220 • References added to the new *setenv()* and *unsetenv()* functions.

33221 The normative text is updated to avoid use of the term “must” for application requirements.

33222
33223
33224**Issue 7**

Austin Group Interpretation 1003.1-2001 #062 is applied, clarifying that a call to *putenv()* may also cause the string to be overwritten.



33225 **NAME**

33226 geteuid — get the effective user ID

33227 **SYNOPSIS**33228 #include <unistd.h>
33229 uid_t geteuid(void);33230 **DESCRIPTION**33231 The *geteuid()* function shall return the effective user ID of the calling process.33232 **RETURN VALUE**33233 The *geteuid()* function shall always be successful and no return value is reserved to indicate an
33234 error.33235 **ERRORS**

33236 No errors are defined.

33237 **EXAMPLES**

33238 None.

33239 **APPLICATION USAGE**

33240 None.

33241 **RATIONALE**

33242 None.

33243 **FUTURE DIRECTIONS**

33244 None.

33245 **SEE ALSO**33246 *getegid(), getgid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*33247 XBD [<sys/types.h>](#), [<unistd.h>](#)33248 **CHANGE HISTORY**

33249 First released in Issue 1. Derived from Issue 1 of the SVID.

33250 **Issue 6**33251 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.33252 The following new requirements on POSIX implementations derive from alignment with the
33253 Single UNIX Specification:

- 33254
- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
33255 required for conforming implementations of previous POSIX specifications, it was not
33256 required for UNIX applications.

33257 **NAME**

33258 getgid — get the real group ID

33259 **SYNOPSIS**

33260 #include <unistd.h>

33261 gid_t getgid(void);

33262 **DESCRIPTION**33263 The *getgid()* function shall return the real group ID of the calling process.33264 **RETURN VALUE**33265 The *getgid()* function shall always be successful and no return value is reserved to indicate an
33266 error.33267 **ERRORS**

33268 No errors are defined.

33269 **EXAMPLES**

33270 None.

33271 **APPLICATION USAGE**

33272 None.

33273 **RATIONALE**

33274 None.

33275 **FUTURE DIRECTIONS**

33276 None.

33277 **SEE ALSO**33278 *getegid(), geteuid(), getuid(), setegid(), seteuid(), setgid(), setregid(), setreuid(), setuid()*

33279 XBD <sys/types.h>, <unistd.h>

33280 **CHANGE HISTORY**

33281 First released in Issue 1. Derived from Issue 1 of the SVID.

33282 **Issue 6**

33283 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

33284 The following new requirements on POSIX implementations derive from alignment with the
33285 Single UNIX Specification:

- 33286 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
33287 required for conforming implementations of previous POSIX specifications, it was not
33288 required for UNIX applications.

getgrent()

33289 **NAME**
33290 getgrent — get the group database entry

33291 **SYNOPSIS**

33292 XSI #include <grp.h>
33293 struct group *getgrent(void);

33294 **DESCRIPTION**

33295 Refer to *endgrent()*.

33296 **NAME**
 33297 `getgrgid, getgrgid_r` — get group database entry for a group ID

33298 **SYNOPSIS**
 33299 `#include <grp.h>`
 33300 `struct group *getgrgid(gid_t gid);`
 33301 `int getgrgid_r(gid_t gid, struct group *grp, char *buffer,`
 33302 `size_t bufsize, struct group **result);`

33303 **DESCRIPTION**
 33304 The `getgrgid()` function shall search the group database for an entry with a matching `gid`.
 33305 The `getgrgid()` function need not be thread-safe. A function that is not required to be thread-safe
 33306 is not required to be reentrant.

33307 The `getgrgid_r()` function shall update the **group** structure pointed to by `grp` and store a pointer
 33308 to that structure at the location pointed to by `result`. The structure shall contain an entry from
 33309 the group database with a matching `gid`. Storage referenced by the group structure is allocated
 33310 from the memory provided with the `buffer` parameter, which is `bufsize` bytes in size. A call to
 33311 `sysconf(_SC_GETGR_R_SIZE_MAX)` returns either `-1` without changing `errno` or an initial value
 33312 suggested for the size of this buffer. A NULL pointer shall be returned at the location pointed to
 33313 by `result` on error or if the requested entry is not found.

33314 **RETURN VALUE**
 33315 Upon successful completion, `getgrgid()` shall return a pointer to a **struct group** with the structure
 33316 defined in `<grp.h>` with a matching entry if one is found. The `getgrgid()` function shall return a
 33317 null pointer if either the requested entry was not found, or an error occurred. On error, `errno`
 33318 shall be set to indicate the error.

33319 The return value may point to a static area which is overwritten by a subsequent call to
 33320 `getgrent()`, `getgrgid()`, or `getgrnam()`.

33321 If successful, the `getgrgid_r()` function shall return zero; otherwise, an error number shall be
 33322 returned to indicate the error.

33323 **ERRORS**
 33324 The `getgrgid()` and `getgrgid_r()` functions may fail if:
 33325 [EIO] An I/O error has occurred.
 33326 [EINTR] A signal was caught during `getgrgid()`.
 33327 [EMFILE] All file descriptors available to the process are currently open.
 33328 [ENFILE] The maximum allowable number of files is currently open in the system.
 33329 The `getgrgid_r()` function may fail if:
 33330 [ERANGE] Insufficient storage was supplied via `buffer` and `bufsize` to contain the data to be
 33331 referenced by the resulting **group** structure.

EXAMPLES

Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with `getgrgid_r()`.

```

33332
33333
33334
33335
33336
33337
33338
33339
33340
33341
33342
33343
33344
33345
33346
33347
33348
33349
33350
33351
33352
33353
33354
33355
33356
33357
33358
33359
33360
33361
33362
33363
33364
33365
33366
33367
33368
33369
33370
33371
33372
33373
33374
33375
33376
33377
33378
33379
33380
33381

```

```

long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
{
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
}
if (e != 0)
    ...handle error...

```

Finding an Entry in the Group Database

The following example uses `getgrgid()` to search the group database for a group ID that was previously stored in a `stat` structure, then prints out the group name if it is found. If the group is not found, the program prints the numeric value of the group for the entry.

```

33366
33367
33368
33369
33370
33371
33372
33373
33374
33375
33376
33377

```

```

#include <sys/types.h>
#include <grp.h>
#include <stdio.h>
...
struct stat statbuf;
struct group *grp;
...
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8d", statbuf.st_gid);
...

```

APPLICATION USAGE

Applications wishing to check for error situations should set `errno` to 0 before calling `getgrgid()`. If `errno` is set on return, an error occurred.

The `getgrgid_r()` function is thread-safe and shall return values in a user-supplied buffer instead

33382 of possibly using a static data area that may be overwritten by each call.

33383 Portable applications should take into account that it is usual for an implementation to return `-1` +
 33384 from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

33385 RATIONALE

33386 None.

33387 FUTURE DIRECTIONS

33388 None.

33389 SEE ALSO

33390 `endgrent()`, `getgrnam()`, `sysconf()`

33391 XBD `<grp.h>`, `<limits.h>`, `<sys/types.h>` |

33392 CHANGE HISTORY

33393 First released in Issue 1. Derived from System V Release 2.0.

33394 Issue 5

33395 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 33396 VALUE section.

33397 The `getgrgid_r()` function is included for alignment with the POSIX Threads Extension.

33398 A note indicating that the `getgrgid()` function need not be reentrant is added to the
 33399 DESCRIPTION.

33400 Issue 6

33401 The `getgrgid_r()` function is marked as part of the Thread-Safe Functions option.

33402 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 33403 describing matching the `gid`.

33404 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

33405 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

33406 The following new requirements on POSIX implementations derive from alignment with the
 33407 Single UNIX Specification:

- 33408 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
- 33409 required for conforming implementations of previous POSIX specifications, it was not
- 33410 required for UNIX applications.
- 33411 • In the RETURN VALUE section, the requirement to set `errno` on error is added.
- 33412 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

33413 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 33414 its avoidance of possibly using a static data area.

33415 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
 33416 buffer from `bufsize` characters to bytes.

33417 Issue 7

33418 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

33419 SD5-XSH-ERN-166 is applied. +

33420 The `getgrgid_r()` function is moved from the Thread-Safe Functions option to the Base.

33421 **NAME**

33422 getgrnam, getgrnam_r — search group database for a name

33423 **SYNOPSIS**

33424 #include <grp.h>

33425 struct group *getgrnam(const char *name);

33426 int getgrnam_r(const char *name, struct group *grp, char *buffer,
33427 size_t bufsize, struct group **result);33428 **DESCRIPTION**33429 The *getgrnam()* function shall search the group database for an entry with a matching *name*.33430 The *getgrnam()* function need not be thread-safe. A function that is not required to be thread-
33431 safe is not required to be reentrant.33432 The *getgrnam_r()* function shall update the **group** structure pointed to by *grp* and store a pointer
33433 to that structure at the location pointed to by *result*. The structure shall contain an entry from
33434 the group database with a matching *gid* or *name*. Storage referenced by the **group** structure is
33435 allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A
33436 call to *sysconf(_SC_GETGR_R_SIZE_MAX)* returns either -1 without changing *errno* or an initial
33437 value suggested for the size of this buffer. A NULL pointer is returned at the location pointed to
33438 by *result* on error or if the requested entry is not found.33439 **RETURN VALUE**33440 The *getgrnam()* function shall return a pointer to a **struct group** with the structure defined in
33441 <grp.h> with a matching entry if one is found. The *getgrnam()* function shall return a null
33442 pointer if either the requested entry was not found, or an error occurred. On error, *errno* shall be
33443 set to indicate the error.33444 The return value may point to a static area which is overwritten by a subsequent call to
33445 *getgrent()*, *getgrgid()*, or *getgrnam()*.33446 The *getgrnam_r()* function shall return zero on success or if the requested entry was not found
33447 and no error has occurred. If any error has occurred, an error number shall be returned to
33448 indicate the error.33449 **ERRORS**33450 The *getgrnam()* and *getgrnam_r()* functions may fail if:

33451 [EIO] An I/O error has occurred.

33452 [EINTR] A signal was caught during *getgrnam()*.

33453 [EMFILE] All file descriptors available to the process are currently open.

33454 [ENFILE] The maximum allowable number of files is currently open in the system.

33455 The *getgrnam_r()* function may fail if:33456 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
33457 referenced by the resulting **group** structure.

EXAMPLES

Note that `sysconf(_SC_GETGR_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with `getgrnam_r()`.

```

33458
33459     long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
33460     size_t len;
33461     if (initlen == -1)
33462         /* Default initial length. */
33463         len = 1024;
33464     else
33465         len = (size_t) initlen;
33466     struct group result;
33467     struct group *resultp;
33468     char *buffer = malloc(len);
33469     if (buffer == NULL)
33470         ...handle error...
33471     int e;
33472     while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
33473           == ERANGE)
33474     {
33475         size_t newlen = 2 * len;
33476         if (newlen < len)
33477             ...handle error...
33478         len = newlen;
33479         char *newbuffer = realloc(buffer, len);
33480         if (newbuffer == NULL)
33481             ...handle error...
33482         buffer = newbuffer;
33483     }
33484     if (e != 0)
33485         ...handle error...

```

APPLICATION USAGE

Applications wishing to check for error situations should set `errno` to 0 before calling `getgrnam()`. If `errno` is set on return, an error occurred.

The `getgrnam_r()` function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return `-1` from `sysconf()` indicating that there is no maximum for `_SC_GETGR_R_SIZE_MAX`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*endgrent\(\)*](#), [*getgrgid\(\)*](#), [*sysconf\(\)*](#)

XBD [*<grp.h>*](#), [*<limits.h>*](#), [*<sys/types.h>*](#)

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

33505 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
33506 VALUE section.
33507

33508 The *getgrnam_r()* function is included for alignment with the POSIX Threads Extension.

33509 A note indicating that the *getgrnam()* function need not be reentrant is added to the
33510 DESCRIPTION.

Issue 6

33511 The *getgrnam_r()* function is marked as part of the Thread-Safe Functions option.

33512 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

33513 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

33514 The following new requirements on POSIX implementations derive from alignment with the
33515 Single UNIX Specification:
33516

- 33517 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
33518 required for conforming implementations of previous POSIX specifications, it was not
33519 required for UNIX applications.
- 33520 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 33521 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

33522 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
33523 its avoidance of possibly using a static data area.

33524 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
33525 buffer from *bufsize* characters to bytes.

Issue 7

33526 Austin Group Interpretation 1003.1-2001 #081 is applied, clarifying the RETURN VALUE section. +

33527 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

33528 SD5-XSH-ERN-166 is applied. |

33530 The *getgrnam_r()* function is moved from the Thread-Safe Functions option to the Base.

33531 **NAME**33532 `getgroups` — get supplementary group IDs33533 **SYNOPSIS**33534 `#include <unistd.h>`33535 `int getgroups(int gidsetsize, gid_t grouplist[]);`33536 **DESCRIPTION**33537 The `getgroups()` function shall fill in the array `grouplist` with the current supplementary group
33538 IDs of the calling process. It is implementation-defined whether `getgroups()` also returns the
33539 effective group ID in the `grouplist` array.33540 The `gidsetsize` argument specifies the number of elements in the array `grouplist`. The actual
33541 number of group IDs stored in the array shall be returned. The values of array entries with
33542 indices greater than or equal to the value returned are undefined.33543 If `gidsetsize` is 0, `getgroups()` shall return the number of group IDs that it would otherwise return
33544 without modifying the array pointed to by `grouplist`.33545 If the effective group ID of the process is returned with the supplementary group IDs, the value
33546 returned shall always be greater than or equal to one and less than or equal to the value of
33547 `{NGROUPS_MAX}+1`.33548 **RETURN VALUE**33549 Upon successful completion, the number of supplementary group IDs shall be returned. A
33550 return value of `-1` indicates failure and `errno` shall be set to indicate the error.33551 **ERRORS**33552 The `getgroups()` function shall fail if:33553 `[EINVAL]` The `gidsetsize` argument is non-zero and less than the number of group IDs
33554 that would have been returned.33555 **EXAMPLES**33556 **Getting the Supplementary Group IDs of the Calling Process**33557 The following example places the current supplementary group IDs of the calling process into
33558 the `group` array.33559 `#include <sys/types.h>`
33560 `#include <unistd.h>`
33561 `...`
33562 `gid_t *group;`
33563 `int nogroups;`
33564 `long ngroups_max;`

33565 `ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;`
33566 `group = (gid_t *)malloc(ngroups_max * sizeof(gid_t));`

33567 `ngroups = getgroups(ngroups_max, group);`33568 **APPLICATION USAGE**

33569 None.

33570
33571
33572

33573
33574
33575
33576
33577
33578
33579

33580
33581

33582
33583

33584
33585
33586

33587
33588

33589
33590
33591

33592
33593
33594
33595

33596
33597
33598

33599
33600

33601

33602
33603

RATIONALE

The related function *setgroups()* is a privileged operation and therefore is not covered by this volume of POSIX.1-200x.

As implied by the definition of supplementary groups, the effective group ID may appear in the array returned by *getgroups()* or it may be returned only by *getegid()*. Duplication may exist, but the application needs to call *getegid()* to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups()* to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of “set”). (The history of a process and its parents could affect the details of the result.)

Application developers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid(), *setgid()*

XBD [<sys/types.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the DESCRIPTION.

Issue 6

In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- A return value of 0 is not permitted, because {NGROUPS_MAX} cannot be 0. This is a FIPS requirement.

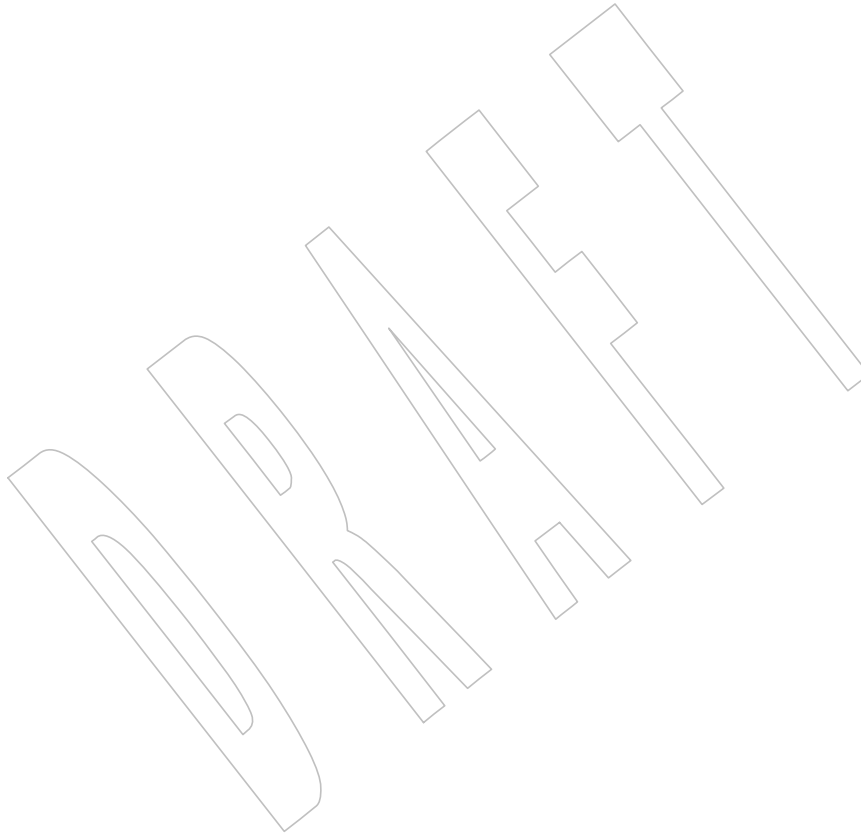
The following changes were made to align with the IEEE P1003.1a draft standard:

- An explanation is added that the effective group ID may be included in the supplementary group list.

33604 **NAME**
33605 gethostent — network host database functions

33606 **SYNOPSIS**
33607 #include <netdb.h>
33608 struct hostent *gethostent(void);

33609 **DESCRIPTION**
33610 Refer to *endhostent()*.



33611 **NAME**
 33612 gethostid — get an identifier for the current host

33613 **SYNOPSIS**

33614 XSI #include <unistd.h>
 33615 long gethostid(void);

33616 **DESCRIPTION**

33617 The *gethostid()* function shall retrieve a 32-bit identifier for the current host.

33618 **RETURN VALUE**

33619 Upon successful completion, *gethostid()* shall return an identifier for the current host.

33620 **ERRORS**

33621 No errors are defined.

33622 **EXAMPLES**

33623 None.

33624 **APPLICATION USAGE**

33625 This volume of POSIX.1-200x does not define the domain in which the return value is unique.

33626 **RATIONALE**

33627 None.

33628 **FUTURE DIRECTIONS**

33629 None.

33630 **SEE ALSO**

33631 *initstate()*
 33632 XBD <unistd.h>

33633 **CHANGE HISTORY**

33634 First released in Issue 4, Version 2.

33635 **Issue 5**

33636 Moved from X/OPEN UNIX extension to BASE.

33637 **NAME**

33638 gethostname — get name of current host

33639 **SYNOPSIS**

33640 #include <unistd.h>

33641 int gethostname(char *name, size_t namelen);

33642 **DESCRIPTION**

33643 The *gethostname()* function shall return the standard host name for the current machine. The
 33644 *namelen* argument shall specify the size of the array pointed to by the *name* argument. The
 33645 returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold
 33646 the host name, then the returned name shall be truncated and it is unspecified whether the
 33647 returned name is null-terminated.

33648 Host names are limited to {HOST_NAME_MAX} bytes.

33649 **RETURN VALUE**

33650 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned.

33651 **ERRORS**

33652 No errors are defined.

33653 **EXAMPLES**

33654 None.

33655 **APPLICATION USAGE**

33656 None.

33657 **RATIONALE**

33658 None.

33659 **FUTURE DIRECTIONS**

33660 None.

33661 **SEE ALSO**33662 *gethostid()*, *uname*

33663 XBD <unistd.h>

33664 **CHANGE HISTORY**

33665 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

33666 The Open Group Base Resolution bwg2001-008 is applied, changing the *namelen* parameter from
 33667 *socklen_t* to *size_t*.

33668 **NAME**33669 `getitimer, setitimer` — get and set value of interval timer33670 **SYNOPSIS**

```

33671 OB XSI #include <sys/time.h>
33672
33672 int getitimer(int which, struct itimerval *value);
33673 int setitimer(int which, const struct itimerval *restrict value,
33674              struct itimerval *restrict ovalue);

```

33675 **DESCRIPTION**

33676 The `getitimer()` function shall store the current value of the timer specified by *which* into the
 33677 structure pointed to by *value*. The `setitimer()` function shall set the timer specified by *which* to the
 33678 value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store the
 33679 previous value of the timer in the structure pointed to by *ovalue*.

33680 A timer value is defined by the **itimerval** structure, specified in **<sys/time.h>**. If *it_value* is non-
 33681 zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall
 33682 specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall
 33683 disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer
 33684 after its next expiration (assuming *it_value* is non-zero).

33685 Implementations may place limitations on the granularity of timer values. For each interval
 33686 timer, if the requested timer value requires a finer granularity than the implementation supports,
 33687 the actual timer value shall be rounded up to the next supported value.

33688 An XSI-conforming implementation provides each process with at least three interval timers,
 33689 which are indicated by the *which* argument:

33690 **ITIMER_PROF** Decrements both in process virtual time and when the system is running
 33691 on behalf of the process. It is designed to be used by interpreters in
 33692 statistically profiling the execution of interpreted programs. Each time the
 33693 **ITIMER_PROF** timer expires, the **SIGPROF** signal is delivered.

33694 **ITIMER_REAL** Decrements in real time. A **SIGALRM** signal is delivered when this timer
 33695 expires.

33696 **ITIMER_VIRTUAL** Decrements in process virtual time. It runs only when the process is
 33697 executing. A **SIGVTALRM** signal is delivered when it expires.

33698 The interaction between `setitimer()` and `alarm()` or `sleep()` is unspecified.

33699 **RETURN VALUE**

33700 Upon successful completion, `getitimer()` or `setitimer()` shall return 0; otherwise, `-1` shall be
 33701 returned and *errno* set to indicate the error.

33702 **ERRORS**

33703 The `setitimer()` function shall fail if:

33704 **[EINVAL]** The *value* argument is not in canonical form. (In canonical form, the number of
 33705 microseconds is a non-negative integer less than 1 000 000 and the number of
 33706 seconds is a non-negative integer.)

33707 The `getitimer()` and `setitimer()` functions may fail if:

33708 **[EINVAL]** The *which* argument is not recognized.

33709
33710
33711
33712
33713
33714
33715
33716
33717
33718
33719
33720
33721
33722
33723
33724
33725
33726
33727
33728
33729

EXAMPLES

None.

APPLICATION USAGE

Applications should use the *timer_gettime()* and *timer_settime()* functions instead of the obsolescent *getitimer()* and *setitimer()* functions, respectively.

RATIONALE

None.

FUTURE DIRECTIONS

The *getitimer()* and *setitimer()* functions may be removed in a future version.

SEE ALSO

alarm(), *exec*, *sleep*, *timer_getoverrun()*

XBD [<signal.h>](#), [<sys/time.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

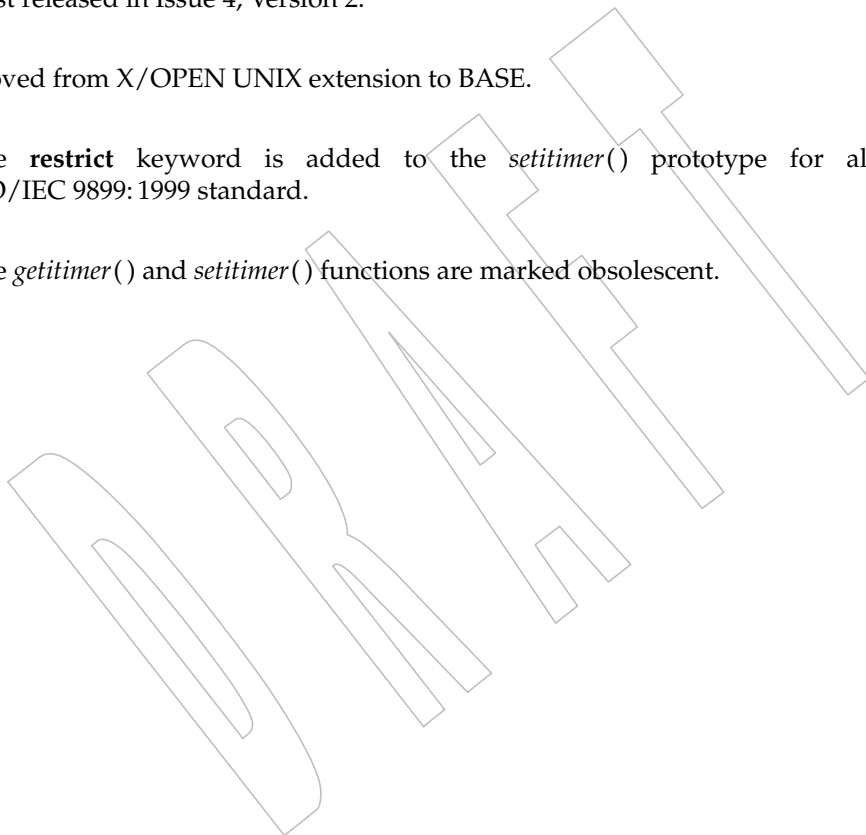
Moved from X/OPEN UNIX extension to BASE.

Issue 6

The **restrict** keyword is added to the *setitimer()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

The *getitimer()* and *setitimer()* functions are marked obsolescent.



getline()

33730

NAME

33731

getline — read a delimited record from *stream*

33732

SYNOPSIS

33733

CX `#include <stdio.h>`

33734

`ssize_t getline(char **lineptr, size_t *n, FILE *stream);`

33735

DESCRIPTION

33736

Refer to [*getdelim\(\)*](#).

33737 **NAME**
 33738 getlogin, getlogin_r — get login name

33739 **SYNOPSIS**
 33740 #include <unistd.h>
 33741 char *getlogin(void);
 33742 int getlogin_r(char *name, size_t namesize);

33743 **DESCRIPTION**
 33744 The *getlogin()* function shall return a pointer to a string containing the user name associated by
 33745 the login activity with the controlling terminal of the current process. If *getlogin()* returns a non-
 33746 null pointer, then that pointer points to the name that the user logged in under, even if there are
 33747 several login names with the same user ID.

33748 The *getlogin()* function need not be thread-safe. A function that is not required to be thread-safe
 33749 is not required to be reentrant.

33750 The *getlogin_r()* function shall put the name associated by the login activity with the controlling
 33751 terminal of the current process in the character array pointed to by *name*. The array is *namesize*
 33752 characters long and should have space for the name and the terminating null character. The
 33753 maximum size of the login name is {LOGIN_NAME_MAX}.

33754 If *getlogin_r()* is successful, *name* points to the name the user used at login, even if there are
 33755 several login names with the same user ID.

33756 **RETURN VALUE**
 33757 Upon successful completion, *getlogin()* shall return a pointer to the login name or a null pointer
 33758 if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to
 33759 indicate the error.

33760 The return value from *getlogin()* may point to static data whose content is overwritten by each
 33761 call.

33762 If successful, the *getlogin_r()* function shall return zero; otherwise, an error number shall be
 33763 returned to indicate the error.

33764 **ERRORS**
 33765 The *getlogin()* and *getlogin_r()* functions may fail if:

- 33766 [EMFILE] All file descriptors available to the process are currently open.
- 33767 [ENFILE] The maximum allowable number of files is currently open in the system.
- 33768 [ENXIO] The calling process has no controlling terminal.

33769 The *getlogin_r()* function may fail if:

- 33770 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 33771 including the terminating null character.

EXAMPLES

Getting the User Login Name

The following example calls the `getlogin()` function to obtain the name of the user associated with the calling process, and passes this information to the `getpwnam()` function to get the associated user database information.

```

33772
33773
33774 #include <unistd.h>
33775 #include <sys/types.h>
33776 #include <pwd.h>
33777 #include <stdio.h>
33778 ...
33779 char *lgn;
33780 struct passwd *pw;
33781 ...
33782 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
33783     fprintf(stderr, "Get of user information failed.\n"); exit(1);
33784 }
33785
33786
33787

```

APPLICATION USAGE

Three names associated with the current process can be determined: `getpwuid(geteuid())` shall return the name associated with the effective user ID of the process; `getlogin()` shall return the name associated with the current login activity; and `getpwuid(getuid())` shall return the name associated with the real user ID of the process.

The `getlogin_r()` function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The `getlogin()` function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of `getlogin()` should be used to provide the argument to the `getpwnam()` function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the `cuserid()` function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(geteuid())
```

while the information provided by historical implementations of `cuserid()` can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

[`getpwnam\(\)`](#), [`getpwuid\(\)`](#), [`geteuid\(\)`](#), [`getuid\(\)`](#)

XBD [`<limits.h>`](#), [`<unistd.h>`](#)

33814
33815

33816
33817
33818

33819

33820
33821

33822
33823
33824

33825
33826

33827
33828

33829
33830

33831
33832

33833

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getlogin_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getlogin()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getlogin_r()* function is marked as part of the Thread-Safe Functions option.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

Issue 7

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

The *getlogin_r()* function is moved from the Thread-Safe Functions option to the Base.

33834 NAME

33835 getmsg, getpmsg — receive next message from a STREAMS file (**STREAMS**)

33836 SYNOPSIS

```

33837 OB XSR #include <stropts.h>
33838
33838 int getmsg(int fildev, struct strbuf *restrict ctlptr,
33839           struct strbuf *restrict dataptr, int *restrict flagsp);
33840 int getpmsg(int fildev, struct strbuf *restrict ctlptr,
33841            struct strbuf *restrict dataptr, int *restrict bandp,
33842            int *restrict flagsp);

```

33843 DESCRIPTION

33844 The *getmsg()* function shall retrieve the contents of a message located at the head of the
 33845 STREAM head read queue associated with a STREAMS file and place the contents into one or
 33846 more buffers. The message contains either a data part, a control part, or both. The data and
 33847 control parts of the message shall be placed into separate buffers, as described below. The
 33848 semantics of each part are defined by the originator of the message.

33849 The *getpmsg()* function shall be equivalent to *getmsg()*, except that it provides finer control over
 33850 the priority of the messages received. Except where noted, all requirements on *getmsg()* also
 33851 pertain to *getpmsg()*.

33852 The *fildev* argument specifies a file descriptor referencing a STREAMS-based file.

33853 The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points
 33854 to a buffer in which the data or control information is to be placed, and the *maxlen* member
 33855 indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall
 33856 contain the number of bytes of data or control information actually received. The *len* member
 33857 shall be set to 0 if there is a zero-length control or data part and *len* shall be set to -1 if no data or
 33858 control information is present in the message.

33859 When *getmsg()* is called, *flagsp* should point to an integer that indicates the type of message the
 33860 process is able to receive. This is described further below.

33861 The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold
 33862 the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is -1, the
 33863 control (or data) part of the message shall not be processed and shall be left on the STREAM
 33864 head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to -1. If the
 33865 *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part
 33866 shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0
 33867 and there are more than 0 bytes of control (or data) information, that information shall be left on
 33868 the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the
 33869 control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder
 33870 of the message shall be left on the STREAM head read queue and a non-zero return value shall
 33871 be provided.

33872 By default, *getmsg()* shall process the first available message on the STREAM head read queue.
 33873 However, a process may choose to retrieve only high-priority messages by setting the integer
 33874 pointed to by *flagsp* to RS_HIPRI. In this case, *getmsg()* shall only process the next message if it is
 33875 a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall
 33876 be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a
 33877 high-priority message was retrieved, or 0 otherwise.

33878 For *getpmsg()*, the flags are different. The *flagsp* argument points to a bitmask with the following
 33879 mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like *getmsg()*,

33880 *getpmsg()* shall process the first available message on the STREAM head read queue. A process
 33881 may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to
 33882 MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg()* shall only process
 33883 the next message if it is a high-priority message. In a similar manner, a process may choose to
 33884 retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to
 33885 MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case,
 33886 *getpmsg()* shall only process the next message if it is in a priority band equal to, or greater than,
 33887 the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the
 33888 first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the
 33889 integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-
 33890 priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer
 33891 pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to
 33892 MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

33893 If O_NONBLOCK is not set, *getmsg()* and *getpmsg()* shall block until a message of the type
 33894 specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is
 33895 set and a message of the specified type is not present at the front of the read queue, *getmsg()* and
 33896 *getpmsg()* shall fail and set *errno* to [EAGAIN].

33897 If a hangup occurs on the STREAM from which messages are retrieved, *getmsg()* and *getpmsg()*
 33898 shall continue to operate normally, as described above, until the STREAM head read queue is
 33899 empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

33900 RETURN VALUE

33901 Upon successful completion, *getmsg()* and *getpmsg()* shall return a non-negative value. A value
 33902 of 0 indicates that a full message was read successfully. A return value of MORECTL indicates
 33903 that more control information is waiting for retrieval. A return value of MOREDATA indicates
 33904 that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL
 33905 and MOREDATA indicates that both types of information remain. Subsequent *getmsg()* and
 33906 *getpmsg()* calls shall retrieve the remainder of the message. However, if a message of higher
 33907 priority has come in on the STREAM head read queue, the next call to *getmsg()* or *getpmsg()*
 33908 shall retrieve that higher-priority message before retrieving the remainder of the previous
 33909 message.

33910 If the high priority control part of the message is consumed, the message shall be placed back on
 33911 the queue as a normal message of band 0. Subsequent *getmsg()* and *getpmsg()* calls shall retrieve
 33912 the remainder of the message. If, however, a priority message arrives or already exists on the
 33913 STREAM head, the subsequent call to *getmsg()* or *getpmsg()* shall retrieve the higher-priority
 33914 message before retrieving the remainder of the message that was put back.

33915 Upon failure, *getmsg()* and *getpmsg()* shall return -1 and set *errno* to indicate the error.

33916 ERRORS

33917 The *getmsg()* and *getpmsg()* functions shall fail if:

- | | | |
|-------|-----------|--|
| 33918 | [EAGAIN] | The O_NONBLOCK flag is set and no messages are available. |
| 33919 | [EBADF] | The <i>fildev</i> argument is not a valid file descriptor open for reading. |
| 33920 | [EBADMSG] | The queued message to be read is not valid for <i>getmsg()</i> or <i>getpmsg()</i> or a pending file descriptor is at the STREAM head. |
| 33922 | [EINTR] | A signal was caught during <i>getmsg()</i> or <i>getpmsg()</i> . |
| 33923 | [EINVAL] | An illegal value was specified by <i>flagsp</i> , or the STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer. |
| 33924 | | |
| 33925 | | |

33926 [ENOSTR] A STREAM is not associated with *files*.

33927 In addition, *getmsg()* and *getpmsg()* shall fail if the STREAM head had processed an
 33928 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
 33929 *getmsg()* or *getpmsg()* but reflects the prior error.

33930 EXAMPLES

33931 Getting Any Message

33932 In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call
 33933 to *getmsg()* retrieves any available message on the associated STREAM-head read queue,
 33934 returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*,
 33935 respectively.

```
33936 #include <stropts.h>
33937 ...
33938 int fd;
33939 char ctrlbuf[128];
33940 char databuf[512];
33941 struct strbuf ctrl;
33942 struct strbuf data;
33943 int flags = 0;
33944 int ret;

33945 ctrl.buf = ctrlbuf;
33946 ctrl.maxlen = sizeof(ctrlbuf);

33947 data.buf = databuf;
33948 data.maxlen = sizeof(databuf);
33949 ret = getmsg (fd, &ctrl, &data, &flags);
```

33950 Getting the First Message off the Queue

33951 In the following example, the call to *getpmsg()* retrieves the first available message on the
 33952 associated STREAM-head read queue.

```
33953 #include <stropts.h>
33954 ...
33955 int fd;
33956 char ctrlbuf[128];
33957 char databuf[512];
33958 struct strbuf ctrl;
33959 struct strbuf data;
33960 int band = 0;
33961 int flags = MSG_ANY;
33962 int ret;

33963 ctrl.buf = ctrlbuf;
33964 ctrl.maxlen = sizeof(ctrlbuf);

33965 data.buf = databuf;
33966 data.maxlen = sizeof(databuf);
33967 ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

33968 **APPLICATION USAGE**

33969 None.

33970 **RATIONALE**

33971 None.

33972 **FUTURE DIRECTIONS**33973 The *getmsg()* and *getpmsg()* functions may be removed in a future version.33974 **SEE ALSO**33975 [Section 2.6](#) (on page 473), *poll()*, *putmsg()*, *read*, *write*33976 XBD [<stropts.h>](#)33977 **CHANGE HISTORY**

33978 First released in Issue 4, Version 2.

33979 **Issue 5**

33980 Moved from X/OPEN UNIX extension to BASE.

33981 A paragraph regarding “high-priority control parts of messages” is added to the RETURN
33982 VALUE section.33983 **Issue 6**

33984 This function is marked as part of the XSI STREAMS Option Group.

33985 The **restrict** keyword is added to the *getmsg()* and *getpmsg()* prototypes for alignment with the
33986 ISO/IEC 9899:1999 standard.33987 **Issue 7**33988 The *getmsg()* and *getpmsg()* functions are marked obsolescent.

33989 **NAME**33990 `getnameinfo` — get name information33991 **SYNOPSIS**33992 `#include <sys/socket.h>`33993 `#include <netdb.h>`

```
33994 int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
33995 char *restrict node, socklen_t nodelen, char *restrict service,
33996 socklen_t servicelen, int flags);
```

33997 **DESCRIPTION**

33998 The `getnameinfo()` function shall translate a socket address to a node name and service location, all of which are defined as in `freaddrinfo()`.

34000 The `sa` argument points to a socket address structure to be translated.

34001 IP6 If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node name for that IPv4 address.

34004 If the address is the IPv6 unspecified address ("::"), a lookup shall not be performed and the behavior shall be the same as when the node's name cannot be located.

34006 If the `node` argument is non-NULL and the `nodelen` argument is non-zero, then the `node` argument points to a buffer able to contain up to `nodelen` characters that receives the node name as a null-terminated string. If the `node` argument is NULL or the `nodelen` argument is zero, the node name shall not be returned. If the node's name cannot be located, the numeric form of the address contained in the socket address structure pointed to by the `sa` argument is returned instead of its name.

34012 If the `service` argument is non-NULL and the `servicelen` argument is non-zero, then the `service` argument points to a buffer able to contain up to `servicelen` bytes that receives the service name as a null-terminated string. If the `service` argument is NULL or the `servicelen` argument is zero, the service name shall not be returned. If the service's name cannot be located, the numeric form of the service address (for example, its port number) shall be returned instead of its name.

34017 The `flags` argument is a flag that changes the default actions of the function. By default the fully-qualified domain name (FQDN) for the host shall be returned, but:

- 34019 • If the flag bit `NI_NOFQDN` is set, only the node name portion of the FQDN shall be returned for local hosts.
- 34020
- 34021 • If the flag bit `NI_NUMERICHOST` is set, the numeric form of the address contained in the socket address structure pointed to by the `sa` argument shall be returned instead of its name.
- 34022
- 34023
- 34024 • If the flag bit `NI_NAMEREQD` is set, an error shall be returned if the host's name cannot be located.
- 34025
- 34026 • If the flag bit `NI_NUMERICSERV` is set, the numeric form of the service address shall be returned (for example, its port number) instead of its name.
- 34027
- 34028 • If the flag bit `NI_NUMERICSERVICE` is set, the numeric form of the service address shall be returned (for example, its port number) instead of its name.
- 34029
- 34030 • If the flag bit `NI_NUMERICSERVICE` is set, the numeric form of the scope identifier shall be returned (for example, interface index) instead of its name. This flag shall be ignored if the `sa` argument is not an IPv6 address.

- 34031 • If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service
 34032 (SOCK_DGRAM). The default behavior shall assume that the service is a stream service
 34033 (SOCK_STREAM).

34034 **Notes:**

- 34035 1. The two NI_NUMERICxxx flags are required to support the `-n` flag that many
 34036 commands provide.
- 34037 2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for
 34038 example, [512,514]) that represent different services for UDP and TCP.

34039 The `getnameinfo()` function shall be thread-safe.

34040 RETURN VALUE

34041 A zero return value for `getnameinfo()` indicates successful completion; a non-zero return value
 34042 indicates failure. The possible values for the failures are listed in the ERRORS section.

34043 Upon successful completion, `getnameinfo()` shall return the *node* and *service* names, if requested,
 34044 in the buffers provided. The returned names are always null-terminated strings.

34045 ERRORS

34046 The `getnameinfo()` function shall fail and return the corresponding value if:

- 34047 [EAI_AGAIN] The name could not be resolved at this time. Future attempts may succeed.
- 34048 [EAI_BADFLAGS]
 34049 The *flags* had an invalid value.
- 34050 [EAI_FAIL] A non-recoverable error occurred.
- 34051 [EAI_FAMILY] The address family was not recognized or the address length was invalid for
 34052 the specified family.
- 34053 [EAI_MEMORY] There was a memory allocation failure.
- 34054 [EAI_NONAME] The name does not resolve for the supplied parameters.
 34055 NI_NAMEREQD is set and the host's name cannot be located, or both
 34056 *nodename* and *servname* were null.
- 34057 [EAI_OVERFLOW]
 34058 An argument buffer overflowed. The buffer pointed to by the *node* argument
 34059 or the *service* argument was too small.
- 34060 [EAI_SYSTEM] A system error occurred. The error code can be found in *errno*.

34061 EXAMPLES

34062 None.

34063 APPLICATION USAGE

34064 If the returned values are to be used as part of any further name resolution (for example, passed
 34065 to `getaddrinfo()`), applications should provide buffers large enough to store any result possible on
 34066 the system.

34067 Given the IPv4-mapped IPv6 address "`::ffff:1.2.3.4`", the implementation performs a
 34068 lookup as if the socket address structure contains the IPv4 address "`1.2.3.4`".

34069 The IPv6 unspecified address ("`:::`") and the IPv6 loopback address ("`:::1`") are not
 34070 IPv4-compatible addresses.

getnameinfo()*System Interfaces*34071
34072
34073
34074
34075
34076
34077
34078
34079
34080
34081
34082
34083
34084
34085
34086
34087
34088**RATIONALE**

None.

FUTURE DIRECTIONS

None.

SEE ALSO*endservent(), freeaddrinfo(), gai_strerror(), inet_ntop(), socket()*XBD [<netdb.h>](#), [<sys/socket.h>](#)**CHANGE HISTORY**

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The **restrict** keyword is added to the *getnameinfo()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/23 is applied, making various changes in the SYNOPSIS and DESCRIPTION for alignment with IPv6.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/24 is applied, adding the [EAI_OVERFLOW] error to the ERRORS section.

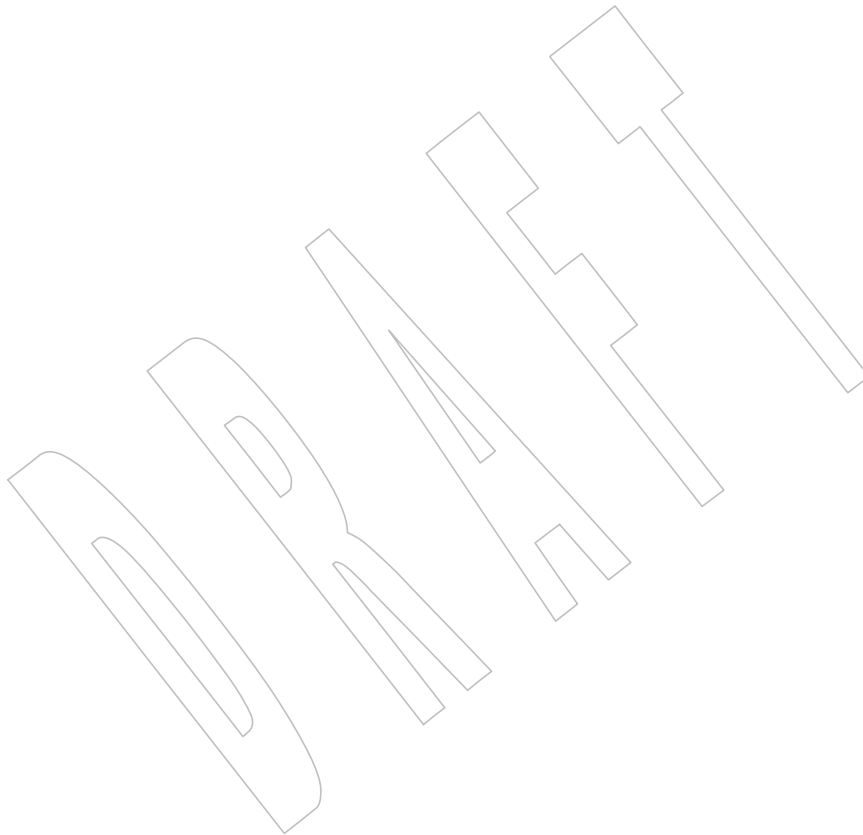
Issue 7

SD5-XSH-ERN-127 is applied, clarifying the behavior if the address is the IPv6 unspecified address.

34089 **NAME**
34090 getnetbyaddr, getnetbyname, getnetent — network database functions

34091 **SYNOPSIS**
34092 #include <netdb.h>
34093 struct netent *getnetbyaddr(uint32_t net, int type);
34094 struct netent *getnetbyname(const char *name);
34095 struct netent *getnetent(void);

34096 **DESCRIPTION**
34097 Refer to *endnetent()*.



34098 **NAME**
 34099 `getopt, optarg, opterr, optind, optopt` — command option parsing

34100 **SYNOPSIS**
 34101 `#include <unistd.h>`
 34102 `int getopt(int argc, char * const argv[], const char *optstring);`
 34103 `extern char *optarg;`
 34104 `extern int opterr, optind, optopt;`

34105 **DESCRIPTION**
 34106 The `getopt()` function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5,
 34107 6, 7, 9, and 10 in XBD [Section 12.2](#) (on page 201).

34108 The parameters `argc` and `argv` are the argument count and argument array as passed to `main()`
 34109 (see [exec](#)). The argument `optstring` is a string of recognized option characters; if a character is
 34110 followed by a colon, the option takes an argument. All option characters allowed by Utility
 34111 Syntax Guideline 3 are allowed in `optstring`. The implementation may accept other characters as
 34112 an extension.

34113 The variable `optind` is the index of the next element of the `argv[]` vector to be processed. It shall
 34114 be initialized to 1 by the system, and `getopt()` shall update it when it finishes with each element
 34115 of `argv[]`. When an element of `argv[]` contains multiple option characters, it is unspecified how
 34116 `getopt()` determines which options have already been processed.

34117 The `getopt()` function shall return the next option character (if one is found) from `argv` that
 34118 matches a character in `optstring`, if there is one that matches. If the option takes an argument,
 34119 `getopt()` shall set the variable `optarg` to point to the option-argument as follows:

- 34120 1. If the option was the last character in the string pointed to by an element of `argv`, then
 34121 `optarg` shall contain the next element of `argv`, and `optind` shall be incremented by 2. If the
 34122 resulting value of `optind` is greater than `argc`, this indicates a missing option-argument,
 34123 and `getopt()` shall return an error indication.
- 34124 2. Otherwise, `optarg` shall point to the string following the option character in that element
 34125 of `argv`, and `optind` shall be incremented by 1.

34126 If, when `getopt()` is called:

34127 `argv[optind]` is a null pointer
 34128 `*argv[optind]` is not the character `-`
 34129 `argv[optind]` points to the string `"-"`

34130 `getopt()` shall return `-1` without changing `optind`. If:

34131 `argv[optind]` points to the string `"--"`

34132 `getopt()` shall return `-1` after incrementing `optind`.

34133 If `getopt()` encounters an option character that is not contained in `optstring`, it shall return the
 34134 question-mark (`' ? '`) character. If it detects a missing option-argument, it shall return the colon
 34135 character (`' : '`) if the first character of `optstring` was a colon, or a question-mark character (`' ? '`)
 34136 otherwise. In either case, `getopt()` shall set the variable `optopt` to the option character that caused
 34137 the error. If the application has not set the variable `opterr` to 0 and the first character of `optstring` is
 34138 not a colon, `getopt()` shall also print a diagnostic message to `stderr` in the format specified for the
 34139 `getopts` utility.

34140 The `getopt()` function need not be thread-safe. A function that is not required to be thread-safe is
 34141 not required to be reentrant.

34142 **RETURN VALUE**34143 The *getopt()* function shall return the next option character specified on the command line.34144 A colon (':') shall be returned if *getopt()* detects a missing argument and the first character of
34145 *optstring* was a colon (':').34146 A question mark ('?') shall be returned if *getopt()* encounters an option character not in
34147 *optstring* or detects a missing argument and the first character of *optstring* was not a colon (':').34148 Otherwise, *getopt()* shall return -1 when all command line options are parsed.34149 **ERRORS**

34150 No errors are defined.

34151 **EXAMPLES**34152 **Parsing Command Line Options**34153 The following code fragment shows how you might process the arguments for a utility that can
34154 take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require
34155 arguments:

```

34156 #include <unistd.h>
34157 int
34158 main(int argc, char *argv[ ])
34159 {
34160     int c;
34161     int bflg, aflg, errflg;
34162     char *ifile;
34163     char *ofile;
34164     extern char *optarg;
34165     extern int optind, optopt;
34166     . . .
34167     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
34168         switch(c) {
34169             case 'a':
34170                 if (bflg)
34171                     errflg++;
34172                 else
34173                     aflg++;
34174                 break;
34175             case 'b':
34176                 if (aflg)
34177                     errflg++;
34178                 else {
34179                     bflg++;
34180                     bproc();
34181                 }
34182                 break;
34183             case 'f':
34184                 ifile = optarg;
34185                 break;
34186             case 'o':
34187                 ofile = optarg;
34188                 break;
34189             case ':': /* -f or -o without operand */
34190                 fprintf(stderr,

```

```

34191         "Option -%c requires an operand\n", optopt);
34192         errflg++;
34193         break;
34194     case '?':
34195         fprintf(stderr,
34196             "Unrecognized option: -%c\n", optopt);
34197         errflg++;
34198     }
34199 }
34200 if (errflg) {
34201     fprintf(stderr, "usage: . . . ");
34202     exit(2);
34203 }
34204 for ( ; optind < argc; optind++) {
34205     if (access(argv[optind], R_OK)) {
34206         . . .
34207     }

```

34208 This code accepts any of the following as equivalent:

```

34209 cmd -ao arg path path
34210 cmd -a -o arg path path
34211 cmd -o arg -a path path
34212 cmd -a -o arg -- path path
34213 cmd -a -oarg path path
34214 cmd -aoarg path path

```

34215 Checking Options and Arguments

34216 The following example parses a set of command line options and prints messages to standard
34217 output for each option and argument that it encounters.

```

34218 #include <unistd.h>
34219 #include <stdio.h>
34220 ...
34221 int c;
34222 char *filename;
34223 extern char *optarg;
34224 extern int optind, optopt, opterr;
34225 ...
34226 while ((c = getopt(argc, argv, ":abf:")) != -1) {
34227     switch(c) {
34228     case 'a':
34229         printf("a is set\n");
34230         break;
34231     case 'b':
34232         printf("b is set\n");
34233         break;
34234     case 'f':
34235         filename = optarg;
34236         printf("filename is %s\n", filename);
34237         break;
34238     case ':':
34239         printf("-%c without filename\n", optopt);
34240         break;
34241     case '?':

```

```

34242         printf("unknown arg %c\n", optopt);
34243         break;
34244     }
34245 }

```

34246 Selecting Options from the Command Line

34247 The following example selects the type of database routines the user wants to use based on the
 34248 *Options* argument.

```

34249 #include <unistd.h>
34250 #include <string.h>
34251 ...
34252 char *Options = "hdbt1";
34253 ...
34254 int dbtype, i;
34255 char c;
34256 char *st;
34257 ...
34258 dbtype = 0;
34259 while ((c = getopt(argc, argv, Options)) != -1) {
34260     if ((st = strchr(Options, c)) != NULL) {
34261         dbtype = st - Options;
34262         break;
34263     }
34264 }

```

34265 APPLICATION USAGE

34266 The *getopt()* function is only required to support option characters included in Utility Syntax
 34267 Guideline 3. Many historical implementations of *getopt()* support other characters as options.
 34268 This is an allowed extension, but applications that use extensions are not maximally portable.
 34269 Note that support for multi-byte option characters is only possible when such characters can be
 34270 represented as type **int**.

34271 RATIONALE

34272 The *optopt* variable represents historical practice and allows the application to obtain the identity
 34273 of the invalid option.

34274 The description has been written to make it clear that *getopt()*, like the *getopts* utility, deals with
 34275 option-arguments whether separated from the option by <blank>s or not. Note that the
 34276 requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

34277 The *getopt()* function shall return -1 , rather than EOF, so that **<stdio.h>** is not required.

34278 The special significance of a colon as the first character of *optstring* makes *getopt()* consistent
 34279 with the *getopts* utility. It allows an application to make a distinction between a missing
 34280 argument and an incorrect option letter without having to examine the option letter. It is true
 34281 that a missing argument can only be detected in one case, but that is a case that has to be
 34282 considered.

34283 FUTURE DIRECTIONS

34284 None.

34285 SEE ALSO

34286 *exec*

34287 XBD Section 12.2 (on page 201), **<unistd.h>**

getopt()

34288

CHANGE HISTORY

34289

First released in Issue 1. Derived from Issue 1 of the SVID.

34290

Issue 5

34291

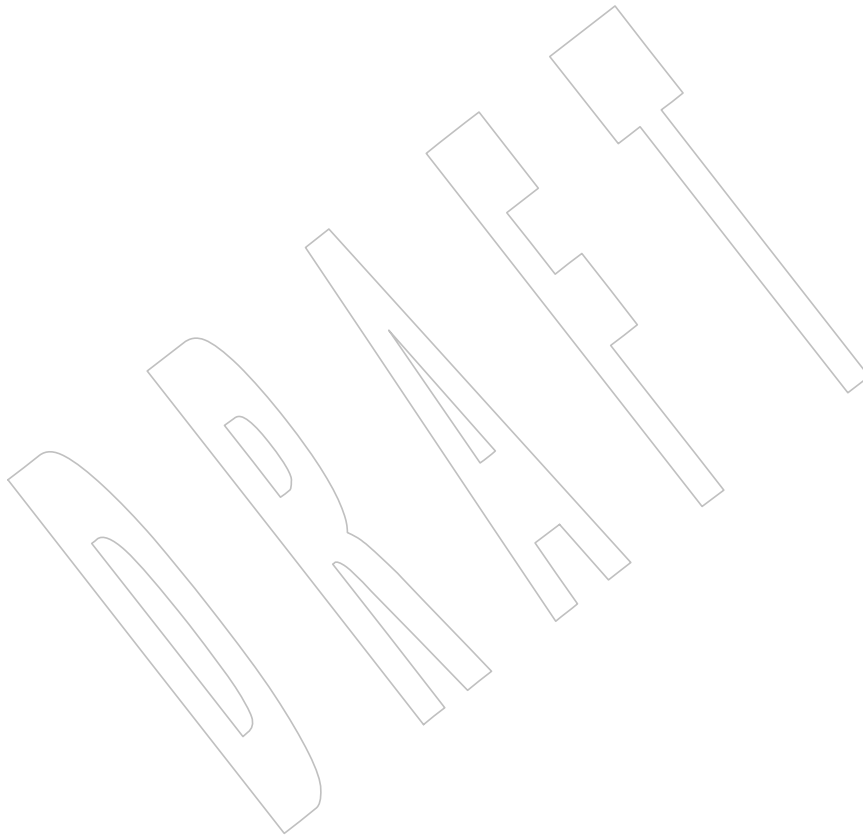
A note indicating that the *getopt()* function need not be reentrant is added to the DESCRIPTION.

34292

Issue 6

34293

IEEE PASC Interpretation 1003.2 #150 is applied.



34294 **NAME**

34295 getpeername — get the name of the peer socket

34296 **SYNOPSIS**

34297 #include <sys/socket.h>

34298 int getpeername(int *socket*, struct sockaddr *restrict *address*,
34299 socklen_t *restrict *address_len*);34300 **DESCRIPTION**34301 The *getpeername()* function shall retrieve the peer address of the specified socket, store this
34302 address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this
34303 address in the object pointed to by the *address_len* argument.34304 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
34305 the stored address shall be truncated.34306 If the protocol permits connections by unbound clients, and the peer is not bound, then the
34307 value stored in the object pointed to by *address* is unspecified.34308 **RETURN VALUE**34309 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
34310 indicate the error.34311 **ERRORS**34312 The *getpeername()* function shall fail if:34313 [EBADF] The *socket* argument is not a valid file descriptor.

34314 [EINVAL] The socket has been shut down.

34315 [ENOTCONN] The socket is not connected or otherwise has not had the peer pre-specified.

34316 [ENOTSOCK] The *socket* argument does not refer to a socket.

34317 [EOPNOTSUPP] The operation is not supported for the socket protocol.

34318 The *getpeername()* function may fail if:

34319 [ENOBUFS] Insufficient resources were available in the system to complete the call.

34320 **EXAMPLES**

34321 None.

34322 **APPLICATION USAGE**

34323 None.

34324 **RATIONALE**

34325 None.

34326 **FUTURE DIRECTIONS**

34327 None.

34328 **SEE ALSO**34329 *accept()*, *bind()*, *getsockname()*, *socket()*

34330 XBD <sys/socket.h>

getpeername()*System Interfaces*

34331

CHANGE HISTORY

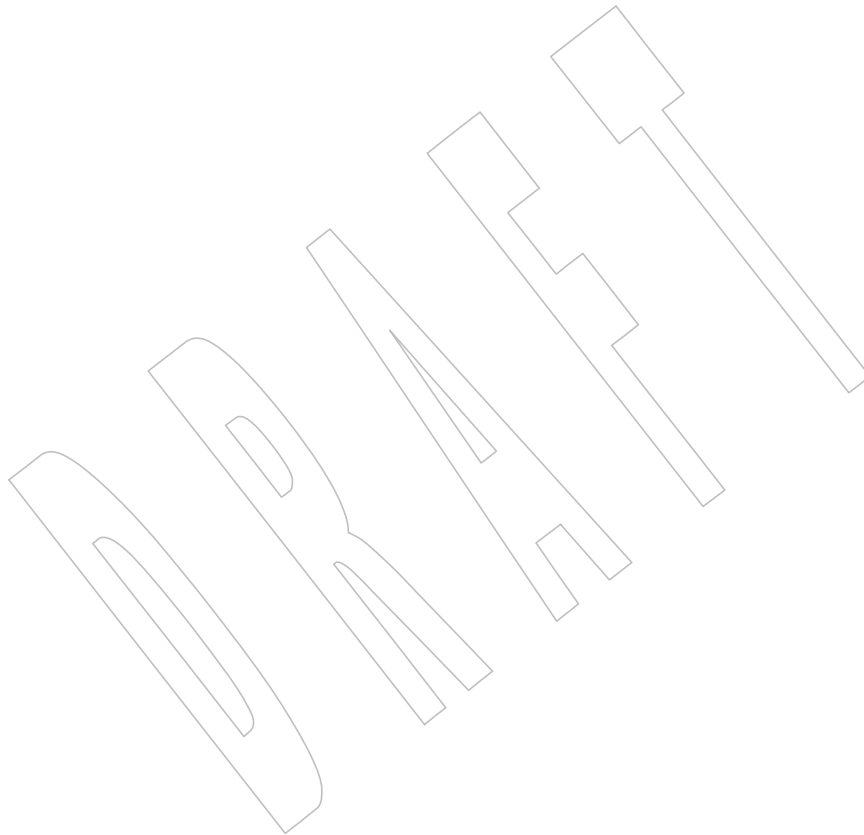
34332

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

34333

The **restrict** keyword is added to the *getpeername()* prototype for alignment with the ISO/IEC 9899:1999 standard.

34334



34335 **NAME**
 34336 getpgid — get the process group ID for a process

34337 **SYNOPSIS**
 34338 #include <unistd.h>
 34339 pid_t getpgid(pid_t pid);

34340 **DESCRIPTION**
 34341 The *getpgid()* function shall return the process group ID of the process whose process ID is equal
 34342 to *pid*. If *pid* is equal to 0, *getpgid()* shall return the process group ID of the calling process.

34343 **RETURN VALUE**
 34344 Upon successful completion, *getpgid()* shall return a process group ID. Otherwise, it shall return
 34345 (**pid_t**)-1 and set *errno* to indicate the error.

34346 **ERRORS**
 34347 The *getpgid()* function shall fail if:
 34348 [EPERM] The process whose process ID is equal to *pid* is not in the same session as the
 34349 calling process, and the implementation does not allow access to the process
 34350 group ID of that process from the calling process.
 34351 [ESRCH] There is no process with a process ID equal to *pid*.

34352 The *getpgid()* function may fail if:
 34353 [EINVAL] The value of the *pid* argument is invalid.

34354 **EXAMPLES**
 34355 None.

34356 **APPLICATION USAGE**
 34357 None.

34358 **RATIONALE**
 34359 None.

34360 **FUTURE DIRECTIONS**
 34361 None.

34362 **SEE ALSO**
 34363 *exec*, *fork()*, *getpgrp()*, *getpid()*, *getsid()*, *setpgrp()*, *setsid()*
 34364 XBD <unistd.h>

34365 **CHANGE HISTORY**
 34366 First released in Issue 4, Version 2.

34367 **Issue 5**
 34368 Moved from X/OPEN UNIX extension to BASE.

34369 **Issue 7**
 34370 The *getpgid()* function is moved from the XSI option to the Base.

34371 **NAME**
 34372 `getpgrp` — get the process group ID of the calling process

34373 **SYNOPSIS**
 34374 `#include <unistd.h>`
 34375 `pid_t getpgrp(void);`

34376 **DESCRIPTION**
 34377 The `getpgrp()` function shall return the process group ID of the calling process.

34378 **RETURN VALUE**
 34379 The `getpgrp()` function shall always be successful and no return value is reserved to indicate an
 34380 error.

34381 **ERRORS**
 34382 No errors are defined.

34383 **EXAMPLES**
 34384 None.

34385 **APPLICATION USAGE**
 34386 None.

34387 **RATIONALE**
 34388 4.3 BSD provides a `getpgrp()` function that returns the process group ID for a specified process.
 34389 Although this function supports job control, all known job control shells always specify the
 34390 calling process with this function. Thus, the simpler System V `getpgrp()` suffices, and the added
 34391 complexity of the 4.3 BSD `getpgrp()` is provided by the XSI extension `getpgid()`.

34392 **FUTURE DIRECTIONS**
 34393 None.

34394 **SEE ALSO**
 34395 *exec, fork(), getpgid(), getpid(), getppid(), kill, setpgid(), setsid()*
 34396 XBD *<sys/types.h>*, *<unistd.h>*

34397 **CHANGE HISTORY**
 34398 First released in Issue 1. Derived from Issue 1 of the SVID.

34399 **Issue 6**
 34400 In the SYNOPSIS, the optional include of the *<sys/types.h>* header is removed.

34401 The following new requirements on POSIX implementations derive from alignment with the
 34402 Single UNIX Specification:

- 34403 • The requirement to include *<sys/types.h>* has been removed. Although *<sys/types.h>* was
 34404 required for conforming implementations of previous POSIX specifications, it was not
 34405 required for UNIX applications.

34406 **NAME**

34407 getpid — get the process ID

34408 **SYNOPSIS**

34409 #include <unistd.h>

34410 pid_t getpid(void);

34411 **DESCRIPTION**34412 The *getpid()* function shall return the process ID of the calling process.34413 **RETURN VALUE**34414 The *getpid()* function shall always be successful and no return value is reserved to indicate an
34415 error.34416 **ERRORS**

34417 No errors are defined.

34418 **EXAMPLES**

34419 None.

34420 **APPLICATION USAGE**

34421 None.

34422 **RATIONALE**

34423 None.

34424 **FUTURE DIRECTIONS**

34425 None.

34426 **SEE ALSO**34427 *exec, fork(), getpgrp(), getppid(), kill, mkdtemp(), setpgid(), setsid()*

34428 XBD <sys/types.h>, <unistd.h>

34429 **CHANGE HISTORY**

34430 First released in Issue 1. Derived from Issue 1 of the SVID.

34431 **Issue 6**

34432 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

34433 The following new requirements on POSIX implementations derive from alignment with the
34434 Single UNIX Specification:

- 34435 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
34436 required for conforming implementations of previous POSIX specifications, it was not
34437 required for UNIX applications.

34438 **NAME**
34439 `getpmsg` — receive next message from a STREAMS file

34440 **SYNOPSIS**

```
34441 OB XSI #include <stropts.h>  
34442 int getpmsg(int fildev, struct strbuf *restrict ctlptr,  
34443 struct strbuf *restrict dataptr, int *restrict bandp,  
34444 int *restrict flagsp);
```

34445 **DESCRIPTION**

34446 Refer to [getmsg\(\)](#).

34447 **NAME**

34448 getppid — get the parent process ID

34449 **SYNOPSIS**34450 #include <unistd.h>
34451 pid_t getppid(void);34452 **DESCRIPTION**34453 The *getppid()* function shall return the parent process ID of the calling process.34454 **RETURN VALUE**34455 The *getppid()* function shall always be successful and no return value is reserved to indicate an
34456 error.34457 **ERRORS**

34458 No errors are defined.

34459 **EXAMPLES**

34460 None.

34461 **APPLICATION USAGE**

34462 None.

34463 **RATIONALE**

34464 None.

34465 **FUTURE DIRECTIONS**

34466 None.

34467 **SEE ALSO**34468 *exec, fork(), getpgid(), getpgrp(), getpid(), kill, setpgid(), setsid()*

34469 XBD <sys/types.h>, <unistd.h>

34470 **CHANGE HISTORY**

34471 First released in Issue 1. Derived from Issue 1 of the SVID.

34472 **Issue 6**

34473 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

34474 The following new requirements on POSIX implementations derive from alignment with the
34475 Single UNIX Specification:

- 34476
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
34477 required for conforming implementations of previous POSIX specifications, it was not
34478 required for UNIX applications.

34479 **NAME**34480 `getpriority, setpriority` — get and set the nice value34481 **SYNOPSIS**

```
34482 XSI      #include <sys/resource.h>
34483
34483      int getpriority(int which, id_t who);
34484      int setpriority(int which, id_t who, int value);
```

34485 **DESCRIPTION**

34486 The `getpriority()` function shall obtain the nice value of a process, process group, or user. The
 34487 `setpriority()` function shall set the nice value of a process, process group, or user to
 34488 `value+{NZERO}`.

34489 Target processes are specified by the values of the `which` and `who` arguments. The `which`
 34490 argument may be one of the following values: `PRIO_PROCESS`, `PRIO_PGRP`, or `PRIO_USER`,
 34491 indicating that the `who` argument is to be interpreted as a process ID, a process group ID, or an
 34492 effective user ID, respectively. A 0 value for the `who` argument specifies the current process,
 34493 process group, or user.

34494 The nice value set with `setpriority()` shall be applied to the process. If the process is multi-
 34495 threaded, the nice value shall affect all system scope threads in the process.

34496 If more than one process is specified, `getpriority()` shall return value `{NZERO}` less than the
 34497 lowest nice value pertaining to any of the specified processes, and `setpriority()` shall set the nice
 34498 values of all of the specified processes to `value+{NZERO}`.

34499 The default nice value is `{NZERO}`; lower nice values shall cause more favorable scheduling.
 34500 While the range of valid nice values is `[0,{NZERO}*2-1]`, implementations may enforce more
 34501 restrictive limits. If `value+{NZERO}` is less than the system's lowest supported nice value,
 34502 `setpriority()` shall set the nice value to the lowest supported value; if `value+{NZERO}` is greater
 34503 than the system's highest supported nice value, `setpriority()` shall set the nice value to the
 34504 highest supported value.

34505 Only a process with appropriate privileges can lower its nice value.

34506 PS|TPS Any processes or threads using `SCHED_FIFO` or `SCHED_RR` shall be unaffected by a call to
 34507 `setpriority()`. This is not considered an error. A process which subsequently reverts to
 34508 `SCHED_OTHER` need not have its priority affected by such a `setpriority()` call.

34509 The effect of changing the nice value may vary depending on the process-scheduling algorithm
 34510 in effect.

34511 Since `getpriority()` can return the value `-1` on successful completion, it is necessary to set `errno` to
 34512 0 prior to a call to `getpriority()`. If `getpriority()` returns the value `-1`, then `errno` can be checked to
 34513 see if an error occurred or if the value is a legitimate nice value.

34514 **RETURN VALUE**

34515 Upon successful completion, `getpriority()` shall return an integer in the range `-{NZERO}` to
 34516 `{NZERO}-1`. Otherwise, `-1` shall be returned and `errno` set to indicate the error.

34517 Upon successful completion, `setpriority()` shall return 0; otherwise, `-1` shall be returned and `errno`
 34518 set to indicate the error.

ERRORS

- 34519 The *getpriority()* and *setpriority()* functions shall fail if:
- 34520
- 34521 [ESRCH] No process could be located using the *which* and *who* argument values
- 34522 specified.
- 34523 [EINVAL] The value of the *which* argument was not recognized, or the value of the *who*
- 34524 argument is not a valid process ID, process group ID, or user ID.
- 34525 In addition, *setpriority()* may fail if:
- 34526 [EPERM] A process was located, but neither the real nor effective user ID of the
- 34527 executing process match the effective user ID of the process whose nice value
- 34528 is being changed.
- 34529 [EACCES] A request was made to change the nice value to a lower numeric value and the
- 34530 current process does not have appropriate privileges.

EXAMPLES**Using *getpriority()***

34532 The following example returns the current scheduling priority for the process ID returned by the

34533 call to *getpid()*.

34534

```
34535 #include <sys/resource.h>
34536 ...
34537 int which = PRIO_PROCESS;
34538 id_t pid;
34539 int ret;
34540
34541 pid = getpid();
34542 ret = getpriority(which, pid);
```

Using *setpriority()*

34543 The following example sets the priority for the current process ID to -20.

```
34544 #include <sys/resource.h>
34545 ...
34546 int which = PRIO_PROCESS;
34547 id_t pid;
34548 int priority = -20;
34549 int ret;
34550
34551 pid = getpid();
34552 ret = setpriority(which, pid, priority);
```

APPLICATION USAGE

34553 The *getpriority()* and *setpriority()* functions work with an offset nice value (nice value

34554 -{NZERO}). The nice value is in the range [0,2*{NZERO} -1], while the return value for

34555 *getpriority()* and the third parameter for *setpriority()* are in the range [-{NZERO},{NZERO} -1].

RATIONALE

34556 None.

34557

FUTURE DIRECTIONS

34558 None.

34559

getpriority()

34560

SEE ALSO

34561

nice, *sched_get_priority_max()*, *sched_setscheduler()*

34562

XBD <[sys/resource.h](#)>

34563

CHANGE HISTORY

34564

First released in Issue 4, Version 2.

34565

Issue 5

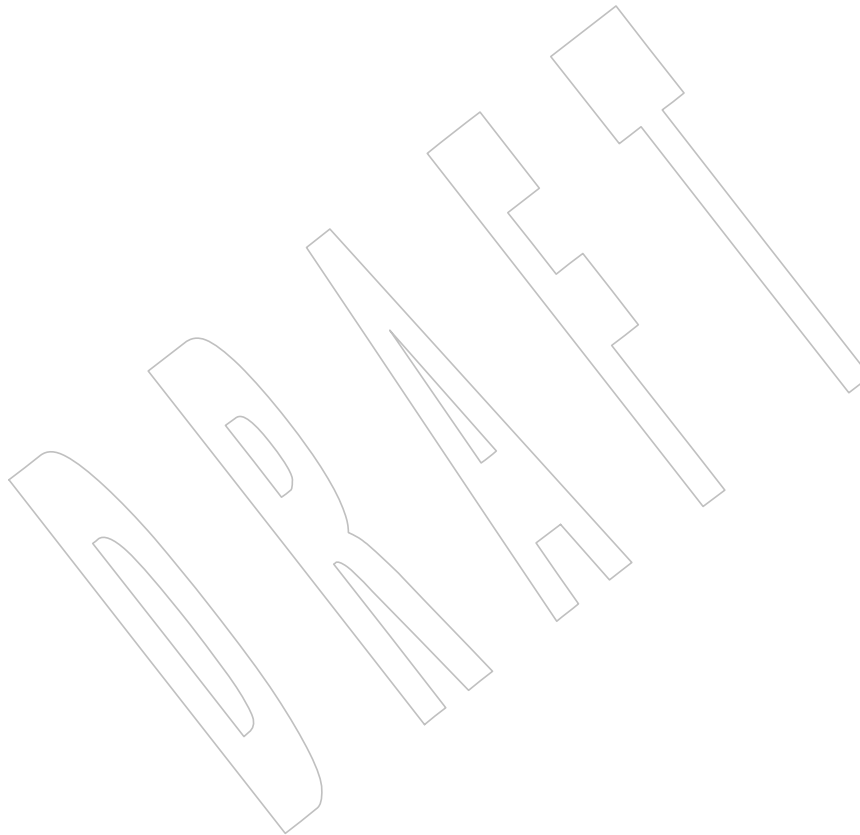
34566

Moved from X/OPEN UNIX extension to BASE.

34567

The DESCRIPTION is reworded in terms of the nice value rather than *priority* to avoid confusion with functionality in the POSIX Realtime Extension.

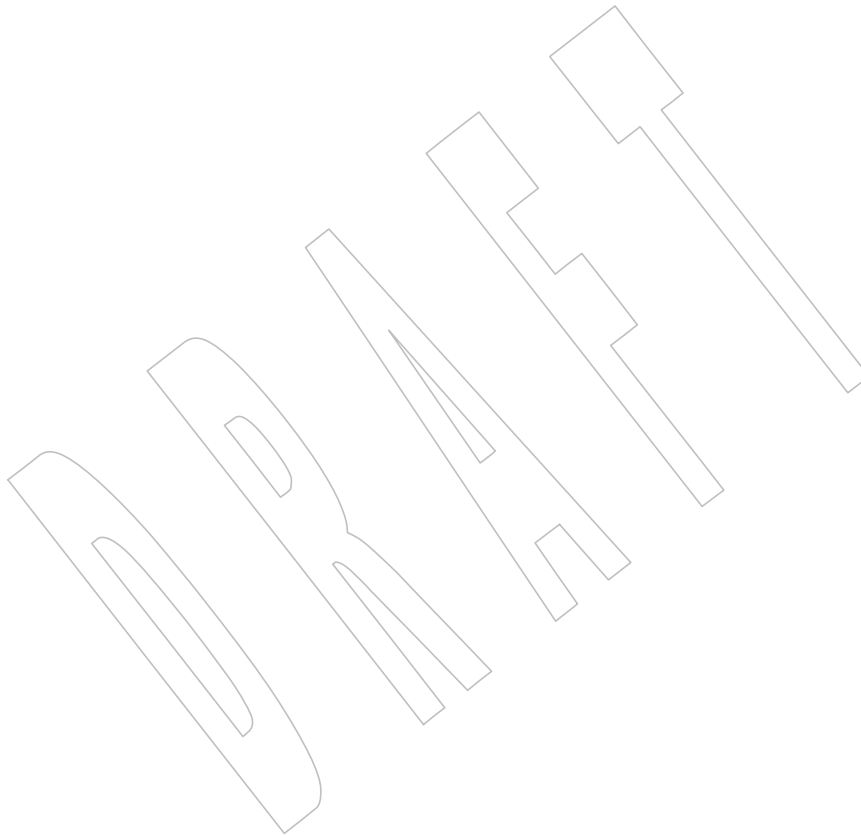
34568



34569 **NAME**
34570 `getprotobyname, getprotobynumber, getprotent` — network protocol database functions

34571 **SYNOPSIS**
34572 `#include <netdb.h>`
34573 `struct protoent *getprotobyname(const char *name);`
34574 `struct protoent *getprotobynumber(int proto);`
34575 `struct protoent *getprotoent(void);`

34576 **DESCRIPTION**
34577 Refer to *endprotoent()*.



getpwent()

34578

NAME

34579

getpwent — get user database entry

34580

SYNOPSIS

34581

XSI `#include <pwd.h>`

34582

`struct passwd *getpwent(void);`

34583

DESCRIPTION

34584

Refer to *endpwent()*.

34585 **NAME**
 34586 `getpwnam, getpwnam_r` — search user database for a name

34587 **SYNOPSIS**
 34588 `#include <pwd.h>`
 34589 `struct passwd *getpwnam(const char *name);`
 34590 `int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,`
 34591 `size_t bufsize, struct passwd **result);`

34592 **DESCRIPTION**
 34593 The `getpwnam()` function shall search the user database for an entry with a matching *name*.
 34594 The `getpwnam()` function need not be thread-safe. A function that is not required to be thread-
 34595 safe is not required to be reentrant.

34596 Applications wishing to check for error situations should set *errno* to 0 before calling
 34597 `getpwnam()`. If `getpwnam()` returns a null pointer and *errno* is non-zero, an error occurred.

34598 The `getpwnam_r()` function shall update the **passwd** structure pointed to by *pwd* and store a
 34599 pointer to that structure at the location pointed to by *result*. The structure shall contain an entry
 34600 from the user database with a matching *name*. Storage referenced by the structure is allocated
 34601 from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to
 34602 `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either -1 without changing *errno* or an initial value
 34603 suggested for the size of this buffer. A NULL pointer shall be returned at the location pointed to
 34604 by *result* on error or if the requested entry is not found.

34605 **RETURN VALUE**
 34606 The `getpwnam()` function shall return a pointer to a **struct passwd** with the structure as defined
 34607 in `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested
 34608 entry is not found, or an error occurs. On error, *errno* shall be set to indicate the error.

34609 The return value may point to a static area which is overwritten by a subsequent call to
 34610 `getpwent()`, `getpwnam()`, or `getpwuid()`.

34611 The `getpwnam_r()` function shall return zero on success or if the requested entry was not found
 34612 and no error has occurred. If an error has occurred, an error number shall be returned to indicate
 34613 the error.

34614 **ERRORS**
 34615 The `getpwnam()` and `getpwnam_r()` functions may fail if:

34616 [EIO] An I/O error has occurred.
 34617 [EINTR] A signal was caught during `getpwnam()`.
 34618 [EMFILE] All file descriptors available to the process are currently open.
 34619 [ENFILE] The maximum allowable number of files is currently open in the system.
 34620 The `getpwnam_r()` function may fail if:
 34621 [ERANGE] Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be
 34622 referenced by the resulting **passwd** structure.

EXAMPLES

Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with `getpwnam_r()`.

```

34627 long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
34628 size_t len;
34629 if (initlen == -1)
34630     /* Default initial length. */
34631     len = 1024;
34632 else
34633     len = (size_t) initlen;
34634 struct passwd result;
34635 struct passwd *resultp;
34636 char *buffer = malloc(len);
34637 if (buffer == NULL)
34638     ...handle error...
34639 int e;
34640 while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
34641     == ERANGE)
34642     {
34643     size_t newlen = 2 * len;
34644     if (newlen < len)
34645         ...handle error...
34646     len = newlen;
34647     char *newbuffer = realloc(buffer, len);
34648     if (newbuffer == NULL)
34649         ...handle error...
34650     buffer = newbuffer;
34651     }
34652 if (e != 0)
34653     ...handle error...

```

Getting an Entry for the Login Name

The following example uses the `getlogin()` function to return the name of the user who logged in; this information is passed to the `getpwnam()` function to get the user database entry for that user.

```

34657 #include <sys/types.h>
34658 #include <pwd.h>
34659 #include <unistd.h>
34660 #include <stdio.h>
34661 #include <stdlib.h>
34662 ...
34663 char *lgn;
34664 struct passwd *pw;
34665 ...
34666 if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
34667     fprintf(stderr, "Get of user information failed.\n"); exit(1);
34668 }
34669 ...

```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid(getuid())* returns the name associated with the effective user ID of the process; *getlogin()* returns the name associated with the current login activity; and *getpwuid(getuid())* returns the name associated with the real user ID of the process.

The *getpwnam_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return -1 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwuid(), *sysconf()*

XBD `<limits.h>`, `<pwd.h>`, `<sys/types.h>`

CHANGE HISTORY

First released in Issue 1. Derived from System V Release 2.0.

Issue 5

Normative text previously in the APPLICATION USAGE section is moved to the RETURN VALUE section.

The *getpwnam_r()* function is included for alignment with the POSIX Threads Extension.

A note indicating that the *getpwnam()* function need not be reentrant is added to the DESCRIPTION.

Issue 6

The *getpwnam_r()* function is marked as part of the Thread-Safe Functions option.

The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION describing matching the *name*.

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the RETURN VALUE section, the requirement to set *errno* on error is added.
- The [EMFILE], [ENFILE], and [ENXIO] optional error conditions are added.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the buffer from *bufsize* characters to bytes.

getpwnam()

34711

Issue 7

34712

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

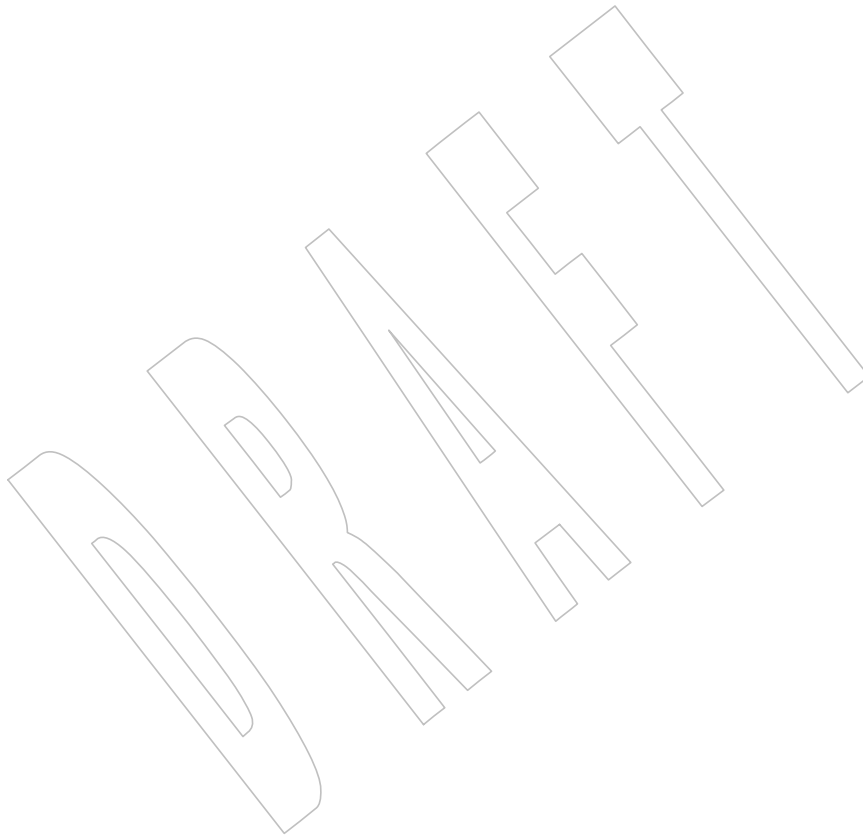
34713

SD5-XSH-ERN-166 is applied.

34714

The *getpwnam_r()* function is moved from the Thread-Safe Functions option to the Base.

+



34715 **NAME**34716 `getpwuid, getpwuid_r` — search user database for a user ID34717 **SYNOPSIS**34718 `#include <pwd.h>`34719 `struct passwd *getpwuid(uid_t uid);`34720 `int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,`
34721 `size_t bufsize, struct passwd **result);`34722 **DESCRIPTION**34723 The `getpwuid()` function shall search the user database for an entry with a matching `uid`.34724 The `getpwuid()` function need not be thread-safe. A function that is not required to be thread-safe
34725 is not required to be reentrant.34726 Applications wishing to check for error situations should set `errno` to 0 before calling `getpwuid()`.
34727 If `getpwuid()` returns a null pointer and `errno` is set to non-zero, an error occurred.34728 The `getpwuid_r()` function shall update the `passwd` structure pointed to by `pwd` and store a
34729 pointer to that structure at the location pointed to by `result`. The structure shall contain an entry
34730 from the user database with a matching `uid`. Storage referenced by the structure is allocated
34731 from the memory provided with the `buffer` parameter, which is `bufsize` bytes in size. A call to
34732 `sysconf(_SC_GETPW_R_SIZE_MAX)` returns either -1 without changing `errno` or an initial value
34733 suggested for the size of this buffer. A NULL pointer shall be returned at the location pointed to
34734 by `result` on error or if the requested entry is not found.34735 **RETURN VALUE**34736 The `getpwuid()` function shall return a pointer to a `struct passwd` with the structure as defined in
34737 `<pwd.h>` with a matching entry if found. A null pointer shall be returned if the requested entry
34738 is not found, or an error occurs. On error, `errno` shall be set to indicate the error.34739 The return value may point to a static area which is overwritten by a subsequent call to
34740 `getpwent()`, `getpwnam()`, or `getpwuid()`.34741 If successful, the `getpwuid_r()` function shall return zero; otherwise, an error number shall be
34742 returned to indicate the error.34743 **ERRORS**34744 The `getpwuid()` and `getpwuid_r()` functions may fail if:

34745 [EIO] An I/O error has occurred.

34746 [EINTR] A signal was caught during `getpwuid()`.

34747 [EMFILE] All file descriptors available to the process are currently open.

34748 [ENFILE] The maximum allowable number of files is currently open in the system.

34749 The `getpwuid_r()` function may fail if:34750 [ERANGE] Insufficient storage was supplied via `buffer` and `bufsize` to contain the data to be
34751 referenced by the resulting `passwd` structure.

EXAMPLES

Note that `sysconf(_SC_GETPW_R_SIZE_MAX)` may return `-1` if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with `getpwuid_r()`.

```

34752
34753
34754
34755
34756
34757
34758
34759
34760
34761
34762
34763
34764
34765
34766
34767
34768
34769
34770
34771
34772
34773
34774
34775
34776
34777
34778
34779
34780
34781
34782
34783
34784
34785
34786
34787
34788
34789
34790
34791
34792
34793
34794
34795
34796
34797
34798
34799

```

```

long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
{
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
}
if (e != 0)
    ...handle error...

```

Getting an Entry for the Root User

The following example gets the user database entry for the user with user ID 0 (root).

```

#include <sys/types.h>
#include <pwd.h>
...
uid_t id = 0;
struct passwd *pwd;

pwd = getpwuid(id);

```

Finding the Name for the Effective User ID

The following example defines `pws` as a pointer to a structure of type `passwd`, which is used to store the structure pointer returned by the call to the `getpwuid()` function. The `geteuid()` function shall return the effective user ID of the calling process; this is used as the search criteria for the `getpwuid()` function. The call to `getpwuid()` shall return a pointer to the structure containing that user ID value.

```

#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
...

```

```

34800     struct passwd *pws;
34801     pws = getpwuid(geteuid());

```

34802 Finding an Entry in the User Database

34803 The following example uses *getpwuid()* to search the user database for a user ID that was
 34804 previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not
 34805 found, the program prints the numeric value of the user ID for the entry.

```

34806     #include <sys/types.h>
34807     #include <pwd.h>
34808     #include <stdio.h>
34809     ...
34810     struct stat statbuf;
34811     struct passwd *pwd;
34812     ...
34813     if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
34814         printf(" %-8.8s", pwd->pw_name);
34815     else
34816         printf(" %-8d", statbuf.st_uid);

```

34817 APPLICATION USAGE

34818 Three names associated with the current process can be determined: *getpwuid(geteuid())* returns
 34819 the name associated with the effective user ID of the process; *getlogin()* returns the name
 34820 associated with the current login activity; and *getpwuid(getuid())* returns the name associated
 34821 with the real user ID of the process.

34822 The *getpwuid_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 34823 possibly using a static data area that may be overwritten by each call.

34824 Portable applications should take into account that it is usual for an implementation to return -1 +
 34825 from *sysconf()* indicating that there is no maximum for `_SC_GETPW_R_SIZE_MAX`.

34826 RATIONALE

34827 None.

34828 FUTURE DIRECTIONS

34829 None.

34830 SEE ALSO

34831 *getpwnam()*, *geteuid()*, *getuid()*, *getlogin()*, *sysconf()*

34832 XBD `<limits.h>`, `<pwd.h>`, `<sys/types.h>`

34833 CHANGE HISTORY

34834 First released in Issue 1. Derived from System V Release 2.0.

34835 Issue 5

34836 Normative text previously in the APPLICATION USAGE section is moved to the RETURN
 34837 VALUE section.

34838 The *getpwuid_r()* function is included for alignment with the POSIX Threads Extension.

34839 A note indicating that the *getpwuid()* function need not be reentrant is added to the
 34840 DESCRIPTION.

34841 Issue 6

34842 The *getpwuid_r()* function is marked as part of the Thread-Safe Functions option.

34843 The Open Group Corrigendum U028/3 is applied, correcting text in the DESCRIPTION
 34844 describing matching the *uid*.

34845 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

34846 In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

34847 The following new requirements on POSIX implementations derive from alignment with the
34848 Single UNIX Specification:

- 34849 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
34850 required for conforming implementations of previous POSIX specifications, it was not
34851 required for UNIX applications.
- 34852 • In the RETURN VALUE section, the requirement to set *errno* on error is added.
- 34853 • The [EIO], [EINTR], [EMFILE], and [ENFILE] optional error conditions are added.

34854 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
34855 its avoidance of possibly using a static data area.

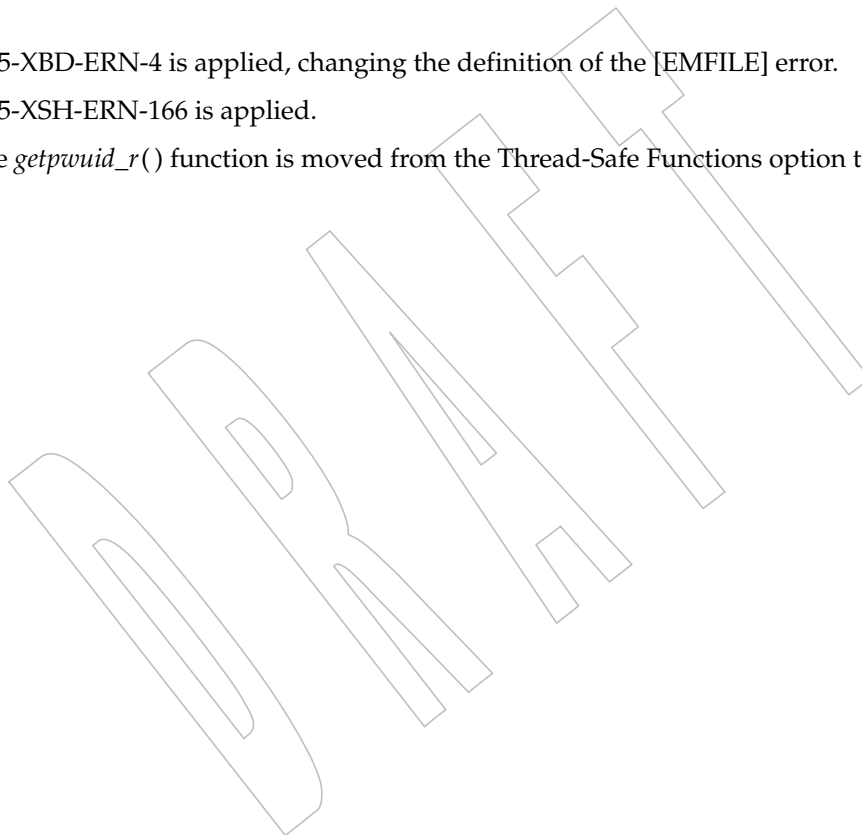
34856 IEEE PASC Interpretation 1003.1 #116 is applied, changing the description of the size of the
34857 buffer from *bufsize* characters to bytes.

34858 **Issue 7**

34859 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

34860 SD5-XSH-ERN-166 is applied. +

34861 The *getpwuid_r()* function is moved from the Thread-Safe Functions option to the Base.



34862 **NAME**
 34863 getrlimit, setrlimit — control maximum resource consumption

34864 **SYNOPSIS**

```
34865 XSI #include <sys/resource.h>
34866 int getrlimit(int resource, struct rlimit *rlp);
34867 int setrlimit(int resource, const struct rlimit *rlp);
```

34868 **DESCRIPTION**

34869 The *getrlimit()* function shall get, and the *setrlimit()* function shall set, limits on the consumption
 34870 of a variety of resources.

34871 Each call to either *getrlimit()* or *setrlimit()* identifies a specific resource to be operated upon as
 34872 well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur*
 34873 member specifies the current or soft limit and the *rlim_max* member specifies the maximum or
 34874 hard limit. Soft limits may be changed by a process to any value that is less than or equal to the
 34875 hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or
 34876 equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both
 34877 hard and soft limits can be changed in a single call to *setrlimit()* subject to the constraints
 34878 described above.

34879 The value RLIM_INFINITY, defined in **<sys/resource.h>**, shall be considered to be larger than
 34880 any other limit value. If a call to *getrlimit()* returns RLIM_INFINITY for a resource, it means the
 34881 implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any
 34882 resource limit value on a successful call to *setrlimit()* shall inhibit enforcement of that resource
 34883 limit.

34884 The following resources are defined:

34885 RLIMIT_CORE This is the maximum size of a **core** file, in bytes, that may be created by a
 34886 process. A limit of 0 shall prevent the creation of a **core** file. If this limit is
 34887 exceeded, the writing of a **core** file shall terminate at this size.

34888 RLIMIT_CPU This is the maximum amount of CPU time, in seconds, used by a process.
 34889 If this limit is exceeded, SIGXCPU shall be generated for the process. If
 34890 the process is catching or ignoring SIGXCPU, or all threads belonging to
 34891 that process are blocking SIGXCPU, the behavior is unspecified.

34892 RLIMIT_DATA This is the maximum size of a data segment of the process, in bytes. If
 34893 this limit is exceeded, the *malloc()* function shall fail with *errno* set to
 34894 [ENOMEM].

34895 RLIMIT_FSIZE This is the maximum size of a file, in bytes, that may be created by a
 34896 process. If a write or truncate operation would cause this limit to be
 34897 exceeded, SIGXFSZ shall be generated for the thread. If the thread is
 34898 blocking, or the process is catching or ignoring SIGXFSZ, continued
 34899 attempts to increase the size of a file from end-of-file to beyond the limit
 34900 shall fail with *errno* set to [EFBIG].

34901 RLIMIT_NOFILE This is a number one greater than the maximum value that the system
 34902 may assign to a newly-created descriptor. If this limit is exceeded,
 34903 functions that allocate a file descriptor shall fail with *errno* set to
 34904 [EMFILE]. This limit constrains the number of file descriptors that a
 34905 process may allocate.

34906	RLIMIT_STACK	This is the maximum size of the initial thread's stack, in bytes. The implementation does not automatically grow the stack beyond this limit. If this limit is exceeded, SIGSEGV shall be generated for the thread. If the thread is blocking SIGSEGV, or the process is ignoring or catching SIGSEGV and has not made arrangements to use an alternate stack, the disposition of SIGSEGV shall be set to SIG_DFL before it is generated.
34907		
34908		
34909		
34910		
34911		
34912	RLIMIT_AS	This is the maximum size of total available memory of the process, in bytes. If this limit is exceeded, the <i>malloc()</i> and <i>mmap()</i> functions shall fail with <i>errno</i> set to [ENOMEM]. In addition, the automatic stack growth fails with the effects outlined above.
34913		
34914		
34915		

34916 When using the *getrlimit()* function, if a resource limit can be represented correctly in an object
 34917 of type **rlim_t**, then its representation is returned; otherwise, if the value of the resource limit is
 34918 equal to that of the corresponding saved hard limit, the value returned shall be
 34919 RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.

34920 When using the *setrlimit()* function, if the requested new limit is RLIM_INFINITY, the new limit
 34921 shall be "no limit"; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit
 34922 shall be the corresponding saved hard limit; otherwise, if the requested new limit is
 34923 RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the
 34924 new limit shall be the requested value. In addition, if the corresponding saved limit can be
 34925 represented correctly in an object of type **rlim_t** then it shall be overwritten with the new limit.

34926 The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless
 34927 a previous call to *getrlimit()* returned that value as the soft or hard limit for the corresponding
 34928 resource limit.

34929 The determination of whether a limit can be correctly represented in an object of type **rlim_t** is
 34930 implementation-defined. For example, some implementations permit a limit whose value is
 34931 greater than RLIM_INFINITY and others do not.

34932 The *exec* family of functions shall cause resource limits to be saved.

34933 RETURN VALUE

34934 Upon successful completion, *getrlimit()* and *setrlimit()* shall return 0. Otherwise, these functions
 34935 shall return -1 and set *errno* to indicate the error.

34936 ERRORS

34937 The *getrlimit()* and *setrlimit()* functions shall fail if:

34938	[EINVAL]	An invalid <i>resource</i> was specified; or in a <i>setrlimit()</i> call, the new <i>rlim_cur</i> 34939 exceeds the new <i>rlim_max</i> .
34940	[EPERM]	The limit specified to <i>setrlimit()</i> would have raised the maximum limit value, 34941 and the calling process does not have appropriate privileges.

34942 The *setrlimit()* function may fail if:

34943	[EINVAL]	The limit specified cannot be lowered because current usage is already higher 34944 than the limit.
-------	----------	--

34945
34946
34947
34948
34949
34950
34951
34952
34953
34954
34955
34956
34957
34958
34959
34960
34961
34962
34963
34964
34965
34966
34967
34968
34969
34970
34971
34972
34973
34974
34975

EXAMPLES

None.

APPLICATION USAGE

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value of `{_POSIX_OPEN_MAX}` from `<limits.h>`, unexpected behavior may occur.

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the highest currently open file descriptor +1, unexpected behavior may occur.

RATIONALE

It should be noted that RLIMIT_STACK applies “at least” to the stack of the initial thread in the process, and not to the sum of all the stacks in the process, as that would be very limiting unless the value is so big as to provide no value at all with a single thread.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, fork(), malloc(), open(), sigaltstack(), sysconf(), ulimit

XBD `<stropts.h>`, `<sys/resource.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

An APPLICATION USAGE section is added.

Large File Summit extensions are added.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/25 is applied, changing wording for RLIMIT_NOFILE in the DESCRIPTION related to functions that allocate a file descriptor failing with [EMFILE]. Text is added to the APPLICATION USAGE section noting the consequences of a process attempting to set the hard or soft limit for RLIMIT_NOFILE less than the highest currently open file descriptor +1.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/46 is applied, updating the definition of RLIMIT_STACK in the DESCRIPTION from “the maximum size of a process stack” to “the maximum size of the initial thread’s stack”. Text is added to the RATIONALE section.

34976 **NAME**
 34977 getrusage — get information about resource utilization

34978 **SYNOPSIS**

34979 XSI `#include <sys/resource.h>`
 34980 `int getrusage(int who, struct rusage *r_usage);`

34981 **DESCRIPTION**

34982 The *getrusage()* function shall provide measures of the resources used by the current process or
 34983 its terminated and waited-for child processes. If the value of the *who* argument is
 34984 RUSAGE_SELF, information shall be returned about resources used by the current process. If the
 34985 value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about
 34986 resources used by the terminated and waited-for children of the current process. If the child is
 34987 never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to
 34988 SIG_IGN), the resource information for the child process is discarded and not included in the
 34989 resource information provided by *getrusage()*.

34990 The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned
 34991 information is stored.

34992 **RETURN VALUE**

34993 Upon successful completion, *getrusage()* shall return 0; otherwise, -1 shall be returned and *errno*
 34994 set to indicate the error.

34995 **ERRORS**

34996 The *getrusage()* function shall fail if:
 34997 [EINVAL] The value of the *who* argument is not valid.

34998 **EXAMPLES**

34999 **Using getrusage()**

35000 The following example returns information about the resources used by the current process.

```
35001 #include <sys/resource.h>
35002 ...
35003 int who = RUSAGE_SELF;
35004 struct rusage usage;
35005 int ret;
35006
35007 ret = getrusage(who, &usage);
```

35007 **APPLICATION USAGE**

35008 None.

35009 **RATIONALE**

35010 None.

35011 **FUTURE DIRECTIONS**

35012 None.

35013 **SEE ALSO**

35014 *exit()*, *sigaction()*, *time*, *times()*, *wait*

35015 XBD [<sys/resource.h>](#)

35016

CHANGE HISTORY

35017

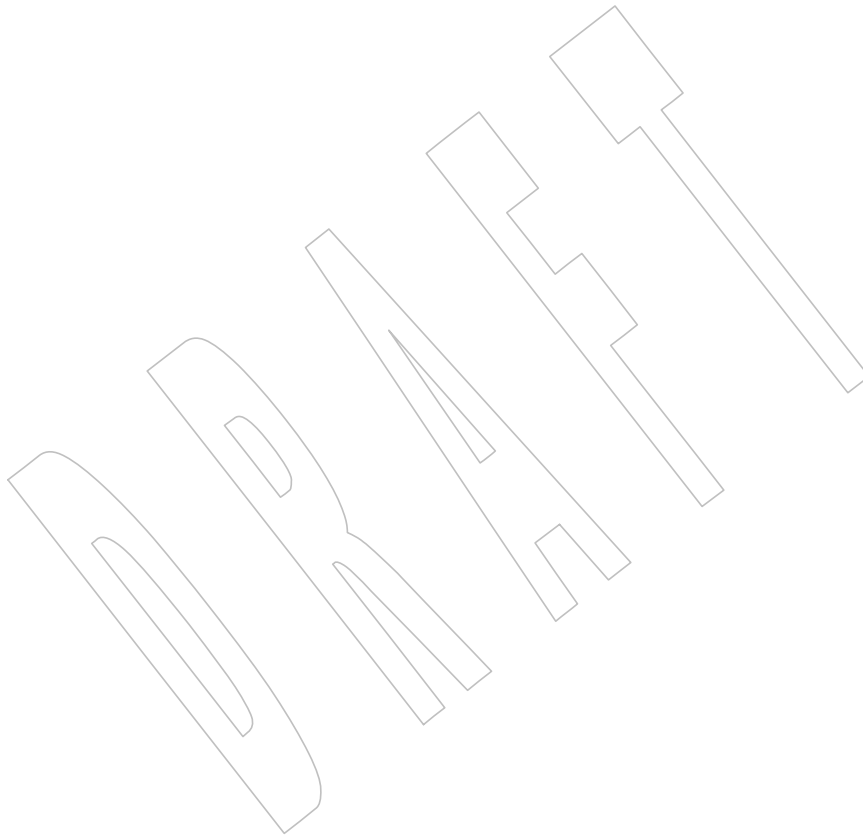
First released in Issue 4, Version 2.

35018

Issue 5

35019

Moved from X/OPEN UNIX extension to BASE.



gets()35020 **NAME**

35021 gets — get a string from a stdin stream

35022 **SYNOPSIS**

```
35023 OB #include <stdio.h>
35024 char *gets(char *s);
```

35025 **DESCRIPTION**

35026 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 35027 conflict between the requirements described here and the ISO C standard is unintentional. This
 35028 volume of POSIX.1-200x defers to the ISO C standard.

35029 The *gets()* function shall read bytes from the standard input stream, *stdin*, into the array pointed
 35030 to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall
 35031 be discarded and a null byte shall be placed immediately after the last byte read into the array.

35032 CX The *gets()* function may mark the last data access timestamp of the file associated with *stream* for
 35033 update. The last data access timestamp shall be marked for update by the first successful
 35034 execution of *fgetc()*, *fgets()*, *fread()*, *getc()*, *getchar()*, *gets()*, *fscanf()*, or *scanf()* using *stream* that
 35035 returns data not supplied by a prior call to *ungetc()*.

35036 **RETURN VALUE**

35037 Upon successful completion, *gets()* shall return *s*. If the end-of-file indicator for the stream is set,
 35038 or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *gets()*
 35039 shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set,
 35040 CX *gets()* shall return a null pointer, and set *errno* to indicate the error.

35041 **ERRORS**35042 Refer to *fgetc()*.35043 **EXAMPLES**

35044 None.

35045 **APPLICATION USAGE**

35046 Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of
 35047 *fgets()* is recommended.

35048 Since the user cannot specify the length of the buffer passed to *gets()*, use of this function is
 35049 discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in
 35050 such a way as to cause applications to fail, or possible system security violations.

35051 Applications should use the *fgets()* function instead of the obsolescent *gets()* function. |

35052 **RATIONALE**

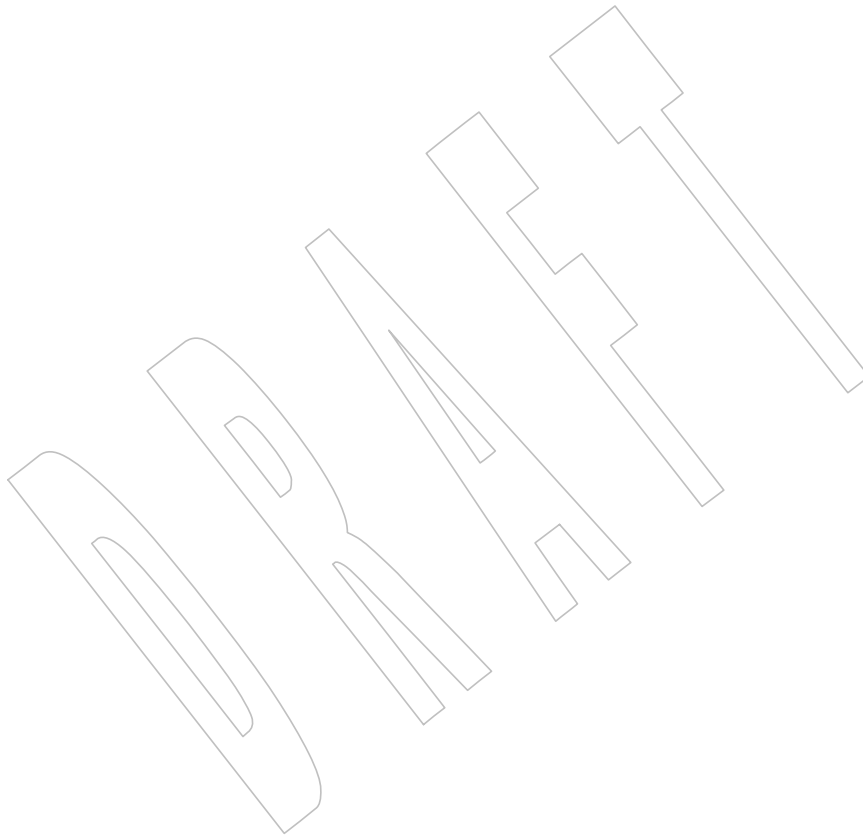
35053 The standard developers decided to mark the *gets()* function as obsolescent even though it is in |
 35054 the ISO C standard due to the possibility of buffer overflow.

35055 **FUTURE DIRECTIONS**

35056 The *gets()* function may be removed in a future version. |

35057 **SEE ALSO**35058 *feof()*, *ferror()*, *fgets()*35059 XBD [<stdio.h>](#) |

35060	CHANGE HISTORY
35061	First released in Issue 1. Derived from Issue 1 of the SVID.
35062	Issue 6
35063	Extensions beyond the ISO C standard are marked.
35064	Issue 7
35065	Austin Group Interpretation 1003.1-2001 #051 is applied, clarifying the RETURN VALUE section.
35066	The <i>gets()</i> function is marked obsolescent. +
35067	Changes are made related to support for finegrained timestamps.



35068 **NAME**
35069 getservbyname, getservbyport, getservent — network services database functions

35070 **SYNOPSIS**
35071 #include <netdb.h>
35072 struct servent *getservbyname(const char *name, const char *proto);
35073 struct servent *getservbyport(int port, const char *proto);
35074 struct servent *getservent(void);

35075 **DESCRIPTION**
35076 Refer to *endservent()*.



35077 **NAME**

35078 getsid — get the process group ID of a session leader

35079 **SYNOPSIS**

35080 #include <unistd.h>

35081 pid_t getsid(pid_t pid);

35082 **DESCRIPTION**35083 The *getsid()* function shall obtain the process group ID of the process that is the session leader of
35084 the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.35085 **RETURN VALUE**35086 Upon successful completion, *getsid()* shall return the process group ID of the session leader of
35087 the specified process. Otherwise, it shall return (**pid_t**)-1 and set *errno* to indicate the error.35088 **ERRORS**35089 The *getsid()* function shall fail if:35090 [EPERM] The process specified by *pid* is not in the same session as the calling process,
35091 and the implementation does not allow access to the process group ID of the
35092 session leader of that process from the calling process.35093 [ESRCH] There is no process with a process ID equal to *pid*.35094 **EXAMPLES**

35095 None.

35096 **APPLICATION USAGE**

35097 None.

35098 **RATIONALE**

35099 None.

35100 **FUTURE DIRECTIONS**

35101 None.

35102 **SEE ALSO**35103 *exec*, *fork()*, *getpid()*, *getpgid()*, *setpgid()*, *setsid()*

35104 XBD <unistd.h>

35105 **CHANGE HISTORY**

35106 First released in Issue 4, Version 2.

35107 **Issue 5**

35108 Moved from X/OPEN UNIX extension to BASE.

35109 **Issue 7**35110 The *getsid()* function is moved from the XSI option to the Base.

getsockname()35111 **NAME**35112 `getsockname` — get the socket name35113 **SYNOPSIS**35114 `#include <sys/socket.h>`35115 `int getsockname(int socket, struct sockaddr *restrict address,`
35116 `socklen_t *restrict address_len);`35117 **DESCRIPTION**35118 The `getsockname()` function shall retrieve the locally-bound name of the specified socket, store
35119 this address in the **sockaddr** structure pointed to by the `address` argument, and store the length of
35120 this address in the object pointed to by the `address_len` argument.35121 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
35122 the stored address shall be truncated.35123 If the socket has not been bound to a local name, the value stored in the object pointed to by
35124 `address` is unspecified.35125 **RETURN VALUE**35126 Upon successful completion, 0 shall be returned, the `address` argument shall point to the address
35127 of the socket, and the `address_len` argument shall point to the length of the address. Otherwise, -1
35128 shall be returned and `errno` set to indicate the error.35129 **ERRORS**35130 The `getsockname()` function shall fail if:35131 [EBADF] The `socket` argument is not a valid file descriptor.35132 [ENOTSOCK] The `socket` argument does not refer to a socket.

35133 [EOPNOTSUPP] The operation is not supported for this socket's protocol.

35134 The `getsockname()` function may fail if:

35135 [EINVAL] The socket has been shut down.

35136 [ENOBUFS] Insufficient resources were available in the system to complete the function.

35137 **EXAMPLES**

35138 None.

35139 **APPLICATION USAGE**

35140 None.

35141 **RATIONALE**

35142 None.

35143 **FUTURE DIRECTIONS**

35144 None.

35145 **SEE ALSO**35146 [*accept\(\)*](#), [*bind\(\)*](#), [*getpeername\(\)*](#), [*socket\(\)*](#)35147 XBD [*<sys/socket.h>*](#)

35148

CHANGE HISTORY

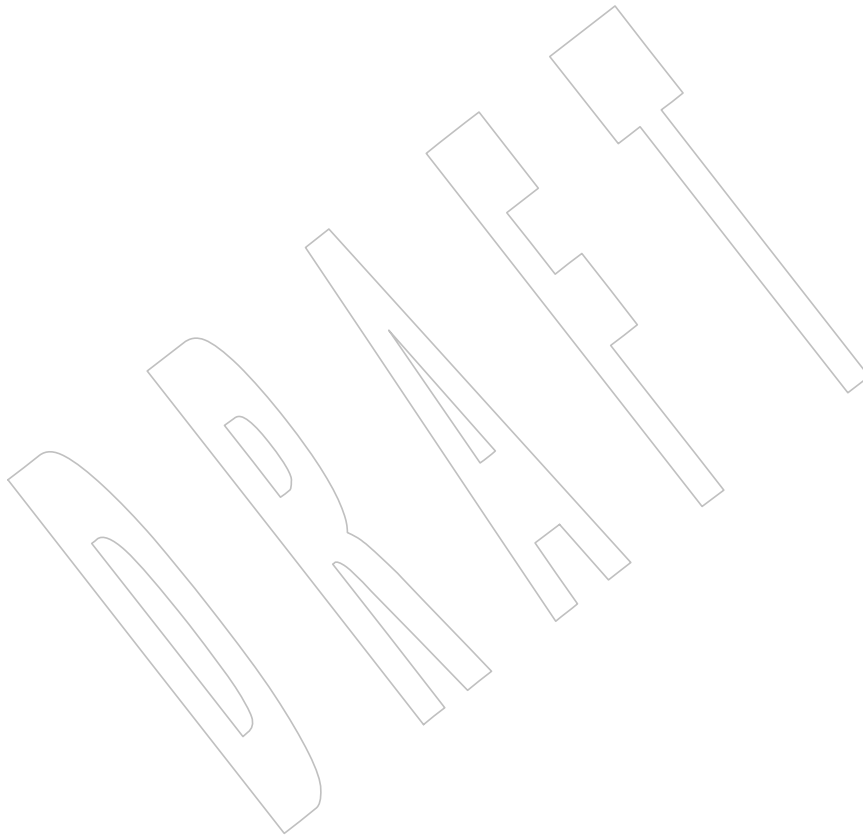
35149

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35150

The **restrict** keyword is added to the *getsockname()* prototype for alignment with the ISO/IEC 9899:1999 standard.

35151



35152 **NAME**

35153 getsockopt — get the socket options

35154 **SYNOPSIS**

```
35155 #include <sys/socket.h>
35156
35157 int getsockopt(int socket, int level, int option_name,
35158               void *restrict option_value, socklen_t *restrict option_len);
```

35158 **DESCRIPTION**35159 The *getsockopt()* function manipulates options associated with a socket.

35160 The *getsockopt()* function shall retrieve the value for the option specified by the *option_name*
 35161 argument for the socket specified by the *socket* argument. If the size of the option value is greater
 35162 than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be
 35163 silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified
 35164 to indicate the actual length of the value.

35165 The *level* argument specifies the protocol level at which the option resides. To retrieve options at
 35166 the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels,
 35167 supply the appropriate level identifier for the protocol controlling the option. For example, to
 35168 indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to
 35169 IPPROTO_TCP as defined in the **<netinet/in.h>** header.

35170 The socket in use may require the process to have appropriate privileges to use the *getsockopt()*
 35171 function.

35172 The *option_name* argument specifies a single option to be retrieved. It can be one of the following
 35173 values defined in **<sys/socket.h>**:

35174 SO_DEBUG Reports whether debugging information is being recorded. This option
 35175 shall store an **int** value. This is a Boolean option.

35176 SO_ACCEPTCONN Reports whether socket listening is enabled. This option shall store an **int**
 35177 value. This is a Boolean option.

35178 SO_BROADCAST Reports whether transmission of broadcast messages is supported, if this
 35179 is supported by the protocol. This option shall store an **int** value. This is a
 35180 Boolean option.

35181 SO_REUSEADDR Reports whether the rules used in validating addresses supplied to *bind()*
 35182 should allow reuse of local addresses, if this is supported by the protocol.
 35183 This option shall store an **int** value. This is a Boolean option.

35184 SO_KEEPALIVE Reports whether connections are kept active with periodic transmission
 35185 of messages, if this is supported by the protocol.

35186 If the connected socket fails to respond to these messages, the connection
 35187 shall be broken and threads writing to that socket shall be notified with a
 35188 SIGPIPE signal. This option shall store an **int** value. This is a Boolean
 35189 option.

35190 SO_LINGER Reports whether the socket lingers on *close()* if data is present. If
 35191 SO_LINGER is set, the system shall block the calling thread during *close()*
 35192 until it can transmit the data or until the end of the interval indicated by
 35193 the *l_linger* member, whichever comes first. If SO_LINGER is not
 35194 specified, and *close()* is issued, the system handles the call in a way that
 35195 allows the calling thread to continue as quickly as possible. This option
 35196 shall store a **linger** structure.

35197	SO_OOBINLINE	Reports whether the socket leaves received out-of-band data (data marked urgent) inline. This option shall store an int value. This is a Boolean option.
35198		
35199		
35200	SO_SNDBUF	Reports send buffer size information. This option shall store an int value.
35201	SO_RCVBUF	Reports receive buffer size information. This option shall store an int value.
35202		
35203	SO_ERROR	Reports information about error status and clears it. This option shall store an int value.
35204		
35205	SO_TYPE	Reports the socket type. This option shall store an int value. Socket types are described in Section 2.10.6 (on page 497).
35206		
35207	SO_DONTROUTE	Reports whether outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option shall store an int value. This is a Boolean option.
35208		
35209		
35210		
35211		
35212		
35213	SO_RCVLOWAT	Reports the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option shall store an int value. Note that not all implementations allow this option to be retrieved.
35214		
35215		
35216		
35217		
35218		
35219		
35220		
35221		
35222	SO_RCVTIMEO	Reports the timeout value for input operations. This option shall store a timeval structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was received. The default for this option is zero, which indicates that a receive operation shall not time out. Note that not all implementations allow this option to be retrieved.
35223		
35224		
35225		
35226		
35227		
35228		
35229		
35230	SO_SNDLOWAT	Reports the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option shall store an int value. Note that not all implementations allow this option to be retrieved.
35231		
35232		
35233		
35234		
35235	SO_SNDTIMEO	Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data was sent. The default for this option is zero, which indicates that a send operation shall not time out. The option shall store a timeval structure. Note that not all implementations allow this option to be retrieved.
35236		
35237		
35238		
35239		
35240		
35241		
35242		
35243		
		For Boolean options, a zero value indicates that the option is disabled and a non-zero value indicates that the option is enabled.

getsockopt()35244 **RETURN VALUE**

35245 Upon successful completion, *getsockopt()* shall return 0; otherwise, -1 shall be returned and *errno*
 35246 set to indicate the error.

35247 **ERRORS**

35248 The *getsockopt()* function shall fail if:

35249 [EBADF] The *socket* argument is not a valid file descriptor.

35250 [EINVAL] The specified option is invalid at the specified socket level.

35251 [ENOPROTOOPT]

35252 The option is not supported by the protocol.

35253 [ENOTSOCK] The *socket* argument does not refer to a socket.

35254 The *getsockopt()* function may fail if:

35255 [EACCES] The calling process does not have the appropriate privileges.

35256 [EINVAL] The socket has been shut down.

35257 [ENOBUFS] Insufficient resources are available in the system to complete the function.

35258 **EXAMPLES**

35259 None.

35260 **APPLICATION USAGE**

35261 None.

35262 **RATIONALE**

35263 None.

35264 **FUTURE DIRECTIONS**

35265 None.

35266 **SEE ALSO**

35267 *bind()*, *close()*, *endprotoent()*, *setsockopt()*, *socket()*

35268 XBD <[sys/socket.h](#)>, <[netinet/in.h](#)>

35269 **CHANGE HISTORY**

35270 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35271 The **restrict** keyword is added to the *getsockopt()* prototype for alignment with the
 35272 ISO/IEC 9899:1999 standard.

35273 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/47 is applied, updating the description of
 35274 SO_LINGER in the DESCRIPTION so that it blocks the calling thread rather than the process.

35275 **NAME**
 35276 `getsubopt` — parse suboption arguments from a string

35277 **SYNOPSIS**
 35278 `#include <stdlib.h>`

35279 `int getsubopt(char **optionp, char * const *keylistp, char **valuep);`

35280 DESCRIPTION

35281 The `getsubopt()` function shall parse suboption arguments in a flag argument. Such options often
 35282 result from the use of `getopt()`.

35283 The `getsubopt()` argument *optionp* is a pointer to a pointer to the option argument string. The
 35284 suboption arguments shall be separated by commas and each may consist of either a single
 35285 token, or a token-value pair separated by an equal sign.

35286 The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified
 35287 by a null pointer. Each entry in the vector is one of the possible tokens that might be found in
 35288 **optionp*. Since commas delimit suboption arguments in *optionp*, they should not appear in any of
 35289 the strings pointed to by *keylistp*. Similarly, because an equal sign separates a token from its
 35290 value, the application should not include an equal sign in any of the strings pointed to by
 35291 *keylistp*.

35292 The *valuep* argument is the address of a value string pointer.

35293 If a comma appears in *optionp*, it shall be interpreted as a suboption separator. After commas
 35294 have been processed, if there are one or more equal signs in a suboption string, the first equal
 35295 sign in any suboption string shall be interpreted as a separator between a token and a value.
 35296 Subsequent equal signs in a suboption string shall be interpreted as part of the value.

35297 If the string at **optionp* contains only one suboption argument (equivalently, no commas),
 35298 `getsubopt()` shall update **optionp* to point to the null character at the end of the string. Otherwise,
 35299 it shall isolate the suboption argument by replacing the comma separator with a null character,
 35300 and shall update **optionp* to point to the start of the next suboption argument. If the suboption
 35301 argument has an associated value (equivalently, contains an equal sign), `getsubopt()` shall update
 35302 **valuep* to point to the value's first character. Otherwise, it shall set **valuep* to a null pointer. The
 35303 calling application may use this information to determine whether the presence or absence of a
 35304 value for the suboption is an error.

35305 Additionally, when `getsubopt()` fails to match the suboption argument with a token in the *keylistp*
 35306 array, the calling application should decide if this is an error, or if the unrecognized option
 35307 should be processed in another way.

35308 RETURN VALUE

35309 The `getsubopt()` function shall return the index of the matched token string, or `-1` if no token
 35310 strings were matched.

35311 ERRORS

35312 No errors are defined.

35313

EXAMPLES

```

35314     #include <stdio.h>
35315     #include <stdlib.h>

35316     int do_all;
35317     const char *type;
35318     int read_size;
35319     int write_size;
35320     int read_only;

35321     enum
35322     {
35323         RO_OPTION = 0,
35324         RW_OPTION,
35325         READ_SIZE_OPTION,
35326         WRITE_SIZE_OPTION
35327     };

35328     const char *mount_opts[] =
35329     {
35330         [RO_OPTION] = "ro",
35331         [RW_OPTION] = "rw",
35332         [READ_SIZE_OPTION] = "rsize",
35333         [WRITE_SIZE_OPTION] = "wsize",
35334         NULL
35335     };

35336     int
35337     main(int argc, char *argv[])
35338     {
35339         char *subopts, *value;
35340         int opt;

35341         while ((opt = getopt(argc, argv, "at:o:")) != -1)
35342             switch(opt)
35343             {
35344                 case 'a':
35345                     do_all = 1;
35346                     break;
35347                 case 't':
35348                     type = optarg;
35349                     break;
35350                 case 'o':
35351                     subopts = optarg;
35352                     while (*subopts != '\0')
35353                         switch(getsubopt(&subopts, mount_opts, &value))
35354                         {
35355                             case RO_OPTION:
35356                                 read_only = 1;
35357                                 break;
35358                             case RW_OPTION:
35359                                 read_only = 0;
35360                                 break;
35361                             case READ_SIZE_OPTION:
35362                                 if (value == NULL)
35363                                     abort();

```

```

35364         read_size = atoi(value);
35365         break;
35366     case WRITE_SIZE_OPTION:
35367         if (value == NULL)
35368             abort();
35369         write_size = atoi(value);
35370         break;
35371     default:
35372         /* Unknown suboption. */
35373         printf("Unknown suboption '%s'\n", value);
35374         break;
35375     }
35376     break;
35377     default:
35378         abort();
35379     }
35380     /* Do the real work. */
35381     return 0;
35382 }

```

35383 Parsing Suboptions

35384 The following example uses the *getsubopt()* function to parse a *value* argument in the *optarg*
 35385 external variable returned by a call to *getopt()*.

```

35386 #include <stdlib.h>
35387 ...
35388 char *tokens[] = {"HOME", "PATH", "LOGNAME", (char *) NULL };
35389 char *value;
35390 int opt, index;
35391 while ((opt = getopt(argc, argv, "e:")) != -1) {
35392     switch(opt) {
35393     case 'e':
35394         while ((index = getsubopt(&optarg, tokens, &value)) != -1) {
35395             switch(index) {
35396             ...
35397             }
35398             break;
35399         ...
35400     }
35401     }
35402     ...

```

35403 APPLICATION USAGE

35404 None.

35405 RATIONALE

35406 None.

35407 FUTURE DIRECTIONS

35408 None.

35409

SEE ALSO

35410

getopt()

35411

XBD <stdlib.h>

35412

CHANGE HISTORY

35413

First released in Issue 4, Version 2.

35414

Issue 5

35415

Moved from X/OPEN UNIX extension to BASE.

35416

Issue 6

35417

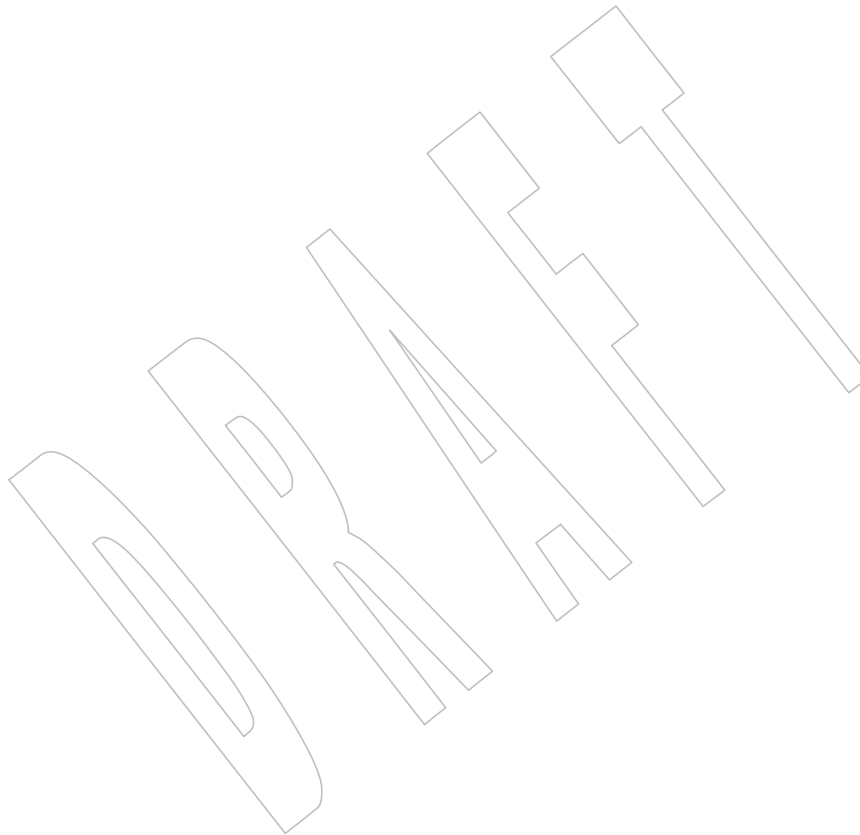
IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/26 is applied, correcting an editorial error in the SYNOPSIS.

35418

35419

Issue 7

35420

The *getsubopt()* function is moved from the XSI option to the Base.

35421 **NAME**
 35422 gettimeofday — get the date and time

35423 **SYNOPSIS**

```
35424 OB XSI #include <sys/time.h>
35425      int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

35426 **DESCRIPTION**

35427 The *gettimeofday()* function shall obtain the current time, expressed as seconds and microseconds
 35428 since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The resolution of the
 35429 system clock is unspecified.

35430 If *tzp* is not a null pointer, the behavior is unspecified.

35431 **RETURN VALUE**

35432 The *gettimeofday()* function shall return 0 and no value shall be reserved to indicate an error.

35433 **ERRORS**

35434 No errors are defined.

35435 **EXAMPLES**

35436 None.

35437 **APPLICATION USAGE**

35438 Applications should use the *clock_gettime()* function instead of the obsolescent *gettimeofday()*
 35439 function.

35440 **RATIONALE**

35441 None.

35442 **FUTURE DIRECTIONS**

35443 The *gettimeofday()* function may be removed in a future version.

35444 **SEE ALSO**

35445 *clock_getres()*, *ctime()*

35446 XBD [<sys/time.h>](#)

35447 **CHANGE HISTORY**

35448 First released in Issue 4, Version 2.

35449 **Issue 5**

35450 Moved from X/OPEN UNIX extension to BASE.

35451 **Issue 6**

35452 The DESCRIPTION is updated to refer to “seconds since the Epoch” rather than “seconds since
 35453 00:00:00 UTC (Coordinated Universal Time), January 1 1970” for consistency with other *time*
 35454 functions.

35455 The **restrict** keyword is added to the *gettimeofday()* prototype for alignment with the
 35456 ISO/IEC 9899:1999 standard.

35457 **Issue 7**

35458 The *gettimeofday()* function is marked obsolescent.

35459 **NAME**35460 `getuid` — get a real user ID35461 **SYNOPSIS**35462 `#include <unistd.h>`
35463 `uid_t getuid(void);`35464 **DESCRIPTION**35465 The `getuid()` function shall return the real user ID of the calling process.35466 **RETURN VALUE**35467 The `getuid()` function shall always be successful and no return value is reserved to indicate the
35468 error.35469 **ERRORS**

35470 No errors are defined.

35471 **EXAMPLES**35472 **Setting the Effective User ID to the Real User ID**35473 The following example sets the effective user ID and the real user ID of the current process to the
35474 real user ID of the caller.35475 `#include <unistd.h>`
35476 `#include <sys/types.h>`
35477 `...`
35478 `setreuid(getuid(), getuid());`
35479 `...`35480 **APPLICATION USAGE**

35481 None.

35482 **RATIONALE**

35483 None.

35484 **FUTURE DIRECTIONS**

35485 None.

35486 **SEE ALSO**35487 [*getegid\(\), geteuid\(\), getgid\(\), setegid\(\), seteuid\(\), setgid\(\), setregid\(\), setreuid\(\), setuid\(\)*](#)35488 XBD [*<sys/types.h>*](#), [*<unistd.h>*](#)35489 **CHANGE HISTORY**

35490 First released in Issue 1. Derived from Issue 1 of the SVID.

35491 **Issue 6**35492 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.35493 The following new requirements on POSIX implementations derive from alignment with the
35494 Single UNIX Specification:

- 35495
- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
35496 required for conforming implementations of previous POSIX specifications, it was not
35497 required for UNIX applications.

35498

NAME35499
35500 getutxent, getutxid, getutxline — get user accounting database entries**SYNOPSIS**35501 XSI

```
#include <utmpx.h>
```

```
35502 struct utmpx *getutxent(void);
```

```
35503 struct utmpx *getutxid(const struct utmpx *id);
```

```
35504 struct utmpx *getutxline(const struct utmpx *line);
```

```
35505
```

DESCRIPTION35506 Refer to *endutxent()*.

35507

35508 **NAME**35509 `getwc` — get a wide character from a stream35510 **SYNOPSIS**35511 `#include <stdio.h>`35512 `#include <wchar.h>`35513 `wint_t getwc(FILE *stream);`35514 **DESCRIPTION**35515 CX The functionality described on this reference page is aligned with the ISO C standard. Any
35516 conflict between the requirements described here and the ISO C standard is unintentional. This
35517 volume of POSIX.1-200x defers to the ISO C standard.35518 The `getwc()` function shall be equivalent to `fgetwc()`, except that if it is implemented as a macro it
35519 may evaluate `stream` more than once, so the argument should never be an expression with side
35520 effects.35521 **RETURN VALUE**35522 Refer to `fgetwc()`.35523 **ERRORS**35524 Refer to `fgetwc()`.35525 **EXAMPLES**

35526 None.

35527 **APPLICATION USAGE**35528 Since it may be implemented as a macro, `getwc()` may treat incorrectly a `stream` argument with
35529 side effects. In particular, `getwc(*f++)` does not necessarily work as expected. Therefore, use of
35530 this function is not recommended; `fgetwc()` should be used instead.35531 **RATIONALE**

35532 None.

35533 **FUTURE DIRECTIONS**

35534 None.

35535 **SEE ALSO**35536 `fgetwc()`35537 XBD `<stdio.h>`, `<wchar.h>`35538 **CHANGE HISTORY**35539 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
35540 draft.35541 **Issue 5**35542 The Optional Header (OH) marking is removed from `<stdio.h>`.

35543 **NAME**
 35544 `getwchar` — get a wide character from a stdin stream

35545 **SYNOPSIS**
 35546 `#include <wchar.h>`
 35547 `wint_t getwchar(void);`

35548 **DESCRIPTION**
 35549 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 35550 conflict between the requirements described here and the ISO C standard is unintentional. This
 35551 volume of POSIX.1-200x defers to the ISO C standard.

35552 The `getwchar()` function shall be equivalent to `getwc(stdin)`.

35553 **RETURN VALUE**
 35554 Refer to `fgetwc()`.

35555 **ERRORS**
 35556 Refer to `fgetwc()`.

35557 **EXAMPLES**
 35558 None.

35559 **APPLICATION USAGE**
 35560 If the `wint_t` value returned by `getwchar()` is stored into a variable of type `wchar_t` and then
 35561 compared against the `wint_t` macro `WEOF`, the result may be incorrect. Only the `wint_t` type is
 35562 guaranteed to be able to represent any wide character and `WEOF`.

35563 **RATIONALE**
 35564 None.

35565 **FUTURE DIRECTIONS**
 35566 None.

35567 **SEE ALSO**
 35568 `fgetwc()`, `getwc()`

35569 XBD `<wchar.h>`

35570 **CHANGE HISTORY**
 35571 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
 35572 draft.

35573 **NAME**
 35574 glob, globfree — generate pathnames matching a pattern

35575 **SYNOPSIS**
 35576 #include <glob.h>
 35577 int glob(const char *restrict pattern, int flags,
 35578 int (*errfunc)(const char *epath, int eerrno),
 35579 glob_t *restrict pglob);
 35580 void globfree(glob_t *pglob);

35581 **DESCRIPTION**
 35582 The *glob()* function is a pathname generator that shall implement the rules defined in XCU |
 35583 Section 2.13 (on page 2278), with optional support for rule 3 in XCU Section 2.13.3 (on page |
 35584 2279).

35585 The structure type **glob_t** is defined in <glob.h> and includes at least the following members:

Member Type	Member Name	Description
size_t	gl_pathc	Count of paths matched by <i>pattern</i> .
char **	gl_pathv	Pointer to a list of matched pathnames.
size_t	gl_offs	Slots to reserve at the beginning of <i>gl_pathv</i> .

35590 The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function
 35591 shall match all accessible pathnames against this pattern and develop a list of all pathnames that
 35592 match. In order to have access to a pathname, *glob()* requires search permission on every
 35593 component of a path except the last, and read permission on each directory of any filename
 35594 component of *pattern* that contains any of the following special characters: '*', '?', and '['.

35595 The *glob()* function shall store the number of matched pathnames into *pglob->gl_pathc* and a
 35596 pointer to a list of pointers to pathnames into *pglob->gl_pathv*. The pathnames shall be in sort
 35597 order as defined by the current setting of the *LC_COLLATE* category; see XBD Section 7.3.2 (on |
 35598 page 132). The first pointer after the last pathname shall be a null pointer. If the pattern does not |
 35599 match any pathnames, the returned number of matched paths is set to 0, and the contents of |
 35600 *pglob->gl_pathv* are implementation-defined.

35601 It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function
 35602 shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree()*
 35603 function shall free any space associated with *pglob* from a previous call to *glob()*.

35604 The *flags* argument is used to control the behavior of *glob()*. The value of *flags* is a bitwise-
 35605 inclusive OR of zero or more of the following constants, which are defined in <glob.h>:

35606	GLOBAL_APPEND	Append pathnames generated to the ones from a previous call to <i>glob()</i> .
35607	GLOBAL_DOOFFS	Make use of <i>pglob->gl_offs</i> . If this flag is set, <i>pglob->gl_offs</i> is used to 35608 specify how many null pointers to add to the beginning of 35609 <i>pglob->gl_pathv</i> . In other words, <i>pglob->gl_pathv</i> shall point to 35610 <i>pglob->gl_offs</i> null pointers, followed by <i>pglob->gl_pathc</i> pathname 35611 pointers, followed by a null pointer.
35612	GLOBAL_ERR	Cause <i>glob()</i> to return when it encounters a directory that it cannot open 35613 or read. Ordinarily, <i>glob()</i> continues to find matches.
35614	GLOBAL_MARK	Each pathname that is a directory that matches <i>pattern</i> shall have a slash 35615 appended.

35616 GLOB_NOCHECK Supports rule 3 in XCU [Section 2.13.3](#) (on page 2279). If *pattern* does not
 35617 match any pathname, then *glob()* shall return a list consisting of only
 35618 *pattern*, and the number of matched pathnames is 1.

35619 GLOB_NOESCAPE Disable backslash escaping.

35620 GLOB_NOSORT Ordinarily, *glob()* sorts the matching pathnames according to the current
 35621 setting of the *LC_COLLATE* category; see XBD [Section 7.3.2](#) (on page 132).
 35622 When this flag is used, the order of pathnames returned is unspecified.

35623 The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a
 35624 previous call to *glob()*. The following rules apply to applications when two or more calls to
 35625 *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- 35626 1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
- 35627 2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
- 35628 3. After the second call, *pglob->gl_pathv* points to a list containing the following:
 - 35629 a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob->gl_offs*.
 - 35630 b. Pointers to the pathnames that were in the *pglob->gl_pathv* list before the call, in
 35631 the same order as before.
 - 35632 c. Pointers to the new pathnames generated by the second call, in the specified order.
- 35633 4. The count returned in *pglob->gl_pathc* shall be the total number of pathnames from the
 35634 two calls.
- 35635 5. The application can change any of the fields after a call to *glob()*. If it does, the
 35636 application shall reset them to the original value before a subsequent call, using the same
 35637 *pglob* value, to *globfree()* or *glob()* with the GLOB_APPEND flag.

35638 If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not
 35639 a null pointer, *glob()* calls (**errfunc()*) with two arguments:

- 35640 1. The *epath* argument is a pointer to the path that failed.
- 35641 2. The *errno* argument is the value of *errno* from the failure, as set by *opendir()*, *readdir()*, or
 35642 *stat()*. (Other values may be used to report other errors not explicitly documented for
 35643 those functions.)

35644 If (**errfunc()*) is called and returns non-zero, or if the GLOB_ERR flag is set in *flags*, *glob()* shall
 35645 stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect
 35646 the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or
 35647 (**errfunc()*) returns 0, the error shall be ignored.

35648 The *glob()* function shall not fail because of large files.

35649 RETURN VALUE

35650 Upon successful completion, *glob()* shall return 0. The argument *pglob->gl_pathc* shall return the
 35651 number of matched pathnames and the argument *pglob->gl_pathv* shall contain a pointer to a
 35652 null-terminated list of matched and sorted pathnames. However, if *pglob->gl_pathc* is 0, the
 35653 content of *pglob->gl_pathv* is undefined.

35654 The *globfree()* function shall not return a value.

35655 If *glob()* terminates due to an error, it shall return one of the non-zero constants defined in
 35656 **<glob.h>**. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above.

ERRORS

35657
35658 The *glob()* function shall fail and return the corresponding value if:

35659 GLOB_ABORTED The scan was stopped because GLOB_ERR was set or (**errfunc()*)
35660 returned non-zero.

35661 GLOB_NOMATCH The pattern does not match any existing pathname, and
35662 GLOB_NOCHECK was not set in flags.

35663 GLOB_NOSPACE An attempt to allocate memory failed.

EXAMPLES

35664 One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with
35665 *execv()*, *execve()*, or *execvp()*. Suppose, for example, that an application wants to do the
35666 equivalent of:
35667

```
35668 ls -l *.c
```

35669 but for some reason:

```
35670 system("ls -l *.c")
```

35671 is not acceptable. The application could obtain approximately the same result using the
35672 sequence:

```
35673 globbuf.gl_offs = 2;  
35674 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);  
35675 globbuf.gl_pathv[0] = "ls";  
35676 globbuf.gl_pathv[1] = "-l";  
35677 execvp("ls", &globbuf.gl_pathv[0]);
```

35678 Using the same example:

```
35679 ls -l *.c *.h
```

35680 could be approximately simulated using GLOB_APPEND as follows:

```
35681 globbuf.gl_offs = 2;  
35682 glob("*.c", GLOB_DOOFFS, NULL, &globbuf);  
35683 glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);  
35684 ...
```

APPLICATION USAGE

35685 This function is not provided for the purpose of enabling utilities to perform pathname
35686 expansion on their arguments, as this operation is performed by the shell, and utilities are
35687 explicitly not expected to redo this. Instead, it is provided for applications that need to do
35688 pathname expansion on strings obtained from other sources, such as a pattern typed by a user or
35689 read from a file.
35690

35691 If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

35692 Note that *gl_pathc* and *gl_pathv* have meaning even if *glob()* fails. This allows *glob()* to report
35693 partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if
35694 *glob()* did not return an error.

35695 The GLOB_NOCHECK option could be used when an application wants to expand a pathname
35696 if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility
35697 might use this for option-arguments, for example.

35698 The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted
35699 together with the previous pathnames. This mirrors the way that the shell handles pathname
35700 expansion when multiple expansions are done on a command line.

35701 Applications that need tilde and parameter expansion should use *wordexp()*.

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
    * sizeof(char *));
(void) memcpy(new+n, pglob->gl_pathv,
    pglob->gl_pathc * sizeof(char *));
(void) memset(new, 0, n * sizeof(char *));
free(pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc()*. This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc()* in a large chunk and then carved up within *glob()*, or it might have been created using a different memory allocator. It is not the intent of the standard developers to specify or imply how the memory used by *glob()* is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fdopendir()*, *fnmatch()*, *fstatat()*, *readdir()*, Section 2.6

XBD Section 7.3.2 (on page 132), [<glob.h>](#)

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *glob()* prototype for alignment with the ISO/IEC 9899:1999 standard.

35732 **NAME**

35733 gmtime, gmtime_r — convert a time value to a broken-down UTC time

35734 **SYNOPSIS**

35735 #include <time.h>

35736 struct tm *gmtime(const time_t *timer);

35737 CX struct tm *gmtime_r(const time_t *restrict timer,
35738 struct tm *restrict result);35739 **DESCRIPTION**35740 CX For *gmtime()*: The functionality described on this reference page is aligned with the ISO C
35741 standard. Any conflict between the requirements described here and the ISO C standard is
35742 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.35743 The *gmtime()* function shall convert the time in seconds since the Epoch pointed to by *timer* into
35744 a broken-down time, expressed as Coordinated Universal Time (UTC).35745 CX The relationship between a time in seconds since the Epoch used as an argument to *gmtime()*
35746 and the **tm** structure (defined in the <time.h> header) is that the result shall be as specified in
35747 the expression given in the definition of seconds since the Epoch (see XBD [Section 4.15](#), on page
35748 100), where the names in the structure and in the expression correspond.35749 The same relationship shall apply for *gmtime_r()*.35750 The *gmtime()* function need not be thread-safe. A function that is not required to be thread-safe
35751 is not required to be reentrant.35752 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
35753 objects: a broken-down time structure and an array of type **char**. Execution of any of the
35754 functions may overwrite the information returned in either of these objects by any of the other
35755 functions.35756 The *gmtime_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*
35757 into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down
35758 time is stored in the structure referred to by *result*. The *gmtime_r()* function shall also return the
35759 address of the same structure.35760 **RETURN VALUE**35761 CX Upon successful completion, the *gmtime()* function shall return a pointer to a **struct tm**. If an
35762 error is detected, *gmtime()* shall return a null pointer and set *errno* to indicate the error.35763 Upon successful completion, *gmtime_r()* shall return the address of the structure pointed to by
35764 the argument *result*. If an error is detected, *gmtime_r()* shall return a null pointer and set *errno* to
35765 indicate the error.35766 **ERRORS**35767 CX The *gmtime()* and *gmtime_r()* functions shall fail if:

35768 CX [EOVERFLOW] The result cannot be represented.

35769
35770

35771
35772
35773

35774
35775

35776
35777

35778
35779
35780

35781
35782

35783
35784
35785
35786

35787
35788
35789
35790
35791

35792
35793

35794
35795

35796
35797

35798
35799

EXAMPLES

None.

APPLICATION USAGE

The *gmtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *time*, *utime()*

XBD Section 4.15 (on page 100), **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A note indicating that the *gmtime()* function need not be reentrant is added to the DESCRIPTION.

The *gmtime_r()* function is included for alignment with the POSIX Threads Extension.

Issue 6

The *gmtime_r()* function is marked as part of the Thread-Safe Functions option.

Extensions beyond the ISO C standard are marked.

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

The **restrict** keyword is added to the *gmtime_r()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/27 is applied, adding the [Eoverflow] error.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/48 is applied, updating the error handling for *gmtime_r()*.

Issue 7

The *gmtime_r()* function is moved from the Thread-Safe Functions option to the Base.

35800 **NAME**

35801 grantpt — grant access to the slave pseudo-terminal device

35802 **SYNOPSIS**

```
35803 XSI #include <stdlib.h>
35804 int grantpt(int fildes);
```

35805 **DESCRIPTION**

35806 The *grantpt()* function shall change the mode and ownership of the slave pseudo-terminal
 35807 device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file
 35808 descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to
 35809 the real UID of the calling process and the group ID shall be set to an unspecified group ID. The
 35810 permission mode of the slave pseudo-terminal shall be set to readable and writable by the
 35811 owner, and writable by the group.

35812 The behavior of the *grantpt()* function is unspecified if the application has installed a signal
 35813 handler to catch SIGCHLD signals.

35814 **RETURN VALUE**

35815 Upon successful completion, *grantpt()* shall return 0; otherwise, it shall return -1 and set *errno* to
 35816 indicate the error.

35817 **ERRORS**

35818 The *grantpt()* function may fail if:

- | | | |
|-------|----------|--|
| 35819 | [EBADF] | The <i>fildes</i> argument is not a valid open file descriptor. |
| 35820 | [EINVAL] | The <i>fildes</i> argument is not associated with a master pseudo-terminal device. |
| 35821 | [EACCES] | The corresponding slave pseudo-terminal device could not be accessed. |

35822 **EXAMPLES**

35823 None.

35824 **APPLICATION USAGE**

35825 None.

35826 **RATIONALE**

35827 None.

35828 **FUTURE DIRECTIONS**

35829 None.

35830 **SEE ALSO**

35831 *open()*, *ptsname()*, *unlockpt()*

35832 XBD [<stdlib.h>](#)

35833 **CHANGE HISTORY**

35834 First released in Issue 4, Version 2.

35835 **Issue 5**

35836 Moved from X/OPEN UNIX extension to BASE.

35837 The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

35838 **NAME**
 35839 `hcreate`, `hdestroy`, `hsearch` — manage hash search table

SYNOPSIS

```
35841 XSI #include <search.h>
35842
35842 int hcreate(size_t nel);
35843 void hdestroy(void);
35844 ENTRY *hsearch(ENTRY item, ACTION action);
```

DESCRIPTION

35845 The `hcreate()`, `hdestroy()`, and `hsearch()` functions shall manage hash search tables.

35847 The `hcreate()` function shall allocate sufficient space for the table, and the application shall
 35848 ensure it is called before `hsearch()` is used. The `nel` argument is an estimate of the maximum
 35849 number of entries that the table shall contain. This number may be adjusted upward by the
 35850 algorithm in order to obtain certain mathematically favorable circumstances.

35851 The `hdestroy()` function shall dispose of the search table, and may be followed by another call to
 35852 `hcreate()`. After the call to `hdestroy()`, the data can no longer be considered accessible.

35853 The `hsearch()` function is a hash-table search routine. It shall return a pointer into a hash table
 35854 indicating the location at which an entry can be found. The `item` argument is a structure of type
 35855 **ENTRY** (defined in the `<search.h>` header) containing two pointers: `item.key` points to the
 35856 comparison key (a `char *`), and `item.data` (a `void *`) points to any other data to be associated with
 35857 that key. The comparison function used by `hsearch()` is `strcmp()`. The `action` argument is a
 35858 member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be
 35859 found in the table. **ENTER** indicates that the item should be inserted in the table at an
 35860 appropriate point. **FIND** indicates that no entry should be made. Unsuccessful resolution is
 35861 indicated by the return of a null pointer.

35862 These functions need not be thread-safe. A function that is not required to be thread-safe is not
 35863 required to be reentrant.

RETURN VALUE

35864 The `hcreate()` function shall return 0 if it cannot allocate sufficient space for the table; otherwise,
 35865 it shall return non-zero.

35867 The `hdestroy()` function shall not return a value.

35868 The `hsearch()` function shall return a null pointer if either the action is **FIND** and the item could
 35869 not be found or the action is **ENTER** and the table is full.

ERRORS

35870 The `hcreate()` and `hsearch()` functions may fail if:
 35871
 35872 [ENOMEM] Insufficient storage space is available.

EXAMPLES

The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```

35873
35874
35875
35876
35877
35878
35879
35880
35881
35882
35883
35884
35885
35886
35887
35888
35889
35890
35891
35892
35893
35894
35895
35896
35897
35898
35899
35900
35901
35902
35903
35904
35905
35906
35907
35908
35909
35910
35911
35912
35913
35914
35915
35916
35917
35918
35919
35920
#include <stdio.h>
#include <search.h>
#include <string.h>

struct info {          /* This is the info stored in the table */
    int age, room;     /* other than the key. */
};

#define NUM_EMPL      5000    /* # of elements in search table. */

int main(void)
{
    char string_space[NUM_EMPL*20]; /* Space to store strings. */
    struct info info_space[NUM_EMPL]; /* Space to store employee info. */
    char *str_ptr = string_space; /* Next space in string_space. */
    struct info *info_ptr = info_space; /* Next space in info_space. */

    ENTRY item;
    ENTRY *found_item; /* Name to look for in table. */
    char name_to_find[30];

    int i = 0;

    /* Create table; no error checking is performed. */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
                &info_ptr->room) != EOF && i++ < NUM_EMPL) {

        /* Put information in structure, and structure in item. */
        item.key = str_ptr;
        item.data = info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;

        /* Put item into table. */
        (void) hsearch(item, ENTER);
    }

    /* Access table. */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {

            /* If item is in the table. */
            (void)printf("found %s, age = %d, room = %d\n",
                found_item->key,
                ((struct info *)found_item->data)->age,
                ((struct info *)found_item->data)->room);
        } else
            (void)printf("no such employee %s\n", name_to_find);
    }
    return 0;
}

```

35921 **APPLICATION USAGE**
35922 The *hcreate()* and *hsearch()* functions may use *malloc()* to allocate space.

35923 **RATIONALE**
35924 None.

35925 **FUTURE DIRECTIONS**
35926 None.

35927 **SEE ALSO**
35928 *bsearch()*, *lsearch()*, *malloc()*, *strcmp()*, *tdelete()*
35929 XBD <[search.h](#)>

35930 **CHANGE HISTORY**
35931 First released in Issue 1. Derived from Issue 1 of the SVID.

35932 **Issue 6**
35933 The normative text is updated to avoid use of the term “must” for application requirements.
35934 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

DRAFT

35935 **NAME**
 35936 htonl, htons, ntohl, ntohs — convert values between host and network byte order

35937 **SYNOPSIS**
 35938 #include <arpa/inet.h>
 35939 uint32_t htonl(uint32_t *hostlong*);
 35940 uint16_t htons(uint16_t *hostshort*);
 35941 uint32_t ntohl(uint32_t *netlong*);
 35942 uint16_t ntohs(uint16_t *netshort*);

35943 **DESCRIPTION**
 35944 These functions shall convert 16-bit and 32-bit quantities between network byte order and host
 35945 byte order.

35946 On some implementations, these functions are defined as macros.

35947 The **uint32_t** and **uint16_t** types are defined in **<inttypes.h>**.

35948 **RETURN VALUE**
 35949 The *htonl()* and *htons()* functions shall return the argument value converted from host to
 35950 network byte order.

35951 The *ntohl()* and *ntohs()* functions shall return the argument value converted from network to
 35952 host byte order.

35953 **ERRORS**
 35954 No errors are defined.

35955 **EXAMPLES**
 35956 None.

35957 **APPLICATION USAGE**
 35958 These functions are most often used in conjunction with IPv4 addresses and ports as returned by
 35959 *gethostent()* and *getservent()*.

35960 **RATIONALE**
 35961 None.

35962 **FUTURE DIRECTIONS**
 35963 None.

35964 **SEE ALSO**
 35965 *endhostent()*, *endservent()*
 35966 XBD **<arpa/inet.h>**, **<inttypes.h>**

35967 **CHANGE HISTORY**
 35968 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

35969 **NAME**
 35970 hypot, hypotf, hypotl — Euclidean distance function

35971 **SYNOPSIS**
 35972 #include <math.h>
 35973 double hypot(double x, double y);
 35974 float hypotf(float x, float y);
 35975 long double hypotl(long double x, long double y);

35976 DESCRIPTION

35977 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 35978 conflict between the requirements described here and the ISO C standard is unintentional. This
 35979 volume of POSIX.1-200x defers to the ISO C standard.

35980 These functions shall compute the value of the square root of x^2+y^2 without undue overflow or
 35981 underflow.

35982 An application wishing to check for error situations should set *errno* to zero and call
 35983 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 35984 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 35985 zero, an error has occurred.

35986 RETURN VALUE

35987 Upon successful completion, these functions shall return the length of the hypotenuse of a right-
 35988 angled triangle with sides of length *x* and *y*.

35989 If the correct value would cause overflow, a range error shall occur and *hypot()*, *hypotf()*, and
 35990 *hypotl()* shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL,
 35991 respectively.

35992 MX If *x* or *y* is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned (even if one of *x* or *y* is NaN).

35993 If *x* or *y* is NaN, and the other is not $\pm\text{Inf}$, a NaN shall be returned.

35994 If both arguments are subnormal and the correct result is subnormal, a range error may occur
 35995 and the correct result is returned.

35996 ERRORS

35997 These functions shall fail if:

35998 Range Error The result overflows.

35999 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 36000 then *errno* shall be set to [ERANGE]. If the integer expression
 36001 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 36002 floating-point exception shall be raised.

36003 These functions may fail if:

36004 MX Range Error The result underflows.

36005 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 36006 then *errno* shall be set to [ERANGE]. If the integer expression
 36007 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 36008 floating-point exception shall be raised.

36009
36010
36011
36012
36013
36014
36015
36016
36017
36018
36019
36020
36021
36022
36023
36024
36025
36026
36027
36028
36029
36030
36031
36032
36033
36034
36035
36036
36037
36038
36039
36040

EXAMPLES

See the EXAMPLES section in *atan2()*.

APPLICATION USAGE

hypot(x,y), *hypot(y,x)*, and *hypot(x, -y)* are equivalent.

hypot(x, ±0) is equivalent to *fabs(x)*.

Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

These functions take precautions against overflow during intermediate steps of the computation.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *sqrt()*

XBD Section 4.19 (on page 104), <math.h>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *hypot()* function is no longer marked as an extension.

The *hypotf()* and *hypotl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/49 is applied, updating the EXAMPLES section.

36041 **NAME**

36042 iconv — codeset conversion function

36043 **SYNOPSIS**

36044 #include <iconv.h>

```
36045 size_t iconv(iconv_t cd, char **restrict inbuf,
36046             size_t *restrict inbytesleft, char **restrict outbuf,
36047             size_t *restrict outbytesleft);
```

36048 **DESCRIPTION**

36049 The *iconv()* function shall convert the sequence of characters from one codeset, in the array
 36050 specified by *inbuf*, into a sequence of corresponding characters in another codeset, in the array
 36051 specified by *outbuf*. The codesets are those specified in the *iconv_open()* call that returned the
 36052 conversion descriptor, *cd*. The *inbuf* argument points to a variable that points to the first
 36053 character in the input buffer and *inbytesleft* indicates the number of bytes to the end of the buffer
 36054 to be converted. The *outbuf* argument points to a variable that points to the first available byte in
 36055 the output buffer and *outbytesleft* indicates the number of the available bytes to the end of the
 36056 buffer.

36057 For state-dependent encodings, the conversion descriptor *cd* is placed into its initial shift state by
 36058 a call for which *inbuf* is a null pointer, or for which *inbuf* points to a null pointer. When *iconv()* is
 36059 called in this way, and if *outbuf* is not a null pointer or a pointer to a null pointer, and *outbytesleft*
 36060 points to a positive value, *iconv()* shall place, into the output buffer, the byte sequence to change
 36061 the output buffer to its initial shift state. If the output buffer is not large enough to hold the
 36062 entire reset sequence, *iconv()* shall fail and set *errno* to [E2BIG]. Subsequent calls with *inbuf* as
 36063 other than a null pointer or a pointer to a null pointer cause the conversion to take place from
 36064 the current state of the conversion descriptor.

36065 If a sequence of input bytes does not form a valid character in the specified codeset, conversion
 36066 shall stop after the previous successfully converted character. If the input buffer ends with an
 36067 incomplete character or shift sequence, conversion shall stop after the previous successfully
 36068 converted bytes. If the output buffer is not large enough to hold the entire converted input,
 36069 conversion shall stop just prior to the input bytes that would cause the output buffer to
 36070 overflow. The variable pointed to by *inbuf* shall be updated to point to the byte following the last
 36071 byte successfully used in the conversion. The value pointed to by *inbytesleft* shall be
 36072 decremented to reflect the number of bytes still not converted in the input buffer. The variable
 36073 pointed to by *outbuf* shall be updated to point to the byte following the last byte of converted
 36074 output data. The value pointed to by *outbytesleft* shall be decremented to reflect the number of
 36075 bytes still available in the output buffer. For state-dependent encodings, the conversion
 36076 descriptor shall be updated to reflect the shift state in effect at the end of the last successfully
 36077 converted byte sequence.

36078 If *iconv()* encounters a character in the input buffer that is valid, but for which an identical
 36079 character does not exist in the target codeset, *iconv()* shall perform an implementation-defined
 36080 conversion on this character.

36081 **RETURN VALUE**

36082 The *iconv()* function shall update the variables pointed to by the arguments to reflect the extent
 36083 of the conversion and return the number of non-identical conversions performed. If the entire
 36084 string in the input buffer is converted, the value pointed to by *inbytesleft* shall be 0. If the input
 36085 conversion is stopped due to any conditions mentioned above, the value pointed to by *inbytesleft*
 36086 shall be non-zero and *errno* shall be set to indicate the condition. If an error occurs, *iconv()* shall
 36087 return (*size_t*)-1 and set *errno* to indicate the error.

ERRORS

- 36088 The *iconv()* function shall fail if:
- 36089
- 36090 [EILSEQ] Input conversion stopped due to an input byte that does not belong to the
- 36091 input codeset.
- 36092 [E2BIG] Input conversion stopped due to lack of space in the output buffer.
- 36093 [EINVAL] Input conversion stopped due to an incomplete character or shift sequence at
- 36094 the end of the input buffer.
- 36095 The *iconv()* function may fail if:
- 36096 [EBADF] The *cd* argument is not a valid open conversion descriptor.

EXAMPLES

- 36097 None.
- 36098

APPLICATION USAGE

- 36100 The *inbuf* argument indirectly points to the memory area which contains the conversion input
- 36101 data. The *outbuf* argument indirectly points to the memory area which is to contain the result of
- 36102 the conversion. The objects indirectly pointed to by *inbuf* and *outbuf* are not restricted to
- 36103 containing data that is directly representable in the ISO C standard language **char** data type. The
- 36104 type of *inbuf* and *outbuf*, **char ****, does not imply that the objects pointed to are interpreted as
- 36105 null-terminated C strings or arrays of characters. Any interpretation of a byte sequence that
- 36106 represents a character in a given character set encoding scheme is done internally within the
- 36107 codeset converters. For example, the area pointed to indirectly by *inbuf* and/or *outbuf* can
- 36108 contain all zero octets that are not interpreted as string terminators but as coded character data
- 36109 according to the respective codeset encoding scheme. The type of the data (**char**, **short**, **long**, and
- 36110 so on) read or stored in the objects is not specified, but may be inferred for both the input and
- 36111 output data by the converters determined by the *fromcode* and *to code* arguments of *iconv_open()*.
- 36112 Regardless of the data type inferred by the converter, the size of the remaining space in both
- 36113 input and output objects (the *inbytesleft* and *outbytesleft* arguments) is always measured in bytes.
- 36114 For implementations that support the conversion of state-dependent encodings, the conversion
- 36115 descriptor must be able to accurately reflect the shift-state in effect at the end of the last
- 36116 successful conversion. It is not required that the conversion descriptor itself be updated, which
- 36117 would require it to be a pointer type. Thus, implementations are free to implement the
- 36118 descriptor as a handle (other than a pointer type) by which the conversion information can be
- 36119 accessed and updated.

RATIONALE

- 36120 None.
- 36121

FUTURE DIRECTIONS

- 36122 None.
- 36123

SEE ALSO

- 36124 *iconv_open()*, *iconv_close()*, *mbsrtowcs()*
- 36125
- 36126 XBD [<iconv.h>](#)

CHANGE HISTORY

- 36127 First released in Issue 4. Derived from the HP-UX Manual.
- 36128

Issue 6

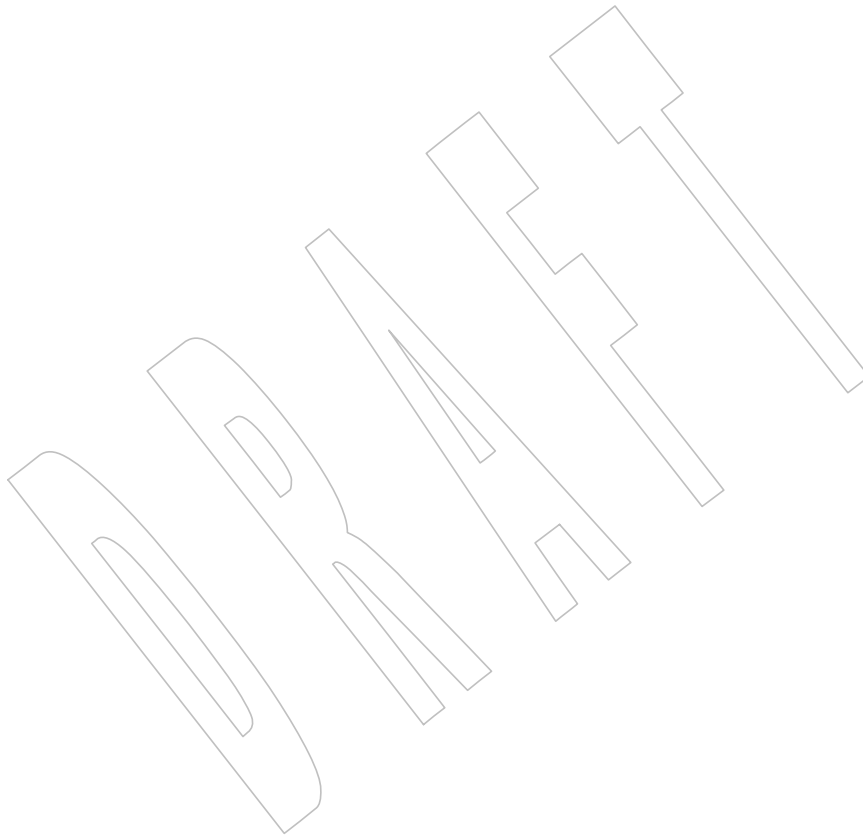
- 36129 The SYNOPSIS has been corrected to align with the [<iconv.h>](#) reference page.
- 36130
- 36131 The **restrict** keyword is added to the *iconv()* prototype for alignment with the
- 36132 ISO/IEC 9899:1999 standard.

36133

Issue 7

36134

The *iconv()* function is moved from the XSI option to the Base.



36135 **NAME**
 36136 `iconv_close` — codeset conversion deallocation function

36137 **SYNOPSIS**
 36138 `#include <iconv.h>`
 36139 `int iconv_close(iconv_t cd);`

36140 **DESCRIPTION**
 36141 The `iconv_close()` function shall deallocate the conversion descriptor `cd` and all other associated
 36142 resources allocated by `iconv_open()`.

36143 If a file descriptor is used to implement the type `iconv_t`, that file descriptor shall be closed.

36144 **RETURN VALUE**
 36145 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and `errno` set to
 36146 indicate the error.

36147 **ERRORS**
 36148 The `iconv_close()` function may fail if:
 36149 [EBADF] The conversion descriptor is invalid.

36150 **EXAMPLES**
 36151 None.

36152 **APPLICATION USAGE**
 36153 None.

36154 **RATIONALE**
 36155 None.

36156 **FUTURE DIRECTIONS**
 36157 None.

36158 **SEE ALSO**
 36159 [iconv](#), [iconv_open\(\)](#)
 36160 XBD [<iconv.h>](#)

36161 **CHANGE HISTORY**
 36162 First released in Issue 4. Derived from the HP-UX Manual.

36163 **Issue 7**
 36164 The `iconv_close()` function is moved from the XSI option to the Base.

36165 **NAME**
 36166 iconv_open — codeset conversion allocation function

36167 **SYNOPSIS**
 36168 #include <iconv.h>

36169 iconv_t iconv_open(const char *tocode, const char *fromcode);

36170 DESCRIPTION

36171 The *iconv_open()* function shall return a conversion descriptor that describes a conversion from
 36172 the codeset specified by the string pointed to by the *fromcode* argument to the codeset specified
 36173 by the string pointed to by the *tocode* argument. For state-dependent encodings, the conversion
 36174 descriptor shall be in a codeset-dependent initial shift state, ready for immediate use with
 36175 *iconv()*.

36176 Settings of *fromcode* and *tocode* and their permitted combinations are implementation-defined.

36177 A conversion descriptor shall remain valid until it is closed by *iconv_close()* or an implicit close.

36178 If a file descriptor is used to implement conversion descriptors, the FD_CLOEXEC flag shall be
 36179 set; see <fcntl.h>.

36180 RETURN VALUE

36181 Upon successful completion, *iconv_open()* shall return a conversion descriptor for use on
 36182 subsequent calls to *iconv()*. Otherwise, *iconv_open()* shall return (*iconv_t*)-1 and set *errno* to
 36183 indicate the error.

36184 ERRORS

36185 The *iconv_open()* function may fail if:

36186 [EMFILE] All file descriptors available to the process are currently open.

36187 [ENFILE] Too many files are currently open in the system.

36188 [ENOMEM] Insufficient storage space is available.

36189 [EINVAL] The conversion specified by *fromcode* and *tocode* is not supported by the
 36190 implementation.

36191 EXAMPLES

36192 None.

36193 APPLICATION USAGE

36194 Some implementations of *iconv_open()* use *malloc()* to allocate space for internal buffer areas.
 36195 The *iconv_open()* function may fail if there is insufficient storage space to accommodate these
 36196 buffers.

36197 Conforming applications must assume that conversion descriptors are not valid after a call to
 36198 one of the *exec* functions.

36199 Application developers should consult the system documentation to determine the supported
 36200 codesets and their naming schemes.

36201 RATIONALE

36202 None.

36203 FUTURE DIRECTIONS

36204 None.

iconv_open()

36205

SEE ALSO

36206

iconv, *iconv_close()*

36207

XBD <*fcntl.h*>, <*iconv.h*>

36208

CHANGE HISTORY

36209

First released in Issue 4. Derived from the HP-UX Manual.

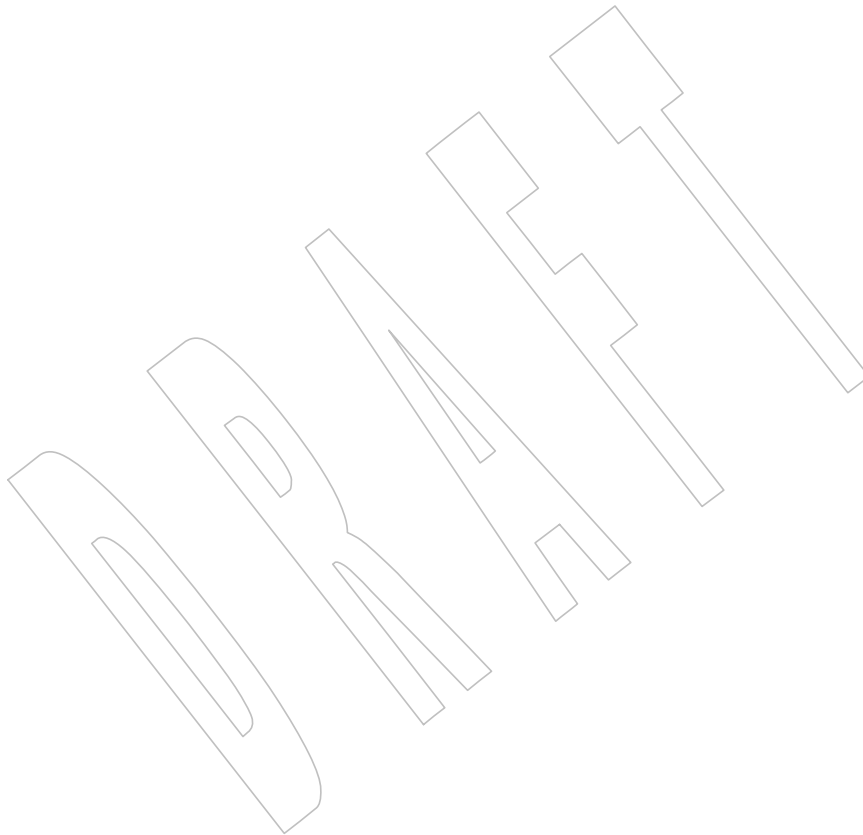
36210

Issue 7

36211

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

36212

The *iconv_open()* function is moved from the XSI option to the Base.

36213 NAME

36214 `if_freenameindex` — free memory allocated by `if_nameindex`

36215 SYNOPSIS

36216 `#include <net/if.h>`

36217 `void if_freenameindex(struct if_nameindex *ptr);`

36218 DESCRIPTION

36219 The `if_freenameindex()` function shall free the memory allocated by `if_nameindex()`. The `ptr`
 36220 argument shall be a pointer that was returned by `if_nameindex()`. After `if_freenameindex()` has
 36221 been called, the application shall not use the array of which `ptr` is the address.

36222 RETURN VALUE

36223 None.

36224 ERRORS

36225 No errors are defined.

36226 EXAMPLES

36227 None.

36228 APPLICATION USAGE

36229 None.

36230 RATIONALE

36231 None.

36232 FUTURE DIRECTIONS

36233 None.

36234 SEE ALSO

36235 [*getsockopt\(\), if_indextoname\(\), if_nameindex\(\), if_nametoindex\(\), setsockopt\(\)*](#)

36236 XBD [*<net/if.h>*](#)

36237 CHANGE HISTORY

36238 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36239 NAME

36240 `if_indexname` — map a network interface index to its corresponding name

36241 SYNOPSIS

36242 `#include <net/if.h>`

36243 `char *if_indexname(unsigned ifindex, char *ifname);`

36244 DESCRIPTION

36245 The `if_indexname()` function shall map an interface index to its corresponding name.

36246 When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The
36247 function shall place in this buffer the name of the interface with index *ifindex*.

36248 RETURN VALUE

36249 If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which
36250 points to a buffer now containing the interface name. Otherwise, the function shall return a
36251 NULL pointer and set *errno* to indicate the error.

36252 ERRORS

36253 The `if_indexname()` function shall fail if:

36254 [ENXIO] The interface does not exist.

36255 EXAMPLES

36256 None.

36257 APPLICATION USAGE

36258 None.

36259 RATIONALE

36260 None.

36261 FUTURE DIRECTIONS

36262 None.

36263 SEE ALSO

36264 [getsockopt\(\)](#), [if_freenameindex\(\)](#), [if_nameindex\(\)](#), [if_nametoindex\(\)](#), [setsockopt\(\)](#)

36265 XBD [<net/if.h>](#)

36266 CHANGE HISTORY

36267 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36268 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/28 is applied, changing {IFNAMSIZ} to
36269 {IF_NAMESIZ} in the DESCRIPTION.

36270 **NAME**

36271 if_nameindex — return all network interface names and indexes

36272 **SYNOPSIS**

36273 #include <net/if.h>

36274 struct if_nameindex *if_nameindex(void);

36275 **DESCRIPTION**36276 The *if_nameindex()* function shall return an array of *if_nameindex* structures, one structure per
36277 interface. The end of the array is indicated by a structure with an *if_index* field of zero and an
36278 *if_name* field of NULL.36279 Applications should call *if_freenameindex()* to release the memory that may be dynamically
36280 allocated by this function, after they have finished using it.36281 **RETURN VALUE**36282 An array of structures identifying local interfaces. A NULL pointer is returned upon an error,
36283 with *errno* set to indicate the error.36284 **ERRORS**36285 The *if_nameindex()* function may fail if:

36286 [ENOBUFS] Insufficient resources are available to complete the function.

36287 **EXAMPLES**

36288 None.

36289 **APPLICATION USAGE**

36290 None.

36291 **RATIONALE**

36292 None.

36293 **FUTURE DIRECTIONS**

36294 None.

36295 **SEE ALSO**36296 *getsockopt()*, *if_freenameindex()*, *if_indextoname()*, *if_nametoindex()*, *setsockopt()*

36297 XBD <net/if.h>

36298 **CHANGE HISTORY**

36299 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

if_nametoindex()36300 **NAME**36301 `if_nametoindex` — map a network interface name to its corresponding index36302 **SYNOPSIS**36303 `#include <net/if.h>`36304 `unsigned if_nametoindex(const char *ifname);`36305 **DESCRIPTION**36306 The `if_nametoindex()` function shall return the interface index corresponding to name *ifname*.36307 **RETURN VALUE**36308 The corresponding index if *ifname* is the name of an interface; otherwise, zero.36309 **ERRORS**

36310 No errors are defined.

36311 **EXAMPLES**

36312 None.

36313 **APPLICATION USAGE**

36314 None.

36315 **RATIONALE**

36316 None.

36317 **FUTURE DIRECTIONS**

36318 None.

36319 **SEE ALSO**36320 [*getsockopt\(\)*](#), [*if_freenameindex\(\)*](#), [*if_indextoname\(\)*](#), [*if_nameindex\(\)*](#), [*setsockopt\(\)*](#)36321 XBD [**<net/if.h>**](#)36322 **CHANGE HISTORY**

36323 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36324 **NAME**
 36325 `ilogb, ilogbf, ilogbl` — return an unbiased exponent

36326 **SYNOPSIS**
 36327 `#include <math.h>`
 36328 `int ilogb(double x);`
 36329 `int ilogbf(float x);`
 36330 `int ilogbl(long double x);`

36331 DESCRIPTION

36332 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36333 conflict between the requirements described here and the ISO C standard is unintentional. This
 36334 volume of POSIX.1-200x defers to the ISO C standard.

36335 These functions shall return the exponent part of their argument x . Formally, the return value is
 36336 the integral part of $\log_r |x|$ as a signed integral value, for non-zero x , where r is the radix of the
 36337 machine's floating-point arithmetic, which is the value of `FLT_RADIX` defined in `<float.h>`.

36338 An application wishing to check for error situations should set `errno` to zero and call
 36339 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 36340 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 36341 zero, an error has occurred.

36342 RETURN VALUE

36343 Upon successful completion, these functions shall return the exponent part of x as a signed
 36344 integer value. They are equivalent to calling the corresponding `logb()` function and casting the
 36345 returned value to type `int`.

36346 XSI If x is 0, the value `FP_ILOGB0` shall be returned. On XSI-conformant systems, a domain error
 36347 shall occur;

36348 CX otherwise, a domain error may occur.

36349 XSI If x is $\pm\text{Inf}$, the value `{INT_MAX}` shall be returned. On XSI-conformant systems, a domain
 36350 error shall occur;

36351 CX otherwise, a domain error may occur.

36352 XSI If x is a NaN, the value `FP_ILOGBNAN` shall be returned. On XSI-conformant systems, a
 36353 domain error shall occur;

36354 CX otherwise, a domain error may occur.

36355 MX If the correct value is greater than `{INT_MAX}`, a domain error shall occur and an unspecified
 36356 XSI value shall be returned. On XSI-conformant systems, a domain error shall occur and
 36357 `{INT_MAX}` shall be returned.

36358 MX If the correct value is less than `{INT_MIN}`, a domain error shall occur and an unspecified value
 36359 XSI shall be returned. On XSI-conformant systems, a domain error shall occur and `{INT_MIN}` shall
 36360 be returned.

36361 ERRORS

36362 These functions shall fail if:

36363 XSI|MX **Domain Error** The correct value is not representable as an integer.

36364 XSI The x argument is zero, NaN, or $\pm\text{Inf}$.

36365 XSI|MX If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 36366 then `errno` shall be set to `[EDOM]`. If the integer expression `(math_errhandling
 36367 & MATH_ERREXCEPT)` is non-zero, then the invalid floating-point exception

36368 shall be raised.

36369 These functions may fail if:

36370 Domain Error The x argument is zero, NaN, or $\pm\text{Inf}$.

36371 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 36372 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling* &
 36373 MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 36374 shall be raised.

36375 EXAMPLES

36376 None.

36377 APPLICATION USAGE

36378 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 36379 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

36380 RATIONALE

36381 The errors come from taking the expected floating-point value and converting it to **int**, which is
 36382 an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not
 36383 representable in a type **int**), so should be a domain error.

36384 There are no known implementations that overflow. For overflow to happen, {INT_MAX} must
 36385 be less than $\text{LDBL_MAX_EXP} \cdot \log_2(\text{FLT_RADIX})$ or {INT_MIN} must be greater than
 36386 $\text{LDBL_MIN_EXP} \cdot \log_2(\text{FLT_RADIX})$ if subnormals are not supported, or {INT_MIN} must be
 36387 greater than $(\text{LDBL_MIN_EXP} - \text{LDBL_MANT_DIG}) \cdot \log_2(\text{FLT_RADIX})$ if subnormals are
 36388 supported.

36389 FUTURE DIRECTIONS

36390 None.

36391 SEE ALSO

36392 *feclearexcept()*, *fetestexcept()*, *logb()*, *scalbln()*

36393 XBD Section 4.19 (on page 104), **<float.h>**, **<math.h>**

36394 CHANGE HISTORY

36395 First released in Issue 4, Version 2.

36396 Issue 5

36397 Moved from X/OPEN UNIX extension to BASE.

36398 Issue 6

36399 The *ilogb()* function is no longer marked as an extension.

36400 The *ilogbf()* and *ilogbl()* functions are added for alignment with the ISO/IEC 9899:1999
 36401 standard.

36402 The RETURN VALUE section is revised for alignment with the ISO/IEC 9899:1999 standard.

36403 Functionality relating to the XSI option is marked.

36404 Issue 7

36405 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #48 (SD5-XSH-ERN-71), #49, and #79
 36406 (SD5-XSH-ERN-72) are applied.

36407 **NAME**
 36408 `imaxabs` — return absolute value

36409 **SYNOPSIS**
 36410 `#include <inttypes.h>`
 36411 `intmax_t imaxabs(intmax_t j);`

36412 **DESCRIPTION**
 36413 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36414 conflict between the requirements described here and the ISO C standard is unintentional. This
 36415 volume of POSIX.1-200x defers to the ISO C standard.

36416 The `imaxabs()` function shall compute the absolute value of an integer *j*. If the result cannot be
 36417 represented, the behavior is undefined.

36418 **RETURN VALUE**
 36419 The `imaxabs()` function shall return the absolute value.

36420 **ERRORS**
 36421 No errors are defined.

36422 **EXAMPLES**
 36423 None.

36424 **APPLICATION USAGE**
 36425 The absolute value of the most negative number cannot be represented in two's complement.

36426 **RATIONALE**
 36427 None.

36428 **FUTURE DIRECTIONS**
 36429 None.

36430 **SEE ALSO**
 36431 [*imaxdiv\(\)*](#)
 36432 XBD [`<inttypes.h>`](#)

36433 **CHANGE HISTORY**
 36434 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

36435 **NAME**
 36436 `imaxdiv` — return quotient and remainder

36437 **SYNOPSIS**
 36438 `#include <inttypes.h>`
 36439 `imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`

36440 **DESCRIPTION**
 36441 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 36442 conflict between the requirements described here and the ISO C standard is unintentional. This
 36443 volume of POSIX.1-200x defers to the ISO C standard.

36444 The *imaxdiv()* function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.

36445 **RETURN VALUE**
 36446 The *imaxdiv()* function shall return a structure of type **imaxdiv_t**, comprising both the quotient
 36447 and the remainder. The structure shall contain (in either order) the members *quot* (the quotient)
 36448 and *rem* (the remainder), each of which has type **intmax_t**.

36449 If either part of the result cannot be represented, the behavior is undefined.

36450 **ERRORS**
 36451 No errors are defined.

36452 **EXAMPLES**
 36453 None.

36454 **APPLICATION USAGE**
 36455 None.

36456 **RATIONALE**
 36457 None.

36458 **FUTURE DIRECTIONS**
 36459 None.

36460 **SEE ALSO**
 36461 [imaxabs\(\)](#)
 36462 XBD [<inttypes.h>](#)

36463 **CHANGE HISTORY**
 36464 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

36465 **NAME**
 36466 `inet_addr, inet_ntoa` — IPv4 address manipulation

36467 **SYNOPSIS**
 36468 `#include <arpa/inet.h>`
 36469 `in_addr_t inet_addr(const char *cp);`
 36470 `char *inet_ntoa(struct in_addr in);`

36471 **DESCRIPTION**
 36472 The `inet_addr()` function shall convert the string pointed to by `cp`, in the standard IPv4 dotted
 36473 decimal notation, to an integer value suitable for use as an Internet address.

36474 The `inet_ntoa()` function shall convert the Internet host address specified by `in` to a string in the
 36475 Internet standard dot notation.

36476 The `inet_ntoa()` function need not be thread-safe. A function that is not required to be thread-safe
 36477 is not required to be reentrant.

36478 All Internet addresses shall be returned in network order (bytes ordered from left to right).

36479 Values specified using IPv4 dotted decimal notation take one of the following forms:

36480 `a.b.c.d` When four parts are specified, each shall be interpreted as a byte of data and
 36481 assigned, from left to right, to the four bytes of an Internet address.

36482 `a.b.c` When a three-part address is specified, the last part shall be interpreted as a 16-bit
 36483 quantity and placed in the rightmost two bytes of the network address. This makes
 36484 the three-part address format convenient for specifying Class B network addresses
 36485 as "`128.net.host`".

36486 `a.b` When a two-part address is supplied, the last part shall be interpreted as a 24-bit
 36487 quantity and placed in the rightmost three bytes of the network address. This
 36488 makes the two-part address format convenient for specifying Class A network
 36489 addresses as "`net.host`".

36490 `a` When only one part is given, the value shall be stored directly in the network
 36491 address without any byte rearrangement.

36492 All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or
 36493 hexadecimal, as specified in the ISO C standard (that is, a leading `0x` or `0X` implies hexadecimal;
 36494 otherwise, a leading `'0'` implies octal; otherwise, the number is interpreted as decimal).

36495 **RETURN VALUE**
 36496 Upon successful completion, `inet_addr()` shall return the Internet address. Otherwise, it shall
 36497 return `(in_addr_t)(-1)`.

36498 The `inet_ntoa()` function shall return a pointer to the network address in Internet standard dot
 36499 notation.

36500 **ERRORS**
 36501 No errors are defined.

inet_addr()

36502

EXAMPLES

36503

None.

36504

APPLICATION USAGE

36505

The return value of *inet_ntoa()* may point to static data that may be overwritten by subsequent calls to *inet_ntoa()*.

36506

36507

RATIONALE

36508

None.

36509

FUTURE DIRECTIONS

36510

None.

36511

SEE ALSO

36512

endhostent(), *endnetent()*

36513

XBD <arpa/inet.h>

36514

CHANGE HISTORY

36515

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

DRAFT

36516 **NAME**36517 `inet_ntop, inet_pton` — convert IPv4 and IPv6 addresses between binary and text form36518 **SYNOPSIS**36519

```
#include <arpa/inet.h>
```

36520

```
const char *inet_ntop(int af, const void *restrict src,  
char *restrict dst, socklen_t size);
```

36522

```
int inet_pton(int af, const char *restrict src, void *restrict dst);
```

36523 **DESCRIPTION**

36524 The `inet_ntop()` function shall convert a numeric address into a text string suitable for
 36525 IP6 presentation. The `af` argument shall specify the family of the address. This can be `AF_INET` or
 36526 `AF_INET6`. The `src` argument points to a buffer holding an IPv4 address if the `af` argument is
 36527 IP6 `AF_INET`, or an IPv6 address if the `af` argument is `AF_INET6`; the address must be in network
 36528 byte order. The `dst` argument points to a buffer where the function stores the resulting text string;
 36529 it shall not be NULL. The `size` argument specifies the size of this buffer, which shall be large
 36530 IP6 enough to hold the text string (`INET_ADDRSTRLEN` characters for IPv4,
 36531 `INET6_ADDRSTRLEN` characters for IPv6).

36532 The `inet_pton()` function shall convert an address in its standard text presentation form into its
 36533 IP6 numeric binary form. The `af` argument shall specify the family of the address. The `AF_INET` and
 36534 `AF_INET6` address families shall be supported. The `src` argument points to the string being
 36535 passed in. The `dst` argument points to a buffer into which the function stores the numeric
 36536 IP6 address; this shall be large enough to hold the numeric address (32 bits for `AF_INET`, 128 bits
 36537 for `AF_INET6`).

36538 If the `af` argument of `inet_pton()` is `AF_INET`, the `src` string shall be in the standard IPv4 dotted-
 36539 decimal form:

36540 `ddd . ddd . ddd . ddd`

36541 where "ddd" is a one to three digit decimal number between 0 and 255 (see `inet_addr()`). The
 36542 `inet_pton()` function does not accept other formats (such as the octal numbers, hexadecimal
 36543 numbers, and fewer than four numbers that `inet_addr()` accepts).

36544 IP6 If the `af` argument of `inet_pton()` is `AF_INET6`, the `src` string shall be in one of the following
 36545 standard IPv6 text forms:

- 36546 1. The preferred form is "`x:x:x:x:x:x:x:x`", where the 'x's are the hexadecimal values
 36547 of the eight 16-bit pieces of the address. Leading zeros in individual fields can be
 36548 omitted, but there shall be at least one numeral in every field.
- 36549 2. A string of contiguous zero fields in the preferred form can be shown as "`:::`". The "`:::`"
 36550 can only appear once in an address. Unspecified addresses ("`0:0:0:0:0:0:0:0`") may
 36551 be represented simply as "`:::`".
- 36552 3. A third form that is sometimes more convenient when dealing with a mixed environment
 36553 of IPv4 and IPv6 nodes is "`x:x:x:x:x:x.d.d.d.d`", where the 'x's are the
 36554 hexadecimal values of the six high-order 16-bit pieces of the address, and the 'd's are the
 36555 decimal values of the four low-order 8-bit pieces of the address (standard IPv4
 36556 representation).

36557 **Note:** A more extensive description of the standard representations of IPv6 addresses can be found in
 36558 RFC 2373.

inet_ntop()36559 **RETURN VALUE**

36560 The *inet_ntop()* function shall return a pointer to the buffer containing the text string if the
 36561 conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

36562 The *inet_pton()* function shall return 1 if the conversion succeeds, with the address pointed to by
 36563 IP6 *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string
 36564 or a valid IPv6 address string, or -1 with *errno* set to [EAFNOSUPPORT] if the *af* argument is
 36565 unknown.

36566 **ERRORS**

36567 The *inet_ntop()* and *inet_pton()* functions shall fail if:

36568 [EAFNOSUPPORT]

36569 The *af* argument is invalid.

36570 [ENOSPC] The size of the *inet_ntop()* result buffer is inadequate.

36571 **EXAMPLES**

36572 None.

36573 **APPLICATION USAGE**

36574 None.

36575 **RATIONALE**

36576 None.

36577 **FUTURE DIRECTIONS**

36578 None.

36579 **SEE ALSO**

36580 XBD <[arpa/inet.h](#)>

36581 **CHANGE HISTORY**

36582 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

36583 IPv6 extensions are marked.

36584 The **restrict** keyword is added to the *inet_ntop()* and *inet_pton()* prototypes for alignment with
 36585 the ISO/IEC 9899:1999 standard.

36586 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/29 is applied, adding “the address must
 36587 be in network byte order” to the end of the fourth sentence of the first paragraph in the
 36588 DESCRIPTION.

36589 **NAME**36590 `initstate`, `random`, `setstate`, `srandom` — pseudo-random number functions36591 **SYNOPSIS**

```

36592 XSI      #include <stdlib.h>
36593
36593 char *initstate(unsigned seed, char *state, size_t size);
36594 long random(void);
36595 char *setstate(const char *state);
36596 void srandom(unsigned seed);

```

36597 **DESCRIPTION**

36598 The `random()` function shall use a non-linear additive feedback random-number generator
 36599 employing a default state array size of 31 **long** integers to return successive pseudo-random
 36600 numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is
 36601 approximately $16 \times (2^{31}-1)$. The size of the state array determines the period of the random-
 36602 number generator. Increasing the state array size shall increase the period.

36603 With 256 bytes of state information, the period of the random-number generator shall be greater
 36604 than 2^{69} .

36605 Like `rand()`, `random()` shall produce by default a sequence of numbers that can be duplicated by
 36606 calling `srandom()` with 1 as the seed.

36607 The `srandom()` function shall initialize the current state array using the value of `seed`.

36608 The `initstate()` and `setstate()` functions handle restarting and changing random-number
 36609 generators. The `initstate()` function allows a state array, pointed to by the `state` argument, to be
 36610 initialized for future use. The `size` argument, which specifies the size in bytes of the state array,
 36611 shall be used by `initstate()` to decide what type of random-number generator to use; the larger
 36612 the state array, the more random the numbers. Values for the amount of state information are 8,
 36613 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one
 36614 of these values. If `initstate()` is called with $8 \leq \text{size} < 32$, then `random()` shall use a simple linear
 36615 congruential random number generator. The `seed` argument specifies a starting point for the
 36616 random-number sequence and provides for restarting at the same point. The `initstate()` function
 36617 shall return a pointer to the previous state information array.

36618 If `initstate()` has not been called, then `random()` shall behave as though `initstate()` had been called
 36619 with `seed=1` and `size=128`.

36620 Once a state has been initialized, `setstate()` allows switching between state arrays. The array
 36621 defined by the `state` argument shall be used for further random-number generation until
 36622 `initstate()` is called or `setstate()` is called again. The `setstate()` function shall return a pointer to the
 36623 previous state array.

36624 **RETURN VALUE**

36625 If `initstate()` is called with `size` less than 8, it shall return NULL.

36626 The `random()` function shall return the generated pseudo-random number.

36627 The `srandom()` function shall not return a value.

36628 Upon successful completion, `initstate()` and `setstate()` shall return a pointer to the previous state
 36629 array; otherwise, a null pointer shall be returned.

36630 ERRORS

36631 No errors are defined.

36632 EXAMPLES

36633 None.

36634 APPLICATION USAGE

36635 After initialization, a state array can be restarted at a different point in one of two ways:

- 36636 1. The *initstate()* function can be used, with the desired seed, state array, and size of the
- 36637 array.
- 36638 2. The *setstate()* function, with the desired state, can be used, followed by *srandom()* with
- 36639 the desired seed. The advantage of using both of these functions is that the size of the
- 36640 state array does not have to be saved once it is initialized.

36641 Although some implementations of *random()* have written messages to standard error, such

36642 implementations do not conform to POSIX.1-200x.

36643 Issue 5 restored the historical behavior of this function.

36644 Threaded applications should use *erand48()*, *nrand48()*, or *rand48()* instead of *random()* when

36645 an independent random number sequence in multiple threads is required.

36646 RATIONALE

36647 None.

36648 FUTURE DIRECTIONS

36649 None.

36650 SEE ALSO

36651 *drand48()*, *rand()*

36652 XBD [<stdlib.h>](#)

36653 CHANGE HISTORY

36654 First released in Issue 4, Version 2.

36655 Issue 5

36656 Moved from X/OPEN UNIX extension to BASE.

36657 In the DESCRIPTION, the phrase “values smaller than 8” is replaced with “values greater than

36658 or equal to 8, or less than 32”, “*size*<8” is replaced with “ $8 \leq \textit{size} < 32$ ”, and a new first paragraph

36659 is added to the RETURN VALUE section. A note is added to the APPLICATION USAGE

36660 indicating that these changes restore the historical behavior of the function.

36661 Issue 6

36662 In the DESCRIPTION, duplicate text “For values greater than or equal to 8 . . .” is removed.

36663 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/30 is applied, removing *rand_r()* from the

36664 list of suggested functions in the APPLICATION USAGE section.

36665 **NAME**
 36666 insque, remque — insert or remove an element in a queue

36667 SYNOPSIS

```
36668 XSI      #include <search.h>
36669
36669 void insque(void *element, void *pred);
36670 void remque(void *element);
```

36671 DESCRIPTION

36672 The *insque()* and *remque()* functions shall manipulate queues built from doubly-linked lists. The
 36673 queue can be either circular or linear. An application using *insque()* or *remque()* shall ensure it
 36674 defines a structure in which the first two members of the structure are pointers to the same type
 36675 of structure, and any further members are application-specific. The first member of the structure
 36676 is a forward pointer to the next entry in the queue. The second member is a backward pointer to
 36677 the previous entry in the queue. If the queue is linear, the queue is terminated with null
 36678 pointers. The names of the structure and of the pointer members are not subject to any special
 36679 restriction.

36680 The *insque()* function shall insert the element pointed to by *element* into a queue immediately
 36681 after the element pointed to by *pred*.

36682 The *remque()* function shall remove the element pointed to by *element* from a queue.

36683 If the queue is to be used as a linear list, invoking *insque(&element, NULL)*, where *element* is the
 36684 initial element of the queue, shall initialize the forward and backward pointers of *element* to null
 36685 pointers.

36686 If the queue is to be used as a circular list, the application shall ensure it initializes the forward
 36687 pointer and the backward pointer of the initial element of the queue to the element's own
 36688 address.

36689 RETURN VALUE

36690 The *insque()* and *remque()* functions do not return a value.

36691 ERRORS

36692 No errors are defined.

36693 EXAMPLES

36694 Creating a Linear Linked List

36695 The following example creates a linear linked list.

```
36696 #include <search.h>
36697 ...
36698 struct myque element1;
36699 struct myque element2;
36700
36700 char *data1 = "DATA1";
36701 char *data2 = "DATA2";
36702 ...
36703 element1.data = data1;
36704 element2.data = data2;
36705
36705 insque (&element1, NULL);
36706 insque (&element2, &element1);
```

36707 **Creating a Circular Linked List**

36708 The following example creates a circular linked list.

```

36709 #include <search.h>
36710 ...
36711 struct myque element1;
36712 struct myque element2;
36713
36714 char *data1 = "DATA1";
36715 char *data2 = "DATA2";
36716 ...
36717 element1.data = data1;
36718 element2.data = data2;
36719
36720 element1.fwd = &element1;
36721 element1.bck = &element1;
36722
36723 insque (&element2, &element1);

```

36721 **Removing an Element**36722 The following example removes the element pointed to by *element1*.

```

36723 #include <search.h>
36724 ...
36725 struct myque element1;
36726 ...
36727 remque (&element1);

```

36728 **APPLICATION USAGE**

36729 The historical implementations of these functions described the arguments as being of type
 36730 **struct qelem *** rather than as being of type **void *** as defined here. In those implementations,
 36731 **struct qelem** was commonly defined in **<search.h>** as:

```

36732 struct qelem {
36733     struct qelem *q_forw;
36734     struct qelem *q_back;
36735 };

```

36736 Applications using these functions, however, were never able to use this structure directly since
 36737 it provided no room for the actual data contained in the elements. Most applications defined
 36738 structures that contained the two pointers as the initial elements and also provided space for, or
 36739 pointers to, the object's data. Applications that used these functions to update more than one
 36740 type of table also had the problem of specifying two or more different structures with the same
 36741 name, if they literally used **struct qelem** as specified.

36742 As described here, the implementations were actually expecting a structure type where the first
 36743 two members were forward and backward pointers to structures. With C compilers that didn't
 36744 provide function prototypes, applications used structures as specified in the DESCRIPTION
 36745 above and the compiler did what the application expected.

36746 If this method had been carried forward with an ISO C standard compiler and the historical
 36747 function prototype, most applications would have to be modified to cast pointers to the
 36748 structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By
 36749 specifying **void *** as the argument type, applications do not need to change (unless they
 36750 specifically referenced **struct qelem** and depended on it being defined in **<search.h>**).

36751

RATIONALE

36752

None.

36753

FUTURE DIRECTIONS

36754

None.

36755

SEE ALSO

36756

XBD <[search.h](#)>

36757

CHANGE HISTORY

36758

First released in Issue 4, Version 2.

36759

Issue 5

36760

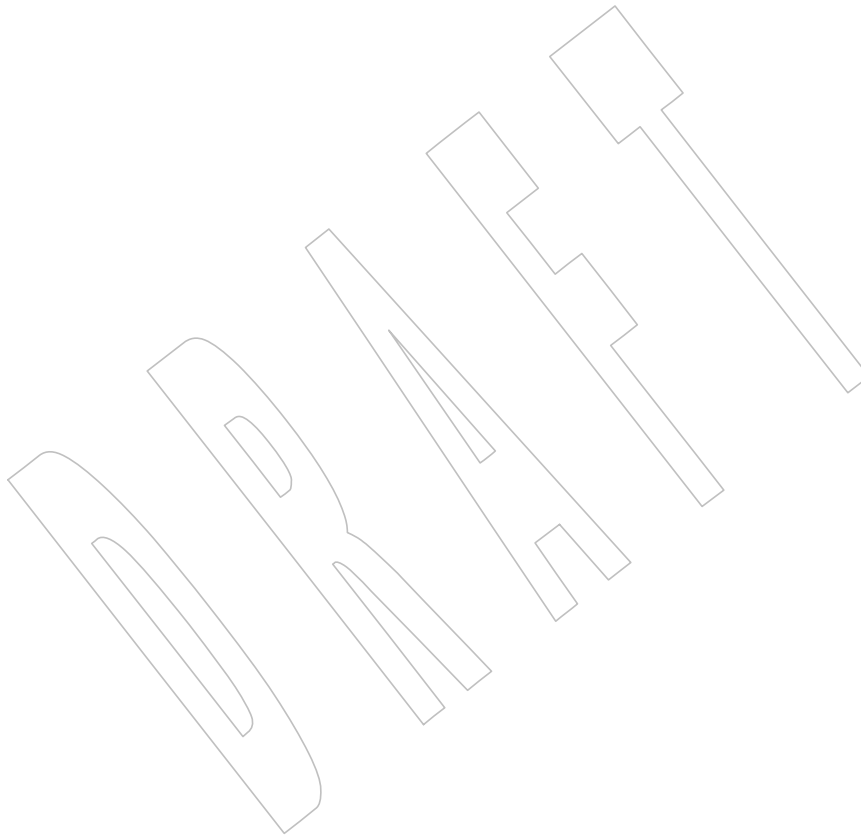
Moved from X/OPEN UNIX extension to BASE.

36761

Issue 6

36762

The normative text is updated to avoid use of the term “must” for application requirements.



36763 **NAME**
 36764 ioctl — control a STREAMS device (**STREAMS**)

36765 **SYNOPSIS**

```
36766 OB XSR #include <stropts.h>
36767 int ioctl(int fildev, int request, ... /* arg */);
```

36768 **DESCRIPTION**

36769 The *ioctl()* function shall perform a variety of control functions on STREAMS devices. For non-
 36770 STREAMS devices, the functions performed by this call are unspecified. The *request* argument
 36771 and an optional third argument (with varying type) shall be passed to and interpreted by the
 36772 appropriate part of the STREAM associated with *fildev*.

36773 The *fildev* argument is an open file descriptor that refers to a device.

36774 The *request* argument selects the control function to be performed and shall depend on the
 36775 STREAMS device being addressed.

36776 The *arg* argument represents additional information that is needed by this specific STREAMS
 36777 device to perform the requested function. The type of *arg* depends upon the particular control
 36778 request, but it shall be either an integer or a pointer to a device-specific data structure.

36779 The *ioctl()* commands applicable to STREAMS, their arguments, and error conditions that apply
 36780 to each individual command are described below.

36781 The following *ioctl()* commands, with error values indicated, are applicable to all STREAMS
 36782 files:

36783 **I_PUSH** Pushes the module whose name is pointed to by *arg* onto the top of the current
 36784 STREAM, just below the STREAM head. It then calls the *open()* function of the
 36785 newly-pushed module.

36786 The *ioctl()* function with the I_PUSH command shall fail if:

36787 [EINVAL] Invalid module name.
 36788 [ENXIO] Open function of new module failed.
 36789 [ENXIO] Hangup received on *fildev*.

36790 **I_POP** Removes the module just below the STREAM head of the STREAM pointed to
 36791 by *fildev*. The *arg* argument should be 0 in an I_POP request.

36792 The *ioctl()* function with the I_POP command shall fail if:

36793 [EINVAL] No module present in the STREAM.
 36794 [ENXIO] Hangup received on *fildev*.

36795 **I_LOOK** Retrieves the name of the module just below the STREAM head of the
 36796 STREAM pointed to by *fildev*, and places it in a character string pointed to by
 36797 *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long,
 36798 where FMNAMESZ is defined in **<stropts.h>**.

36799 The *ioctl()* function with the I_LOOK command shall fail if:

36800 [EINVAL] No module present in the STREAM.

36801	I_FLUSH	Flushes read and/or write queues, depending on the value of <i>arg</i> . Valid <i>arg</i>
36802		values are:
36803	FLUSHR	Flush all read queues.
36804	FLUSHW	Flush all write queues.
36805	FLUSHRW	Flush all read and all write queues.
36806		The <i>ioctl()</i> function with the I_FLUSH command shall fail if:
36807	[EINVAL]	Invalid <i>arg</i> value.
36808	[EAGAIN] or [ENOSR]	Unable to allocate buffers for flush message.
36809		
36810	[ENXIO]	Hangup received on <i>fildev</i> .
36811	I_FLUSHBAND	Flushes a particular band of messages. The <i>arg</i> argument points to a bandinfo
36812		structure. The <i>bi_flag</i> member may be one of FLUSHR, FLUSHW, or
36813		FLUSHRW as described above. The <i>bi_pri</i> member determines the priority
36814		band to be flushed.
36815	I_SETSIG	Requests that the STREAMS implementation send the SIGPOLL signal to the
36816		calling process when a particular event has occurred on the STREAM
36817		associated with <i>fildev</i> . I_SETSIG supports an asynchronous processing
36818		capability in STREAMS. The value of <i>arg</i> is a bitmask that specifies the events
36819		for which the process should be signaled. It is the bitwise-inclusive OR of any
36820		combination of the following constants:
36821	S_RDNORM	A normal (priority band set to 0) message has arrived at the
36822		head of a STREAM head read queue. A signal shall be
36823		generated even if the message is of zero length.
36824	S_RDBAND	A message with a non-zero priority band has arrived at the
36825		head of a STREAM head read queue. A signal shall be
36826		generated even if the message is of zero length.
36827	S_INPUT	A message, other than a high-priority message, has arrived
36828		at the head of a STREAM head read queue. A signal shall be
36829		generated even if the message is of zero length.
36830	S_HIPRI	A high-priority message is present on a STREAM head read
36831		queue. A signal shall be generated even if the message is of
36832		zero length.
36833	S_OUTPUT	The write queue for normal data (priority band 0) just below
36834		the STREAM head is no longer full. This notifies the process
36835		that there is room on the queue for sending (or writing)
36836		normal data downstream.
36837	S_WRNORM	Equivalent to S_OUTPUT.
36838	S_WRBAND	The write queue for a non-zero priority band just below the
36839		STREAM head is no longer full. This notifies the process
36840		that there is room on the queue for sending (or writing)
36841		priority data downstream.
36842	S_MSG	A STREAMS signal message that contains the SIGPOLL
36843		signal has reached the front of the STREAM head read
36844		queue.

36845		S_ERROR	Notification of an error condition has reached the STREAM head.
36846			
36847		S_HANGUP	Notification of a hangup has reached the STREAM head.
36848		S_BANDURG	When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.
36849			
36850			
36851			If <i>arg</i> is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with <i>fdes</i> .
36852			
36853			Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.
36854			
36855			
36856			
36857			The <i>ioctl()</i> function with the I_SETSIG command shall fail if:
36858		[EINVAL]	The value of <i>arg</i> is invalid.
36859		[EINVAL]	The value of <i>arg</i> is 0 and the calling process is not registered to receive the SIGPOLL signal.
36860			
36861		[EAGAIN]	There were insufficient resources to store the signal request.
36862	L_GETSIG		Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an int pointed to by <i>arg</i> , where the events are those specified in the description of I_SETSIG above.
36863			
36864			
36865			
36866			The <i>ioctl()</i> function with the I_GETSIG command shall fail if:
36867		[EINVAL]	Process is not registered to receive the SIGPOLL signal.
36868	L_FIND		Compares the names of all modules currently present in the STREAM to the name pointed to by <i>arg</i> , and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.
36869			
36870			
36871			The <i>ioctl()</i> function with the I_FIND command shall fail if:
36872		[EINVAL]	<i>arg</i> does not contain a valid module name.
36873	L_PEEK		Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to <i>getmsg()</i> except that this command does not remove the message from the queue. The <i>arg</i> argument points to a strpeek structure.
36874			
36875			
36876			
36877			The application shall ensure that the <i>maxlen</i> member in the ctlbuf and databuf structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The <i>flags</i> member may be marked RS_HIPRI or 0, as described by <i>getmsg()</i> . If the process sets <i>flags</i> to RS_HIPRI, for example, L_PEEK shall only look for a high-priority message on the STREAM head read queue.
36878			
36879			
36880			
36881			
36882			
36883			L_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in <i>flags</i> and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, ctlbuf specifies information in the control buffer, databuf specifies information in the data buffer, and <i>flags</i> contains the value RS_HIPRI or 0.
36884			
36885			
36886			
36887			
36888			

36889	I_SRDOPT	Sets the read mode using the value of the argument <i>arg</i> . Read modes are described in <i>read</i> . Valid <i>arg</i> flags are:
36890		
36891		RNORM Byte-stream mode, the default.
36892		RMSGD Message-discard mode.
36893		RMSGN Message-nondiscard mode.
36894		The bitwise-inclusive OR of RMSGD and RMSGN shall return [EINVAL]. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.
36895		
36896		
36897		In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in <i>arg</i> :
36898		
36899		RPROTNORM Fail <i>read</i> () with [EBADMSG] if a message containing a control part is at the front of the STREAM head read queue.
36900		
36901		RPROTDAT Deliver the control part of a message as data when a process issues a <i>read</i> () .
36902		
36903		RPROTDIS Discard the control part of a message, delivering any data portion, when a process issues a <i>read</i> () .
36904		
36905		The <i>ioctl</i> () function with the I_SRDOPT command shall fail if:
36906		[EINVAL] The <i>arg</i> argument is not valid.
36907	I_GRDOPT	Returns the current read mode setting, as described above, in an int pointed to by the argument <i>arg</i> . Read modes are described in <i>read</i> .
36908		
36909	I_NREAD	Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the int pointed to by <i>arg</i> . The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in <i>arg</i> , but the <i>ioctl</i> () return value is greater than 0, this indicates that a zero-length message is next on the queue.
36910		
36911		
36912		
36913		
36914		
36915	I_FDINSERT	Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The <i>arg</i> argument points to a strfdinsert structure.
36916		
36917		
36918		
36919		
36920		The application shall ensure that the <i>len</i> member in the ctlbuf strbuf structure is set to the size of a t_uscalar_t plus the number of bytes of control information to be sent with the message. The <i>fildev</i> member specifies the file descriptor of the other STREAM, and the <i>offset</i> member, which must be suitably aligned for use as a t_uscalar_t , specifies the offset from the start of the control buffer where I_FDINSERT shall store a t_uscalar_t whose interpretation is specific to the STREAM end. The application shall ensure that the <i>len</i> member in the databuf strbuf structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.
36921		
36922		
36923		
36924		
36925		
36926		
36927		
36928		
36929		
36930		The <i>flags</i> member specifies the type of message to be created. A normal message is created if <i>flags</i> is set to 0, and a high-priority message is created if <i>flags</i> is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full
36931		
36932		
36933		
36934		
36935		

36936 and O_NONBLOCK is set. Instead, it fails and sets *errno* to [EAGAIN].

36937 I_FDINSERT also blocks, unless prevented by lack of internal resources,
36938 waiting for the availability of message blocks in the STREAM, regardless of
36939 priority or whether O_NONBLOCK has been specified. No partial message is
36940 sent.

36941 The *ioctl()* function with the I_FDINSERT command shall fail if:

36942 [EAGAIN] A non-priority message is specified, the O_NONBLOCK
36943 flag is set, and the STREAM write queue is full due to
36944 internal flow control conditions.

36945 [EAGAIN] or [ENOSR]
36946 Buffers cannot be allocated for the message that is to be
36947 created.

36948 [EINVAL] One of the following:

- 36949 — The *fildev* member of the **strfdinsert** structure is not a
36950 valid, open STREAM file descriptor.
- 36951 — The size of a **t_uscalar_t** plus *offset* is greater than the
36952 *len* member for the buffer specified through **ctlbuf**.
- 36953 — The *offset* member does not specify a properly-aligned
36954 location in the data buffer.
- 36955 — An undefined value is stored in *flags*.

36956 [ENXIO] Hangup received on the STREAM identified by either the
36957 *fildev* argument or the *fildev* member of the **strfdinsert**
36958 structure.

36959 [ERANGE] The *len* member for the buffer specified through **databuf**
36960 does not fall within the range specified by the maximum
36961 and minimum packet sizes of the topmost STREAM
36962 module; or the *len* member for the buffer specified through
36963 **databuf** is larger than the maximum configured size of the
36964 data part of a message; or the *len* member for the buffer
36965 specified through **ctlbuf** is larger than the maximum
36966 configured size of the control part of a message.

36967 I_STR Constructs an internal STREAMS *ioctl()* message from the data pointed to by
36968 *arg*, and sends that message downstream.

36969 This mechanism is provided to send *ioctl()* requests to downstream modules
36970 and drivers. It allows information to be sent with *ioctl()*, and returns to the
36971 process any information sent upstream by the downstream recipient. I_STR
36972 shall block until the system responds with either a positive or negative
36973 acknowledgement message, or until the request times out after some period of
36974 time. If the request times out, it shall fail with *errno* set to [ETIME].

36975 At most, one I_STR can be active on a STREAM. Further I_STR calls shall
36976 block until the active I_STR completes at the STREAM head. The default
36977 timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has
36978 no effect on this call.

36979 To send requests downstream, the application shall ensure that *arg* points to a
36980 **strioc** structure.

36981 The *ic_cmd* member is the internal *ioctl()* command intended for a

36982 downstream module or driver and *ic_timeout* is the number of seconds
 36983 (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified)
 36984 an I_STR request shall wait for acknowledgement before timing out. *ic_len* is
 36985 the number of bytes in the data argument, and *ic_dp* is a pointer to the data
 36986 argument. The *ic_len* member has two uses: on input, it contains the length of
 36987 the data argument passed in, and on return from the command, it contains the
 36988 number of bytes being returned to the process (the buffer pointed to by *ic_dp*
 36989 should be large enough to contain the maximum amount of data that any
 36990 module or the driver in the STREAM can return).

36991 The STREAM head shall convert the information pointed to by the **strioc**
 36992 structure to an internal *ioctl()* command message and send it downstream.

36993 The *ioctl()* function with the I_STR command shall fail if:

36994 [EAGAIN] or [ENOSR]

36995 Unable to allocate buffers for the *ioctl()* message.

36996 [EINVAL] The *ic_len* member is less than 0 or larger than the
 36997 maximum configured size of the data part of a message, or
 36998 *ic_timeout* is less than −1.

36999 [ENXIO] Hangup received on *fil*des.

37000 [ETIME] A downstream *ioctl()* timed out before acknowledgement
 37001 was received.

37002 An I_STR can also fail while waiting for an acknowledgement if a message
 37003 indicating an error or a hangup is received at the STREAM head. In addition,
 37004 an error code can be returned in the positive or negative acknowledgement
 37005 message, in the event the *ioctl()* command sent downstream fails. For these
 37006 cases, I_STR shall fail with *errno* set to the value in the message.

37007 I_SWROPT Sets the write mode using the value of the argument *arg*. Valid bit settings for
 37008 *arg* are:

37009 SNDZERO Send a zero-length message downstream when a *write()* of 0
 37010 bytes occurs. To not send a zero-length message when a
 37011 *write()* of 0 bytes occurs, the application shall ensure that
 37012 this bit is not set in *arg* (for example, *arg* would be set to 0).

37013 The *ioctl()* function with the I_SWROPT command shall fail if:

37014 [EINVAL] *arg* is not the above value.

37015 I_GWROPT Returns the current write mode setting, as described above, in the **int** that is
 37016 pointed to by the argument *arg*.

37017 I_SENDFD Creates a new reference to the open file description associated with the file
 37018 descriptor *arg*, and writes a message on the STREAMS-based pipe *fil*des
 37019 containing this reference, together with the user ID and group ID of the calling
 37020 process.

37021 The *ioctl()* function with the I_SENDFD command shall fail if:

37022 [EAGAIN] The sending STREAM is unable to allocate a message block
 37023 to contain the file pointer; or the read queue of the receiving
 37024 STREAM head is full and cannot accept the message sent by
 37025 I_SENDFD.

37026		[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37027		[EINVAL]	The <i>fildev</i> argument is not connected to a STREAM pipe.
37028		[ENXIO]	Hangup received on <i>fildev</i> .
37029	I_RECVFD		Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The <i>arg</i> argument is a pointer to a strrecvfd data structure as defined in <stropts.h> .
37030			
37031			
37032			
37033			
37034			The <i>fd</i> member is a file descriptor. The <i>uid</i> and <i>gid</i> members are the effective user ID and effective group ID, respectively, of the sending process.
37035			
37036			If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with <i>errno</i> set to [EAGAIN] if no message is present at the STREAM head.
37037			
37038			
37039			If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the <i>fd</i> member of the strrecvfd structure pointed to by <i>arg</i> .
37040			
37041			
37042			
37043			The <i>ioctl()</i> function with the I_RECVFD command shall fail if:
37044		[EAGAIN]	A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.
37045			
37046		[EBADMSG]	The message at the STREAM head read queue is not a message containing a passed file descriptor.
37047			
37048		[EMFILE]	All file descriptors available to the process are currently open.
37049			
37050		[ENXIO]	Hangup received on <i>fildev</i> .
37051	I_LIST		Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If <i>arg</i> is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by <i>fildev</i> . This lets the process allocate enough space for the module names. Otherwise, it should point to a str_list structure.
37052			
37053			
37054			
37055			
37056			The <i>sl_nmods</i> member indicates the number of entries the process has allocated in the array. Upon return, the <i>sl_modlist</i> member of the str_list structure shall contain the list of module names, and the number of entries that have been filled into the <i>sl_modlist</i> array is found in the <i>sl_nmods</i> member (the number includes the number of modules including the driver). The return value from <i>ioctl()</i> shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (<i>sl_nmods</i>) is satisfied.
37057			
37058			
37059			
37060			
37061			
37062			
37063			
37064			The <i>ioctl()</i> function with the I_LIST command shall fail if:
37065		[EINVAL]	The <i>sl_nmods</i> member is less than 1.
37066		[EAGAIN] or [ENOSR]	
37067			Unable to allocate buffers.
37068	I_ATMARK		Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The <i>arg</i> argument determines how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:
37069			
37070			
37071			

37072	ANYMARK	Check if the message is marked.
37073	LASTMARK	Check if the message is the last one marked on the queue.
37074		The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is permitted.
37075		
37076		The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.
37077		
37078		The <i>ioctl()</i> function with the I_ATMARK command shall fail if:
37079	[EINVAL]	Invalid <i>arg</i> value.
37080	I_CKBAND	Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or -1 on error. <i>arg</i> should be of type int .
37081		
37082		
37083		The <i>ioctl()</i> function with the I_CKBAND command shall fail if:
37084	[EINVAL]	Invalid <i>arg</i> value.
37085	I_GETBAND	Returns the priority band of the first message on the STREAM head read queue in the integer referenced by <i>arg</i> .
37086		
37087		The <i>ioctl()</i> function with the I_GETBAND command shall fail if:
37088	[ENODATA]	No message on the STREAM head read queue.
37089	I_CANPUT	Checks if a certain band is writable. <i>arg</i> is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or -1 on error.
37090		
37091		
37092		The <i>ioctl()</i> function with the I_CANPUT command shall fail if:
37093	[EINVAL]	Invalid <i>arg</i> value.
37094	I_SETCLTIME	This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The <i>arg</i> argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM, an implementation-defined default timeout interval is used.
37095		
37096		
37097		
37098		
37099		
37100		
37101		
37102		The <i>ioctl()</i> function with the I_SETCLTIME command shall fail if:
37103	[EINVAL]	Invalid <i>arg</i> value.
37104	I_GETCLTIME	Returns the close time delay in the integer pointed to by <i>arg</i> .
37105		
37106		
37107		
37108	I_LINK	Connects two STREAMs, where <i>fdes</i> is the file descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the file descriptor of the STREAM connected to another driver. The STREAM designated by <i>arg</i> is connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the STREAM head regarding the connection. This call shall return a multiplexer ID number (an identifier used to disconnect the multiplexer; see I_UNLINK) on success, and -1 on
37109		
37110		
37111		
37112		
37113		
37114		

Multiplexed STREAMS Configurations

The following commands are used for connecting and disconnecting multiplexed STREAMS configurations. These commands use an implementation-defined default timeout interval.

I_LINK Connects two STREAMs, where *fdes* is the file descriptor of the STREAM connected to the multiplexing driver, and *arg* is the file descriptor of the STREAM connected to another driver. The STREAM designated by *arg* is connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the STREAM head regarding the connection. This call shall return a multiplexer ID number (an identifier used to disconnect the multiplexer; see I_UNLINK) on success, and -1 on

37115		failure.
37116		The <i>ioctl()</i> function with the I_LINK command shall fail if:
37117	[ENXIO]	Hangup received on <i>fildev</i> .
37118	[ETIME]	Timeout before acknowledgement message was received at
37119		STREAM head.
37120	[EAGAIN] or [ENOSR]	
37121		Unable to allocate STREAMS storage to perform the
37122		I_LINK.
37123	[EBADF]	The <i>arg</i> argument is not a valid, open file descriptor.
37124	[EINVAL]	The <i>fildev</i> argument does not support multiplexing; or <i>arg</i> is
37125		not a STREAM or is already connected downstream from a
37126		multiplexer; or the specified I_LINK operation would
37127		connect the STREAM head in more than one place in the
37128		multiplexed STREAM.
37129		An I_LINK can also fail while waiting for the multiplexing driver to
37130		acknowledge the request, if a message indicating an error or a hangup is
37131		received at the STREAM head of <i>fildev</i> . In addition, an error code can be
37132		returned in the positive or negative acknowledgement message. For these
37133		cases, I_LINK fails with <i>errno</i> set to the value in the message.
37134	I_UNLINK	Disconnects the two STREAMs specified by <i>fildev</i> and <i>arg</i> . <i>fildev</i> is the file
37135		descriptor of the STREAM connected to the multiplexing driver. The <i>arg</i>
37136		argument is the multiplexer ID number that was returned by the I_LINK
37137		<i>ioctl()</i> command when a STREAM was connected downstream from the
37138		multiplexing driver. If <i>arg</i> is MUXID_ALL, then all STREAMs that were
37139		connected to <i>fildev</i> shall be disconnected. As in I_LINK, this command requires
37140		acknowledgement.
37141		The <i>ioctl()</i> function with the I_UNLINK command shall fail if:
37142	[ENXIO]	Hangup received on <i>fildev</i> .
37143	[ETIME]	Timeout before acknowledgement message was received at
37144		STREAM head.
37145	[EAGAIN] or [ENOSR]	
37146		Unable to allocate buffers for the acknowledgement
37147		message.
37148	[EINVAL]	Invalid multiplexer ID number.
37149		An I_UNLINK can also fail while waiting for the multiplexing driver to
37150		acknowledge the request if a message indicating an error or a hangup is
37151		received at the STREAM head of <i>fildev</i> . In addition, an error code can be
37152		returned in the positive or negative acknowledgement message. For these
37153		cases, I_UNLINK shall fail with <i>errno</i> set to the value in the message.
37154	I_PLINK	Creates a <i>persistent connection</i> between two STREAMs, where <i>fildev</i> is the file
37155		descriptor of the STREAM connected to the multiplexing driver, and <i>arg</i> is the
37156		file descriptor of the STREAM connected to another driver. This call shall
37157		create a persistent connection which can exist even if the file descriptor <i>fildev</i>
37158		associated with the upper STREAM to the multiplexing driver is closed. The
37159		STREAM designated by <i>arg</i> gets connected via a persistent connection below
37160		the multiplexing driver. I_PLINK requires the multiplexing driver to send an
37161		acknowledgement message to the STREAM head. This call shall return a

37162 multiplexer ID number (an identifier that may be used to disconnect the
37163 multiplexer; see I_PUNLINK) on success, and -1 on failure.

37164 The *ioctl()* function with the I_PLINK command shall fail if:

- 37165 [ENXIO] Hangup received on *fildev*.
- 37166 [ETIME] Timeout before acknowledgement message was received at
37167 STREAM head.
- 37168 [EAGAIN] or [ENOSR]
37169 Unable to allocate STREAMS storage to perform the
37170 I_PLINK.
- 37171 [EBADF] The *arg* argument is not a valid, open file descriptor.
- 37172 [EINVAL] The *fildev* argument does not support multiplexing; or *arg* is
37173 not a STREAM or is already connected downstream from a
37174 multiplexer; or the specified I_PLINK operation would
37175 connect the STREAM head in more than one place in the
37176 multiplexed STREAM.

37177 An I_PLINK can also fail while waiting for the multiplexing driver to
37178 acknowledge the request, if a message indicating an error or a hangup is
37179 received at the STREAM head of *fildev*. In addition, an error code can be
37180 returned in the positive or negative acknowledgement message. For these
37181 cases, I_PLINK shall fail with *errno* set to the value in the message.

37182 I_PUNLINK Disconnects the two STREAMS specified by *fildev* and *arg* from a persistent
37183 connection. The *fildev* argument is the file descriptor of the STREAM
37184 connected to the multiplexing driver. The *arg* argument is the multiplexer ID
37185 number that was returned by the I_PLINK *ioctl()* command when a STREAM
37186 was connected downstream from the multiplexing driver. If *arg* is
37187 MUXID_ALL, then all STREAMS which are persistent connections to *fildev*
37188 shall be disconnected. As in I_PLINK, this command requires the multiplexing
37189 driver to acknowledge the request.

37190 The *ioctl()* function with the I_PUNLINK command shall fail if:

- 37191 [ENXIO] Hangup received on *fildev*.
- 37192 [ETIME] Timeout before acknowledgement message was received at
37193 STREAM head.
- 37194 [EAGAIN] or [ENOSR]
37195 Unable to allocate buffers for the acknowledgement
37196 message.
- 37197 [EINVAL] Invalid multiplexer ID number.

37198 An I_PUNLINK can also fail while waiting for the multiplexing driver to
37199 acknowledge the request if a message indicating an error or a hangup is
37200 received at the STREAM head of *fildev*. In addition, an error code can be
37201 returned in the positive or negative acknowledgement message. For these
37202 cases, I_PUNLINK shall fail with *errno* set to the value in the message.

37203 RETURN VALUE

37204 Upon successful completion, *ioctl()* shall return a value other than -1 that depends upon the
37205 STREAMS device control function. Otherwise, it shall return -1 and set *errno* to indicate the
37206 error.

ERRORS37207
37208

Under the following general conditions, *ioctl()* shall fail if:

37209

[EBADF] The *fildev* argument is not a valid open file descriptor.

37210

[EINTR] A signal was caught during the *ioctl()* operation.

37211

[EINVAL] The STREAM or multiplexer referenced by *fildev* is linked (directly or indirectly) downstream from a multiplexer.

37212

37213

If an underlying device driver detects an error, then *ioctl()* shall fail if:

37214

[EINVAL] The *request* or *arg* argument is not valid for this device.

37215

[EIO] Some physical I/O error has occurred.

37216

[ENOTTY] The file associated with the *fildev* argument is not a STREAMS device that accepts control functions.

37217

37218

[ENXIO] The *request* and *arg* arguments are valid for this device driver, but the service requested cannot be performed on this particular sub-device.

37219

37220

[ENODEV] The *fildev* argument refers to a valid STREAMS device, but the corresponding device driver does not support the *ioctl()* function.

37221

37222

If a STREAM is connected downstream from a multiplexer, any *ioctl()* command except *I_UNLINK* and *I_PUNLINK* shall set *errno* to [EINVAL].

37223

EXAMPLES

37224

None.

37225

APPLICATION USAGE

37226

The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

37227

RATIONALE

37228

None.

37229

FUTURE DIRECTIONS

37230

The *ioctl()* function may be removed in a future version.

37231

SEE ALSO

37232

Section 2.6 (on page 473), *close()*, *fcntl()*, *getmsg()*, *open()*, *pipe()*, *poll()*, *putmsg()*, *read*, *sigaction()*, *write*

37233

37234

XBD <[stropts.h](#)>

37235

CHANGE HISTORY

37236

First released in Issue 4, Version 2.

37237

Issue 5

37238

Moved from X/OPEN UNIX extension to BASE.

37239

Issue 6

37240

The Open Group Corrigendum U028/4 is applied, correcting text in the *I_FDINSERT* [EINVAL] case to refer to *ctlbuf*.

37241

37242

This function is marked as part of the XSI STREAMS Option Group.

37243

The normative text is updated to avoid use of the term “must” for application requirements.

37244

Issue 7

37245

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

37246

The *ioctl()* function is marked obsolescent.

37247

37248 **NAME**
 37249 isalnum, isalnum_l — test for an alphanumeric character

37250 SYNOPSIS

```
37251 #include <ctype.h>
37252
37252 int isalnum(int c);
37253 CX int isalnum_l(int c, locale_t locale);
```

37254 DESCRIPTION

37255 CX For *isalnum()*: The functionality described on this reference page is aligned with the ISO C
 37256 standard. Any conflict between the requirements described here and the ISO C standard is
 37257 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37258 CX The *isalnum()* and *isalnum_l()* functions shall test whether *c* is a character of class **alpha** or
 37259 CX **digit** in the current locale of the process, or in the locale represented by *locale*, respectively; see
 37260 XBD Chapter 7 (on page 121).

37261 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
 37262 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
 37263 behavior is undefined.

37264 RETURN VALUE

37265 CX The *isalnum()* and *isalnum_l()* functions shall return non-zero if *c* is an alphanumeric character;
 37266 otherwise, they shall return 0.

37267 ERRORS

37268 The *isalnum_l()* function may fail if:

37269 CX [EINVAL] *locale* is not a valid locale object handle.

37270 EXAMPLES

37271 None.

37272 APPLICATION USAGE

37273 To ensure applications portability, especially across natural languages, only these functions and
 37274 the functions in the reference pages listed in the SEE ALSO section should be used for character
 37275 classification.

37276 RATIONALE

37277 None.

37278 FUTURE DIRECTIONS

37279 None.

37280 SEE ALSO

37281 *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
 37282 *isxdigit()*, *setlocale()*, *uselocale()*

37283 XBD Chapter 7 (on page 121), *<ctype.h>*, *<stdio.h>*

37284 CHANGE HISTORY

37285 First released in Issue 1. Derived from Issue 1 of the SVID.

37286 Issue 6

37287 The normative text is updated to avoid use of the term “must” for application requirements.

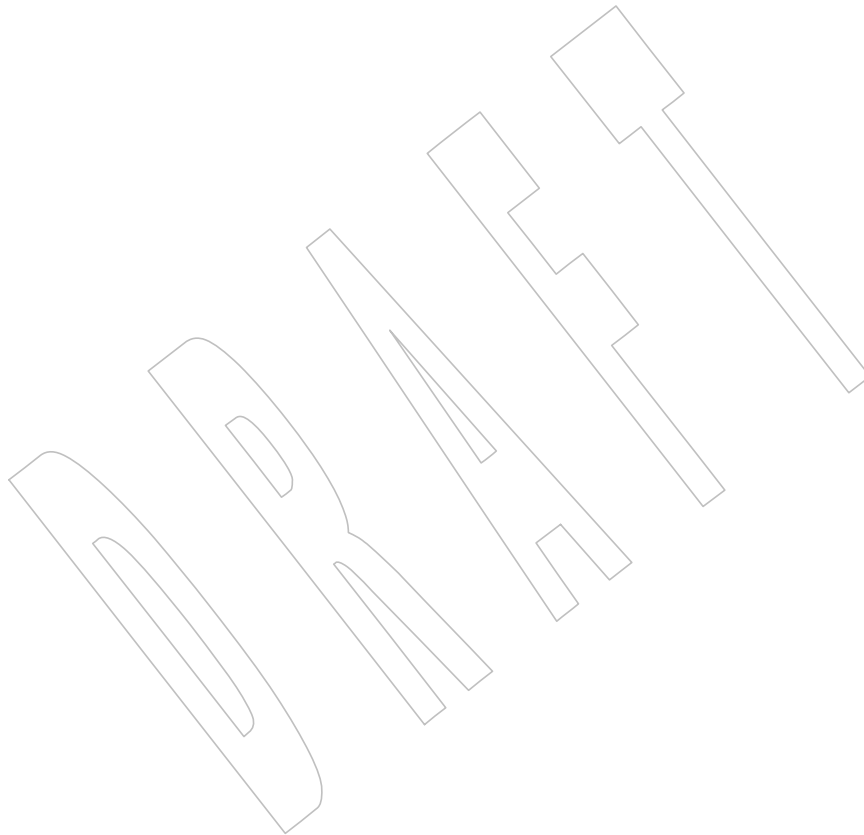
37288

Issue 7

37289

The *isalnum_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

37290



37291 **NAME**
 37292 isalpha, isalpha_l — test for an alphabetic character

37293 **SYNOPSIS**

37294 #include <ctype.h>
 37295 int isalpha(int c);
 37296 CX int isalpha_l(int c, locale_t locale);

37297 **DESCRIPTION**

37298 CX For *isalpha()*: The functionality described on this reference page is aligned with the ISO C
 37299 standard. Any conflict between the requirements described here and the ISO C standard is
 37300 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37301 CX The *isalpha()* and *isalpha_l()* functions shall test whether *c* is a character of class **alpha** in the
 37302 current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 37303 Chapter 7 (on page 121).

37304 The *c* argument is an **int**, the value of which the application shall ensure is representable as an
 37305 **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the
 37306 behavior is undefined.

37307 **RETURN VALUE**

37308 CX The *isalpha()* and *isalpha_l()* functions shall return non-zero if *c* is an alphabetic character;
 37309 otherwise, they shall return 0.

37310 **ERRORS**

37311 The *isalpha_l()* function may fail if:

37312 CX [EINVAL] *locale* is not a valid locale object handle.

37313 **EXAMPLES**

37314 None.

37315 **APPLICATION USAGE**

37316 To ensure applications portability, especially across natural languages, only these functions and
 37317 the functions in the reference pages listed in the SEE ALSO section should be used for character
 37318 classification.

37319 **RATIONALE**

37320 None.

37321 **FUTURE DIRECTIONS**

37322 None.

37323 **SEE ALSO**

37324 *isalnum()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*,
 37325 *isxdigit()*, *setlocale()*, *uselocale()*

37326 XBD Chapter 7 (on page 121), <ctype.h>, <locale.h>, <stdio.h>

37327 **CHANGE HISTORY**

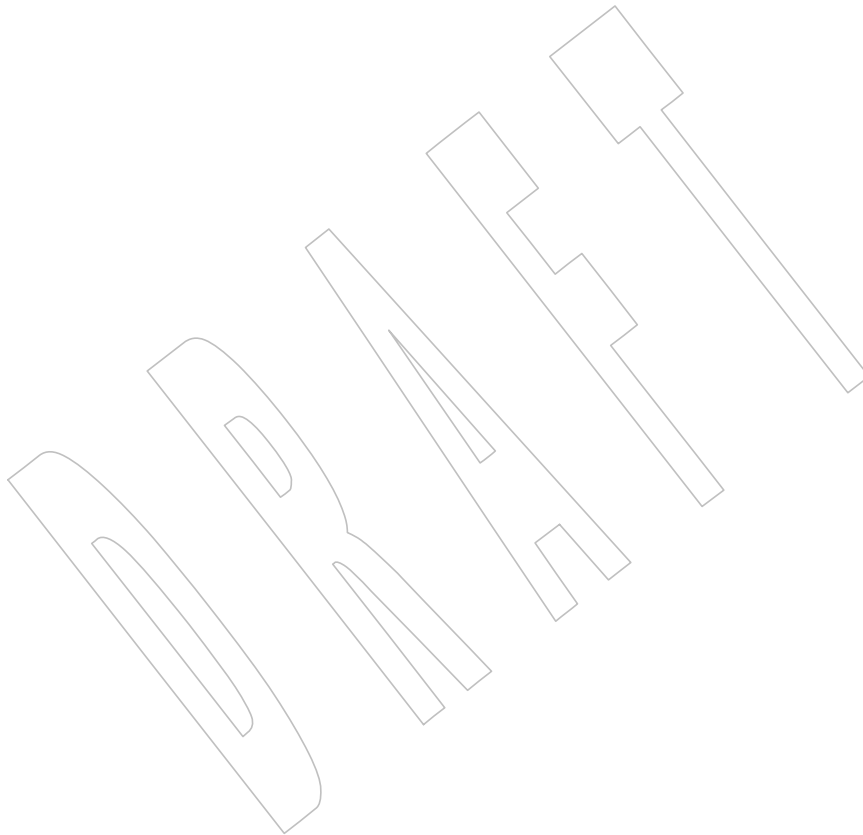
37328 First released in Issue 1. Derived from Issue 1 of the SVID.

37329 **Issue 6**

37330 The normative text is updated to avoid use of the term “must” for application requirements.

37331
37332
37333**Issue 7**

The *isalpha_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



37334 **NAME**
 37335 isascii — test for a 7-bit US-ASCII character

37336 **SYNOPSIS**

37337 OB XSI `#include <ctype.h>`
 37338 `int isascii(int c);`

37339 **DESCRIPTION**

37340 The *isascii()* function shall test whether *c* is a 7-bit US-ASCII character code.

37341 The *isascii()* function is defined on all integer values.

37342 **RETURN VALUE**

37343 The *isascii()* function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and
 37344 octal 0177 inclusive; otherwise, it shall return 0.

37345 **ERRORS**

37346 No errors are defined.

37347 **EXAMPLES**

37348 None.

37349 **APPLICATION USAGE**

37350 The *isascii()* function cannot be used portably in a localized application.

37351 **RATIONALE**

37352 None.

37353 **FUTURE DIRECTIONS**

37354 The *isascii()* function may be removed in a future version.

37355 **SEE ALSO**

37356 XBD [<ctype.h>](#)

37357 **CHANGE HISTORY**

37358 First released in Issue 1. Derived from Issue 1 of the SVID.

37359 **Issue 7**

37360 The *isascii()* function is marked obsolescent.

37361 **NAME**
 37362 `isastream` — test a file descriptor (**STREAMS**)

37363 **SYNOPSIS**

37364 OB XSR `#include <stropts.h>`
 37365 `int isastream(int fildev);`

37366 **DESCRIPTION**

37367 The `isastream()` function shall test whether *fildev*, an open file descriptor, is associated with a
 37368 STREAMS-based file.

37369 **RETURN VALUE**

37370 Upon successful completion, `isastream()` shall return 1 if *fildev* refers to a STREAMS-based file
 37371 and 0 if not. Otherwise, `isastream()` shall return `-1` and set `errno` to indicate the error.

37372 **ERRORS**

37373 The `isastream()` function shall fail if:
 37374 [EBADF] The *fildev* argument is not a valid open file descriptor.

37375 **EXAMPLES**

37376 None.

37377 **APPLICATION USAGE**

37378 None.

37379 **RATIONALE**

37380 None.

37381 **FUTURE DIRECTIONS**

37382 The `isastream()` function may be removed in a future version.

37383 **SEE ALSO**

37384 XBD [<stropts.h>](#)

37385 **CHANGE HISTORY**

37386 First released in Issue 4, Version 2.

37387 **Issue 5**

37388 Moved from X/OPEN UNIX extension to BASE.

37389 **Issue 7**

37390 The `isastream()` function is marked obsolescent.

37391 **NAME**

37392 isatty — test for a terminal device

37393 **SYNOPSIS**

37394 #include <unistd.h>

37395 int isatty(int *fd*);

37396 **DESCRIPTION**

37397 The *isatty()* function shall test whether *fd*, an open file descriptor, is associated with a

37398 terminal device.

37399 **RETURN VALUE**

37400 The *isatty()* function shall return 1 if *fd* is associated with a terminal; otherwise, it shall return

37401 0 and may set *errno* to indicate the error.

37402 **ERRORS**

37403 The *isatty()* function may fail if:

37404 [EBADF] The *fd* argument is not a valid open file descriptor.

37405 [ENOTTY] The file associated with the *fd* argument is not a terminal.

37406 **EXAMPLES**

37407 None.

37408 **APPLICATION USAGE**

37409 The *isatty()* function does not necessarily indicate that a human being is available for interaction

37410 via *fd*. It is quite possible that non-terminal devices are connected to the communications

37411 line.

37412 **RATIONALE**

37413 None.

37414 **FUTURE DIRECTIONS**

37415 None.

37416 **SEE ALSO**

37417 XBD <unistd.h>

37418 **CHANGE HISTORY**

37419 First released in Issue 1. Derived from Issue 1 of the SVID.

37420 **Issue 6**

37421 The following new requirements on POSIX implementations derive from alignment with the

37422 Single UNIX Specification:

37423

- The optional setting of *errno* to indicate an error is added.
- The [EBADF] and [ENOTTY] optional error conditions are added.

37424

37425 **Issue 7**

37426 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

37427 **NAME**
 37428 `isblank, isblank_l` — test for a blank character

37429 **SYNOPSIS**
 37430 `#include <ctype.h>`
 37431 `int isblank(int c);`
 37432 CX `int isblank_l(int c, locale_t locale);`

37433 **DESCRIPTION**
 37434 CX For `isblank()`: The functionality described on this reference page is aligned with the ISO C
 37435 standard. Any conflict between the requirements described here and the ISO C standard is
 37436 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37437 CX The `isblank()` and `isblank_l()` functions shall test whether `c` is a character of class **blank** in the
 37438 current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 37439 Chapter 7 (on page 121).

37440 The `c` argument is a type **int**, the value of which the application shall ensure is a character
 37441 representable as an **unsigned char** or equal to the value of the macro `EOF`. If the argument has
 37442 any other value, the behavior is undefined.

37443 **RETURN VALUE**
 37444 CX The `isblank()` and `isblank_l()` functions shall return non-zero if `c` is a <blank>; otherwise, they
 37445 shall return 0.

37446 **ERRORS**
 37447 The `isblank_l()` function may fail if:
 37448 CX `[EINVAL]` `locale` is not a valid locale object handle.

37449 **EXAMPLES**
 37450 None.

37451 **APPLICATION USAGE**
 37452 To ensure applications portability, especially across natural languages, only these functions and
 37453 the functions in the reference pages listed in the SEE ALSO section should be used for character
 37454 classification.

37455 **RATIONALE**
 37456 None.

37457 **FUTURE DIRECTIONS**
 37458 None.

37459 **SEE ALSO**
 37460 `isalnum()`, `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,
 37461 `isxdigit()`, `setlocale()`, `uselocale()`
 37462 XBD Chapter 7 (on page 121), `<ctype.h>`, `<locale.h>`

37463 **CHANGE HISTORY**
 37464 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37465 **Issue 7**
 37466 The `isblank_l()` function is added from The Open Group Technical Standard, 2006, Extended API
 37467 Set Part 4.

37468 **NAME**
 37469 iscntrl, iscntrl_l — test for a control character

37470 **SYNOPSIS**

```
37471       #include <ctype.h>
37472       int iscntrl(int c);
37473 CX     int iscntrl_l(int c, locale_t locale);
```

37474 **DESCRIPTION**

37475 CX For *iscntrl()*: The functionality described on this reference page is aligned with the ISO C
 37476 standard. Any conflict between the requirements described here and the ISO C standard is
 37477 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37478 CX The *iscntrl()* and *iscntrl_l()* functions shall test whether *c* is a character of class **cntrl** in the
 37479 current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 37480 Chapter 7 (on page 121).

37481 The *c* argument is a type **int**, the value of which the application shall ensure is a character
 37482 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 37483 any other value, the behavior is undefined.

37484 **RETURN VALUE**

37485 CX The *iscntrl()* and *iscntrl_l()* functions shall return non-zero if *c* is a control character; otherwise,
 37486 they shall return 0.

37487 **ERRORS**

37488 The *iscntrl_l()* function may fail if:

37489 CX [EINVAL] *locale* is not a valid locale object handle.

37490 **EXAMPLES**

37491 None.

37492 **APPLICATION USAGE**

37493 To ensure applications portability, especially across natural languages, only these functions and
 37494 the functions in the reference pages listed in the SEE ALSO section should be used for character
 37495 classification.

37496 **RATIONALE**

37497 None.

37498 **FUTURE DIRECTIONS**

37499 None.

37500 **SEE ALSO**

37501 *isalnum()*, *isalpha()*, *isblank()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
 37502 *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

37503 XBD Chapter 7 (on page 121), **<ctype.h>**, **<locale.h>**

37504 **CHANGE HISTORY**

37505 First released in Issue 1. Derived from Issue 1 of the SVID.

37506 **Issue 6**

37507 The normative text is updated to avoid use of the term “must” for application requirements.

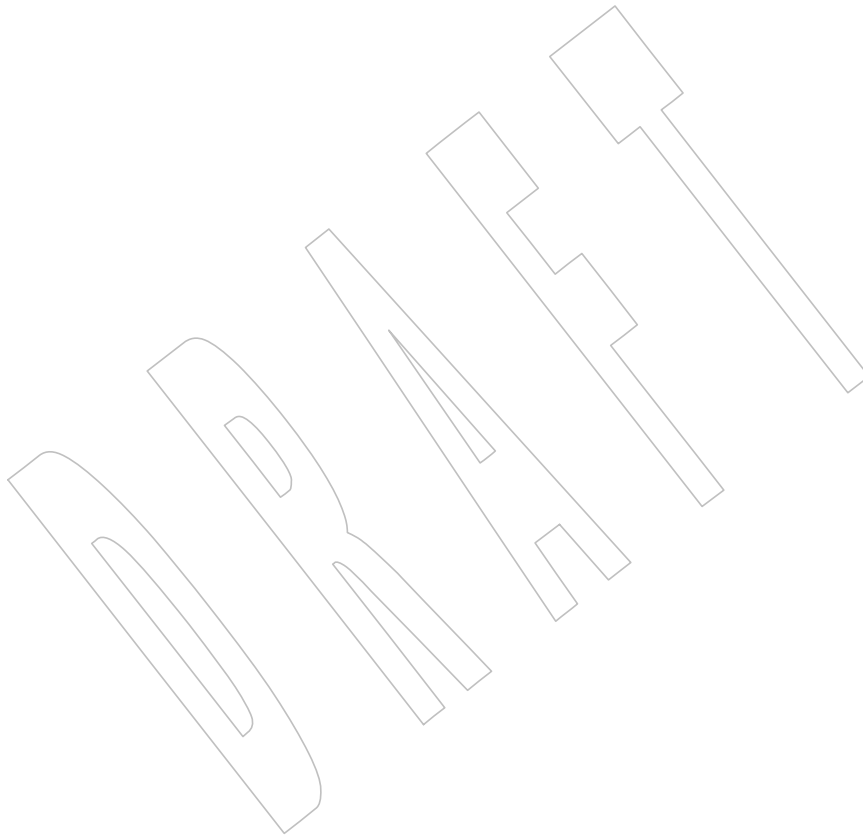
37508

37509

37510

Issue 7

The *iscntrl_I()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



37511 **NAME**

37512 isdigit, isdigit_l — test for a decimal digit

37513 **SYNOPSIS**

37514 #include <ctype.h>

37515 int isdigit(int c);

37516 CX int isdigit_l(int c, locale_t locale);

37517 **DESCRIPTION**37518 CX For *isdigit()*: The functionality described on this reference page is aligned with the ISO C
37519 standard. Any conflict between the requirements described here and the ISO C standard is
37520 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.37521 CX The *isdigit()* and *isdigit_l()* functions shall test whether *c* is a character of class **digit** in the
37522 current locale of the process, or in the locale represented by *locale*, respectively; see XBD
37523 Chapter 7 (on page 121).37524 The *c* argument is an **int**, the value of which the application shall ensure is a character
37525 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
37526 any other value, the behavior is undefined.37527 **RETURN VALUE**37528 CX The *isdigit()* and *isdigit_l()* functions shall return non-zero if *c* is a decimal digit; otherwise,
37529 they shall return 0.37530 **ERRORS**37531 The *isdigit_l()* function may fail if:37532 CX [EINVAL] *locale* is not a valid locale object handle.37533 **EXAMPLES**

37534 None.

37535 **APPLICATION USAGE**37536 To ensure applications portability, especially across natural languages, only these functions and
37537 the functions in the reference pages listed in the SEE ALSO section should be used for character
37538 classification.37539 **RATIONALE**

37540 None.

37541 **FUTURE DIRECTIONS**

37542 None.

37543 **SEE ALSO**37544 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
37545 *isupper()*, *isxdigit()*

37546 XBD Chapter 7 (on page 121), <ctype.h>, <locale.h>

37547 **CHANGE HISTORY**

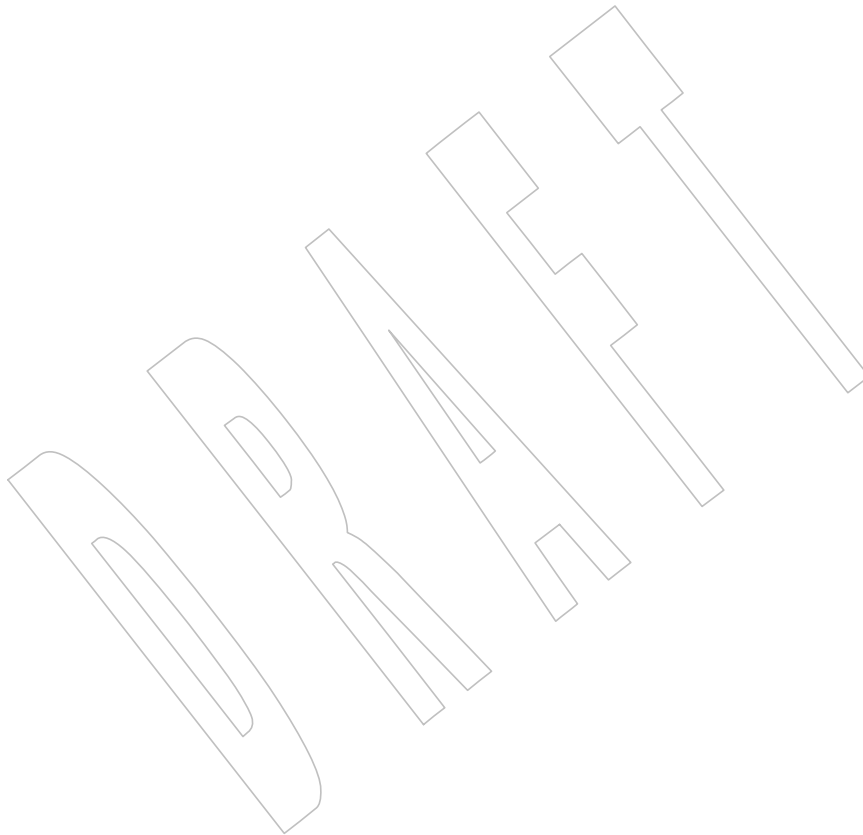
37548 First released in Issue 1. Derived from Issue 1 of the SVID.

37549 **Issue 6**

37550 The normative text is updated to avoid use of the term “must” for application requirements.

37551
37552
37553**Issue 7**

The *isdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



37554 **NAME**
 37555 isfinite — test for finite value

37556 **SYNOPSIS**
 37557 #include <math.h>
 37558 int isfinite(real-floating x);

37559 **DESCRIPTION**
 37560 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37561 conflict between the requirements described here and the ISO C standard is unintentional. This
 37562 volume of POSIX.1-200x defers to the ISO C standard.

37563 The *isfinite()* macro shall determine whether its argument has a finite value (zero, subnormal, or
 37564 normal, and not infinite or NaN). First, an argument represented in a format wider than its
 37565 semantic type is converted to its semantic type. Then determination is based on the type of the
 37566 argument.

37567 **RETURN VALUE**
 37568 The *isfinite()* macro shall return a non-zero value if and only if its argument has a finite value.

37569 **ERRORS**
 37570 No errors are defined.

37571 **EXAMPLES**
 37572 None.

37573 **APPLICATION USAGE**
 37574 None.

37575 **RATIONALE**
 37576 None.

37577 **FUTURE DIRECTIONS**
 37578 None.

37579 **SEE ALSO**
 37580 *fpclassify()*, *isinf()*, *isnan()*, *isnormal()*, *signbit()*

37581 XBD <math.h>

37582 **CHANGE HISTORY**
 37583 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37584 **NAME**
 37585 `isgraph, isgraph_l` — test for a visible character

37586 **SYNOPSIS**
 37587 `#include <ctype.h>`
 37588 `int isgraph(int c);`
 37589 CX `int isgraph_l(int c, locale_t locale);`

37590 **DESCRIPTION**
 37591 CX For `isgraph()`: The functionality described on this reference page is aligned with the ISO C
 37592 standard. Any conflict between the requirements described here and the ISO C standard is
 37593 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37594 CX The `isgraph()` and `isgraph_l()` functions shall test whether `c` is a character of class **graph** in the
 37595 CX current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 37596 Chapter 7 (on page 121).

37597 The `c` argument is an **int**, the value of which the application shall ensure is a character
 37598 representable as an **unsigned char** or equal to the value of the macro `EOF`. If the argument has
 37599 any other value, the behavior is undefined.

37600 **RETURN VALUE**
 37601 CX The `isgraph()` and `isgraph_l()` functions shall return non-zero if `c` is a character with a visible
 37602 representation; otherwise, they shall return 0.

37603 **ERRORS**
 37604 The `isgraph_l()` function may fail if:

37605 CX `[EINVAL]` `locale` is not a valid locale object handle.

37606 **EXAMPLES**
 37607 None.

37608 **APPLICATION USAGE**
 37609 To ensure applications portability, especially across natural languages, only these functions and
 37610 the functions in the reference pages listed in the SEE ALSO section should be used for character
 37611 classification.

37612 **RATIONALE**
 37613 None.

37614 **FUTURE DIRECTIONS**
 37615 None.

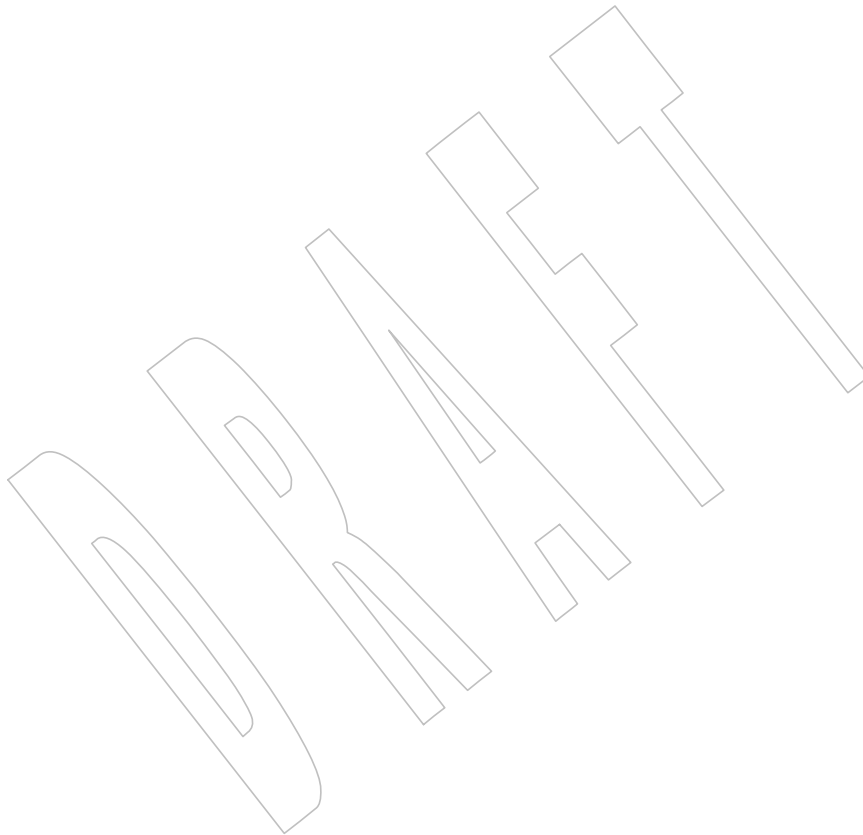
37616 **SEE ALSO**
 37617 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`, `isupper()`,
 37618 `isxdigit()`, `setlocale()`, `uselocale()`
 37619 XBD Chapter 7 (on page 121), `<ctype.h>`, `<locale.h>`

37620 **CHANGE HISTORY**
 37621 First released in Issue 1. Derived from Issue 1 of the SVID.

37622 **Issue 6**
 37623 The normative text is updated to avoid use of the term “must” for application requirements.

37624
37625
37626**Issue 7**

The *isgraph_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



37627 **NAME**37628 `isgreater` — test if x greater than y 37629 **SYNOPSIS**37630 `#include <math.h>`37631 `int isgreater(real-floating x , real-floating y);`37632 **DESCRIPTION**37633 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37634 conflict between the requirements described here and the ISO C standard is unintentional. This
37635 volume of POSIX.1-200x defers to the ISO C standard.37636 The `isgreater()` macro shall determine whether its first argument is greater than its second
37637 argument. The value of `isgreater(x , y)` shall be equal to $(x) > (y)$; however, unlike $(x) > (y)$,
37638 `isgreater(x , y)` shall not raise the invalid floating-point exception when x and y are unordered.37639 **RETURN VALUE**37640 Upon successful completion, the `isgreater()` macro shall return the value of $(x) > (y)$.37641 If x or y is NaN, 0 shall be returned.37642 **ERRORS**

37643 No errors are defined.

37644 **EXAMPLES**

37645 None.

37646 **APPLICATION USAGE**37647 The relational and equality operators support the usual mathematical relationships between
37648 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
37649 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
37650 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
37651 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
37652 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
37653 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
37654 indicates that the argument shall be an expression of **real-floating** type.37655 **RATIONALE**

37656 None.

37657 **FUTURE DIRECTIONS**

37658 None.

37659 **SEE ALSO**37660 [*isgreaterequal\(\)*](#), [*isless\(\)*](#), [*islessequal\(\)*](#), [*islessgreater\(\)*](#), [*isunordered\(\)*](#)37661 XBD `<math.h>`37662 **CHANGE HISTORY**

37663 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37664 **NAME**
 37665 `isgreaterequal` — test if x is greater than or equal to y

37666 **SYNOPSIS**
 37667 `#include <math.h>`
 37668 `int isgreaterequal(real-floating x , real-floating y);`

37669 **DESCRIPTION**
 37670 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37671 conflict between the requirements described here and the ISO C standard is unintentional. This
 37672 volume of POSIX.1-200x defers to the ISO C standard.

37673 The `isgreaterequal()` macro shall determine whether its first argument is greater than or equal to
 37674 its second argument. The value of `isgreaterequal(x , y)` shall be equal to $(x) \geq (y)$; however, unlike
 37675 $(x) \geq (y)$, `isgreaterequal(x , y)` shall not raise the invalid floating-point exception when x and y are
 37676 unordered.

37677 **RETURN VALUE**
 37678 Upon successful completion, the `isgreaterequal()` macro shall return the value of $(x) \geq (y)$.
 37679 If x or y is NaN, 0 shall be returned.

37680 **ERRORS**
 37681 No errors are defined.

37682 **EXAMPLES**
 37683 None.

37684 **APPLICATION USAGE**
 37685 The relational and equality operators support the usual mathematical relationships between
 37686 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 37687 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 37688 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 37689 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 37690 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 37691 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 37692 indicates that the argument shall be an expression of **real-floating** type.

37693 **RATIONALE**
 37694 None.

37695 **FUTURE DIRECTIONS**
 37696 None.

37697 **SEE ALSO**
 37698 [isgreater\(\)](#), [isless\(\)](#), [islessequal\(\)](#), [islessgreater\(\)](#), [isunordered\(\)](#)
 37699 XBD [<math.h>](#)

37700 **CHANGE HISTORY**
 37701 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37702 **NAME**

37703 isinf — test for infinity

37704 **SYNOPSIS**

37705 #include <math.h>

37706 int isinf(real-floating x);

37707 **DESCRIPTION**37708 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37709 conflict between the requirements described here and the ISO C standard is unintentional. This
37710 volume of POSIX.1-200x defers to the ISO C standard.37711 The *isinf()* macro shall determine whether its argument value is an infinity (positive or
37712 negative). First, an argument represented in a format wider than its semantic type is converted
37713 to its semantic type. Then determination is based on the type of the argument.37714 **RETURN VALUE**37715 The *isinf()* macro shall return a non-zero value if and only if its argument has an infinite value.37716 **ERRORS**

37717 No errors are defined.

37718 **EXAMPLES**

37719 None.

37720 **APPLICATION USAGE**

37721 None.

37722 **RATIONALE**

37723 None.

37724 **FUTURE DIRECTIONS**

37725 None.

37726 **SEE ALSO**37727 *fpclassify()*, *isfinite()*, *isnan()*, *isnormal()*, *signbit()*

37728 XBD <math.h>

37729 **CHANGE HISTORY**

37730 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

37731 **NAME**37732 isless — test if x is less than y 37733 **SYNOPSIS**

37734 #include <math.h>

37735 int isless(real-floating x , real-floating y);37736 **DESCRIPTION**37737 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37738 conflict between the requirements described here and the ISO C standard is unintentional. This
37739 volume of POSIX.1-200x defers to the ISO C standard.37740 The *isless()* macro shall determine whether its first argument is less than its second argument.
37741 The value of *isless*(x , y) shall be equal to $(x) < (y)$; however, unlike $(x) < (y)$, *isless*(x , y) shall not
37742 raise the invalid floating-point exception when x and y are unordered.37743 **RETURN VALUE**37744 Upon successful completion, the *isless()* macro shall return the value of $(x) < (y)$.37745 If x or y is NaN, 0 shall be returned.37746 **ERRORS**

37747 No errors are defined.

37748 **EXAMPLES**

37749 None.

37750 **APPLICATION USAGE**37751 The relational and equality operators support the usual mathematical relationships between
37752 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
37753 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
37754 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
37755 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
37756 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
37757 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
37758 indicates that the argument shall be an expression of **real-floating** type.37759 **RATIONALE**

37760 None.

37761 **FUTURE DIRECTIONS**

37762 None.

37763 **SEE ALSO**37764 *isgreater()*, *isgreaterequal()*, *islessequal()*, *islessgreater()*, *isunordered()*

37765 XBD <math.h>

37766 **CHANGE HISTORY**

37767 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37768 **NAME**37769 `islessequal` — test if x is less than or equal to y 37770 **SYNOPSIS**37771 `#include <math.h>`37772 `int islessequal(real-floating x , real-floating y);`37773 **DESCRIPTION**37774 CX The functionality described on this reference page is aligned with the ISO C standard. Any
37775 conflict between the requirements described here and the ISO C standard is unintentional. This
37776 volume of POSIX.1-200x defers to the ISO C standard.37777 The `islessequal()` macro shall determine whether its first argument is less than or equal to its
37778 second argument. The value of `islessequal(x , y)` shall be equal to $(x) \leq (y)$; however, unlike
37779 $(x) \leq (y)$, `islessequal(x , y)` shall not raise the invalid floating-point exception when x and y are
37780 unordered.37781 **RETURN VALUE**37782 Upon successful completion, the `islessequal()` macro shall return the value of $(x) \leq (y)$.37783 If x or y is NaN, 0 shall be returned.37784 **ERRORS**

37785 No errors are defined.

37786 **EXAMPLES**

37787 None.

37788 **APPLICATION USAGE**37789 The relational and equality operators support the usual mathematical relationships between
37790 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
37791 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
37792 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
37793 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
37794 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
37795 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
37796 indicates that the argument shall be an expression of **real-floating** type.37797 **RATIONALE**

37798 None.

37799 **FUTURE DIRECTIONS**

37800 None.

37801 **SEE ALSO**37802 [*isgreater\(\), isgreaterequal\(\), isless\(\), islessgreater\(\), isunordered\(\)*](#)37803 XBD [**<math.h>**](#)37804 **CHANGE HISTORY**

37805 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37806 **NAME**
 37807 islessgreater — test if x is less than or greater than y

37808 **SYNOPSIS**
 37809 #include <math.h>
 37810 int islessgreater(real-floating x , real-floating y);

37811 **DESCRIPTION**
 37812 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37813 conflict between the requirements described here and the ISO C standard is unintentional. This
 37814 volume of POSIX.1-200x defers to the ISO C standard.

37815 The *islessgreater()* macro shall determine whether its first argument is less than or greater than
 37816 its second argument. The *islessgreater(x , y)* macro is similar to $(x) < (y) \parallel (x) > (y)$; however,
 37817 *islessgreater(x , y)* shall not raise the invalid floating-point exception when x and y are unordered
 37818 (nor shall it evaluate x and y twice).

37819 **RETURN VALUE**
 37820 Upon successful completion, the *islessgreater()* macro shall return the value of
 37821 $(x) < (y) \parallel (x) > (y)$.
 37822 If x or y is NaN, 0 shall be returned.

37823 **ERRORS**
 37824 No errors are defined.

37825 **EXAMPLES**
 37826 None.

37827 **APPLICATION USAGE**
 37828 The relational and equality operators support the usual mathematical relationships between
 37829 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
 37830 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
 37831 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
 37832 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
 37833 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
 37834 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
 37835 indicates that the argument shall be an expression of **real-floating** type.

37836 **RATIONALE**
 37837 None.

37838 **FUTURE DIRECTIONS**
 37839 None.

37840 **SEE ALSO**
 37841 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *isunordered()*
 37842 XBD <math.h>

37843 **CHANGE HISTORY**
 37844 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37845 **NAME**
 37846 `islower, islower_l` — test for a lowercase letter

37847 **SYNOPSIS**
 37848 `#include <ctype.h>`
 37849 `int islower(int c);`
 37850 CX `int islower_l(int c, locale_t locale);`

37851 **DESCRIPTION**
 37852 CX For `islower()`: The functionality described on this reference page is aligned with the ISO C
 37853 standard. Any conflict between the requirements described here and the ISO C standard is
 37854 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

37855 CX The `islower()` and `islower_l()` functions shall test whether `c` is a character of class **lower** in the
 37856 current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 37857 Chapter 7 (on page 121).

37858 The `c` argument is an **int**, the value of which the application shall ensure is a character
 37859 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 37860 any other value, the behavior is undefined.

37861 **RETURN VALUE**
 37862 CX The `islower()` and `islower_l()` functions shall return non-zero if `c` is a lowercase letter; otherwise,
 37863 they shall return 0.

37864 **ERRORS**
 37865 The `islower_l()` function may fail if:
 37866 CX **[EINVAL]** `locale` is not a valid locale object handle.

37867 EXAMPLES

37868 Testing for a Lowercase Letter

37869 Two examples follow, the first using `islower()`, the second using multiple concurrent locales and
 37870 `islower_l()`.

37871 The examples test whether the value is a lowercase letter, based on the locale of the user, then
 37872 use it as part of a key value.

```
37873 /* Example 1 -- using islower() */
37874 #include <ctype.h>
37875 #include <stdlib.h>
37876 #include <locale.h>
37877 ...
37878 char *keystr;
37879 int elementlen, len;
37880 char c;
37881 ...
37882 setlocale(LC_ALL, "");
37883 ...
37884 len = 0;
37885 while (len < elementlen) {
37886     c = (char) (rand() % 256);
37887     ...
37888     if (islower(c))
```



```

37889         keystr[len++] = c;
37890     }
37891     ...
37892     /* Example 2 -- using islower_l() */
37893     #include <ctype.h>
37894     #include <stdlib.h>
37895     #include <locale.h>
37896     ...
37897     char *keystr;
37898     int elementlen, len;
37899     char c;
37900     ...
37901     locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
37902     ...
37903     len = 0;
37904     while (len < elementlen) {
37905         c = (char) (rand() % 256);
37906         ...
37907         if (islower_l(c, loc))
37908             keystr[len++] = c;
37909     }
37910     ...

```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum(), *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *isxdigit()*, *setlocale()*, *uselocale()*

XBD Chapter 7 (on page 121), [<ctype.h>](#), [<locale.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

An example is added.

Issue 7

The *islower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

37931 **NAME**37932 **isnan** — test for a NaN37933 **SYNOPSIS**

37934 #include <math.h>

37935 int isnan(real-floating x);

37936 **DESCRIPTION**37937 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
37938 conflict between the requirements described here and the ISO C standard is unintentional. This
37939 volume of POSIX.1-200x defers to the ISO C standard.37940 The *isnan()* macro shall determine whether its argument value is a NaN. First, an argument
37941 represented in a format wider than its semantic type is converted to its semantic type. Then
37942 determination is based on the type of the argument.37943 **RETURN VALUE**37944 The *isnan()* macro shall return a non-zero value if and only if its argument has a NaN value.37945 **ERRORS**

37946 No errors are defined.

37947 **EXAMPLES**

37948 None.

37949 **APPLICATION USAGE**

37950 None.

37951 **RATIONALE**

37952 None.

37953 **FUTURE DIRECTIONS**

37954 None.

37955 **SEE ALSO**37956 *fpclassify()*, *isfinite()*, *isinf()*, *isnormal()*, *signbit()*

37957 XBD <math.h>

37958 **CHANGE HISTORY**

37959 First released in Issue 3.

37960 **Issue 5**37961 The DESCRIPTION is updated to indicate the return value when NaN is not supported. This
37962 text was previously published in the APPLICATION USAGE section.37963 **Issue 6**

37964 Re-written for alignment with the ISO/IEC 9899:1999 standard.

37965 **NAME**
 37966 isnormal — test for a normal value

37967 **SYNOPSIS**
 37968 #include <math.h>
 37969 int isnormal(real-floating x);

37970 **DESCRIPTION**
 37971 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 37972 conflict between the requirements described here and the ISO C standard is unintentional. This
 37973 volume of POSIX.1-200x defers to the ISO C standard.

37974 The *isnormal()* macro shall determine whether its argument value is normal (neither zero,
 37975 subnormal, infinite, nor NaN). First, an argument represented in a format wider than its
 37976 semantic type is converted to its semantic type. Then determination is based on the type of the
 37977 argument.

37978 **RETURN VALUE**
 37979 The *isnormal()* macro shall return a non-zero value if and only if its argument has a normal
 37980 value.

37981 **ERRORS**
 37982 No errors are defined.

37983 **EXAMPLES**
 37984 None.

37985 **APPLICATION USAGE**
 37986 None.

37987 **RATIONALE**
 37988 None.

37989 **FUTURE DIRECTIONS**
 37990 None.

37991 **SEE ALSO**
 37992 *fpclassify()*, *isfinite()*, *isinf()*, *isnan()*, *signbit()*

37993 XBD <math.h>

37994 **CHANGE HISTORY**
 37995 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

37996 **NAME**
 37997 `isprint, isprint_l` — test for a printable character

37998 **SYNOPSIS**
 37999 `#include <ctype.h>`
 38000 `int isprint(int c);`
 38001 CX `int isprint_l(int c, locale_t locale);`

38002 **DESCRIPTION**
 38003 CX For `isprint()`: The functionality described on this reference page is aligned with the ISO C
 38004 standard. Any conflict between the requirements described here and the ISO C standard is
 38005 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38006 CX The `isprint()` and `isprint_l()` functions shall test whether `c` is a character of class **print** in the
 38007 current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 38008 Chapter 7 (on page 121).

38009 The `c` argument is an **int**, the value of which the application shall ensure is a character
 38010 representable as an **unsigned char** or equal to the value of the macro `EOF`. If the argument has
 38011 any other value, the behavior is undefined.

38012 **RETURN VALUE**
 38013 CX The `isprint()` and `isprint_l()` functions shall return non-zero if `c` is a printable character;
 38014 otherwise, they shall return 0.

38015 **ERRORS**
 38016 The `isprint_l()` function may fail if:
 38017 CX `[EINVAL]` `locale` is not a valid locale object handle.

38018 **EXAMPLES**
 38019 None.

38020 **APPLICATION USAGE**
 38021 To ensure applications portability, especially across natural languages, only these functions and
 38022 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38023 classification.

38024 **RATIONALE**
 38025 None.

38026 **FUTURE DIRECTIONS**
 38027 None.

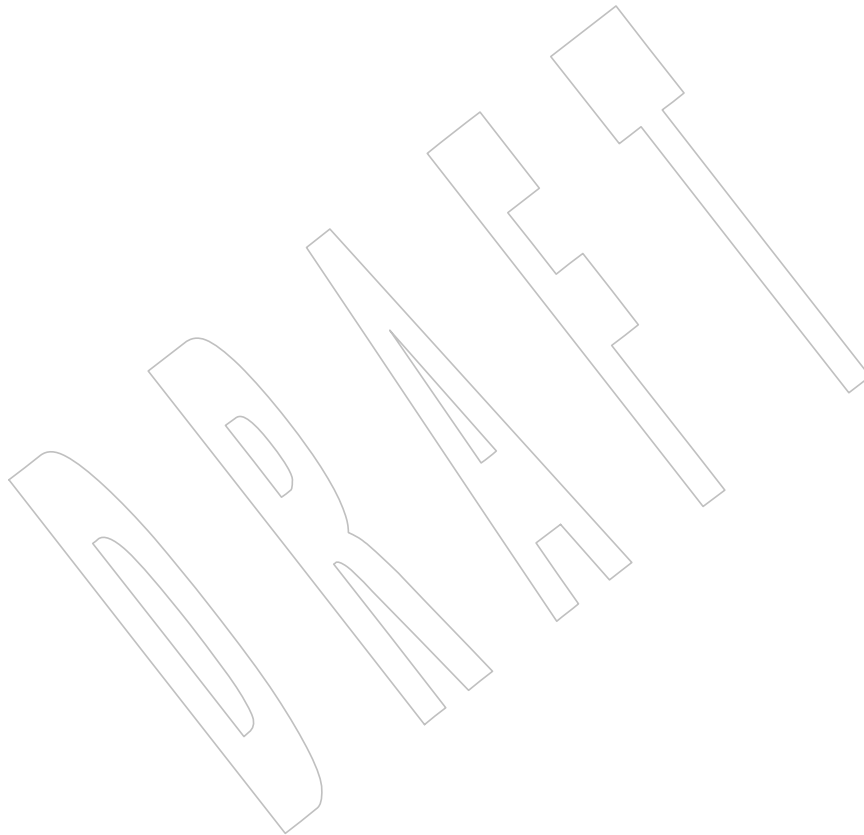
38028 **SEE ALSO**
 38029 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `ispunct()`, `isspace()`, `isupper()`,
 38030 `isxdigit()`, `setlocale()`, `uselocale()`
 38031 XBD Chapter 7 (on page 121), `<ctype.h>`, `<locale.h>`

38032 **CHANGE HISTORY**
 38033 First released in Issue 1. Derived from Issue 1 of the SVID.

38034 **Issue 6**
 38035 The normative text is updated to avoid use of the term “must” for application requirements.

38036
38037
38038**Issue 7**

The *isprint_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38039 **NAME**
 38040 `ispunct, ispunct_l` — test for a punctuation character

38041 **SYNOPSIS**
 38042 `#include <ctype.h>`
 38043 `int ispunct(int c);`
 38044 CX `int ispunct_l(int c, locale_t locale);`

38045 **DESCRIPTION**
 38046 CX For *ispunct()*: The functionality described on this reference page is aligned with the ISO C
 38047 standard. Any conflict between the requirements described here and the ISO C standard is
 38048 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38049 CX The *ispunct()* and *ispunct_l()* functions shall test whether *c* is a character of class **punct** in the
 38050 current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 38051 Chapter 7 (on page 121).

38052 The *c* argument is an **int**, the value of which the application shall ensure is a character
 38053 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38054 any other value, the behavior is undefined.

38055 **RETURN VALUE**
 38056 CX The *ispunct()* and *ispunct_l()* functions shall return non-zero if *c* is a punctuation character;
 38057 otherwise, they shall return 0.

38058 **ERRORS**
 38059 The *ispunct_l()* function may fail if:

38060 CX **[EINVAL]** *locale* is not a valid locale object handle.

38061 **EXAMPLES**
 38062 None.

38063 **APPLICATION USAGE**
 38064 To ensure applications portability, especially across natural languages, only these functions and
 38065 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38066 classification.

38067 **RATIONALE**
 38068 None.

38069 **FUTURE DIRECTIONS**
 38070 None.

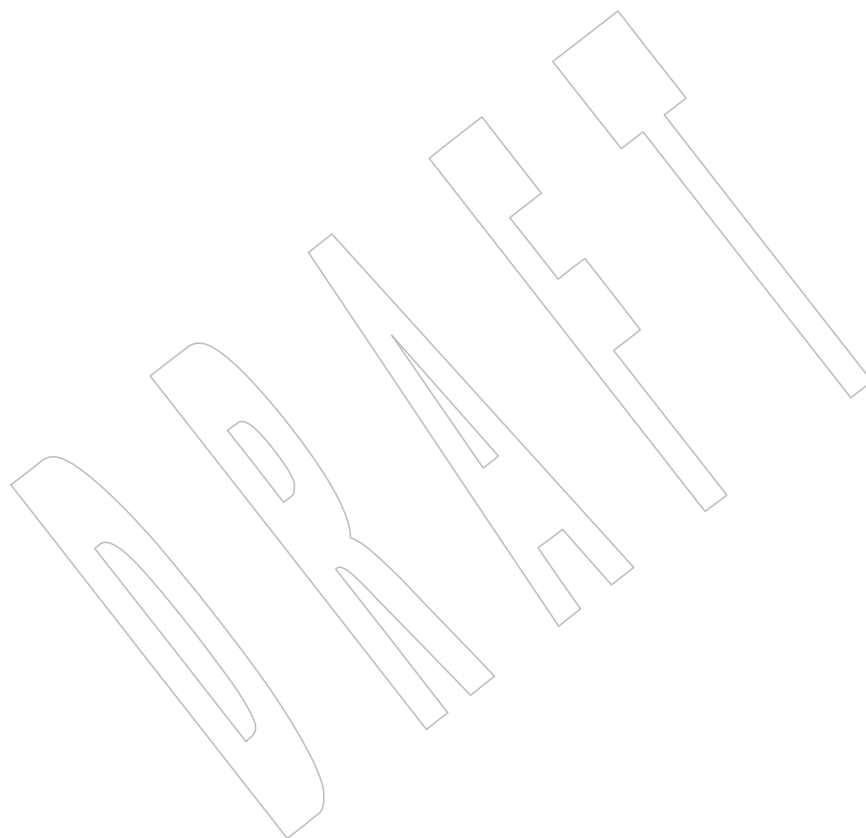
38071 **SEE ALSO**
 38072 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *isspace()*, *isupper()*,
 38073 *isxdigit()*, *setlocale()*, *uselocale()*
 38074 XBD Chapter 7 (on page 121), **<ctype.h>**, **<locale.h>**

38075 **CHANGE HISTORY**
 38076 First released in Issue 1. Derived from Issue 1 of the SVID.

38077 **Issue 6**
 38078 The normative text is updated to avoid use of the term “must” for application requirements.

38079
38080
38081**Issue 7**

The *ispunct_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38082 **NAME**
 38083 `isspace, isspace_l` — test for a white-space character

38084 **SYNOPSIS**
 38085 `#include <ctype.h>`
 38086 `int isspace(int c);`
 38087 CX `int isspace_l(int c, locale_t locale);`

38088 **DESCRIPTION**
 38089 CX For `isspace()`: The functionality described on this reference page is aligned with the ISO C
 38090 standard. Any conflict between the requirements described here and the ISO C standard is
 38091 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38092 CX The `isspace()` and `isspace_l()` functions shall test whether `c` is a character of class **space** in the
 38093 current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 38094 Chapter 7 (on page 121).

38095 The `c` argument is an **int**, the value of which the application shall ensure is a character
 38096 representable as an **unsigned char** or equal to the value of the macro `EOF`. If the argument has
 38097 any other value, the behavior is undefined.

38098 **RETURN VALUE**
 38099 CX The `isspace()` and `isspace_l()` functions shall return non-zero if `c` is a white-space character;
 38100 otherwise, they shall return 0.

38101 **ERRORS**
 38102 The `isspace_l()` function may fail if:
 38103 CX `[EINVAL]` `locale` is not a valid locale object handle.

38104 **EXAMPLES**
 38105 None.

38106 **APPLICATION USAGE**
 38107 To ensure applications portability, especially across natural languages, only these functions and
 38108 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38109 classification.

38110 **RATIONALE**
 38111 None.

38112 **FUTURE DIRECTIONS**
 38113 None.

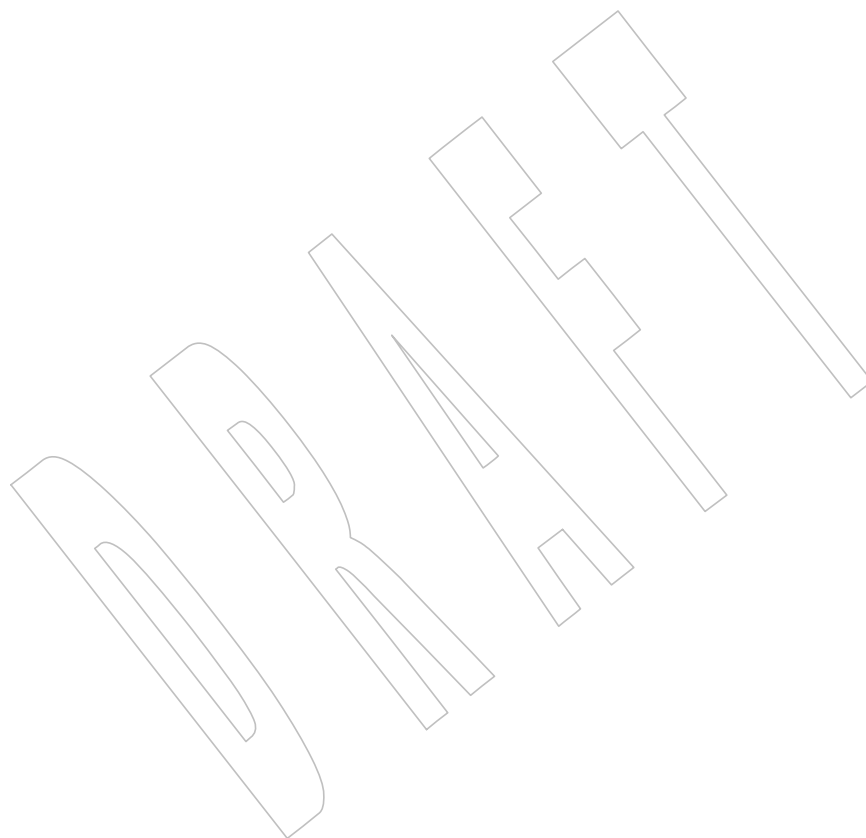
38114 **SEE ALSO**
 38115 `isalnum()`, `isalpha()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isupper()`,
 38116 `isxdigit()`, `setlocale()`, `uselocale()`
 38117 XBD Chapter 7 (on page 121), `<ctype.h>`, `<locale.h>`

38118 **CHANGE HISTORY**
 38119 First released in Issue 1. Derived from Issue 1 of the SVID.

38120 **Issue 6**
 38121 The normative text is updated to avoid use of the term “must” for application requirements.

38122
38123
38124**Issue 7**

The *isspace_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38125 **NAME**38126 `isunordered` — test if arguments are unordered38127 **SYNOPSIS**38128 `#include <math.h>`38129 `int isunordered(real-floating x, real-floating y);`38130 **DESCRIPTION**38131 CX The functionality described on this reference page is aligned with the ISO C standard. Any
38132 conflict between the requirements described here and the ISO C standard is unintentional. This
38133 volume of POSIX.1-200x defers to the ISO C standard.38134 The *isunordered()* macro shall determine whether its arguments are unordered.38135 **RETURN VALUE**38136 Upon successful completion, the *isunordered()* macro shall return 1 if its arguments are
38137 unordered, and 0 otherwise.38138 If *x* or *y* is NaN, 1 shall be returned.38139 **ERRORS**

38140 No errors are defined.

38141 **EXAMPLES**

38142 None.

38143 **APPLICATION USAGE**38144 The relational and equality operators support the usual mathematical relationships between
38145 numeric values. For any ordered pair of numeric values, exactly one of the relationships (less,
38146 greater, and equal) is true. Relational operators may raise the invalid floating-point exception
38147 when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the
38148 unordered relationship is true. This macro is a quiet (non-floating-point exception raising)
38149 version of a relational operator. It facilitates writing efficient code that accounts for NaNs
38150 without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating**
38151 indicates that the argument shall be an expression of **real-floating** type.38152 **RATIONALE**

38153 None.

38154 **FUTURE DIRECTIONS**

38155 None.

38156 **SEE ALSO**38157 *isgreater()*, *isgreaterequal()*, *isless()*, *islessequal()*, *islessgreater()*38158 XBD **<math.h>**38159 **CHANGE HISTORY**

38160 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38161 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/50 is applied, correcting the RETURN
38162 VALUE section when *x* or *y* is NaN.

38163 **NAME**
 38164 `isupper, isupper_l` — test for an uppercase letter

38165 **SYNOPSIS**
 38166 `#include <ctype.h>`
 38167 `int isupper(int c);`
 38168 CX `int isupper_l(int c, locale_t locale);`

38169 **DESCRIPTION**
 38170 CX For `isupper()`: The functionality described on this reference page is aligned with the ISO C
 38171 standard. Any conflict between the requirements described here and the ISO C standard is
 38172 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38173 CX The `isupper()` and `isupper_l()` functions shall test whether `c` is a character of class **upper** in the
 38174 current locale of the process, or in the locale represented by `locale`, respectively; see XBD
 38175 Chapter 7 (on page 121).

38176 The `c` argument is an **int**, the value of which the application shall ensure is a character
 38177 representable as an **unsigned char** or equal to the value of the macro `EOF`. If the argument has
 38178 any other value, the behavior is undefined.

38179 **RETURN VALUE**
 38180 CX The `isupper()` and `isupper_l()` functions shall return non-zero if `c` is an uppercase letter;
 38181 otherwise, they shall return 0.

38182 **ERRORS**
 38183 The `isupper_l()` function may fail if:

38184 CX `[EINVAL]` `locale` is not a valid locale object handle.

38185 **EXAMPLES**
 38186 None.

38187 **APPLICATION USAGE**
 38188 To ensure applications portability, especially across natural languages, only these functions and
 38189 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38190 classification.

38191 **RATIONALE**
 38192 None.

38193 **FUTURE DIRECTIONS**
 38194 None.

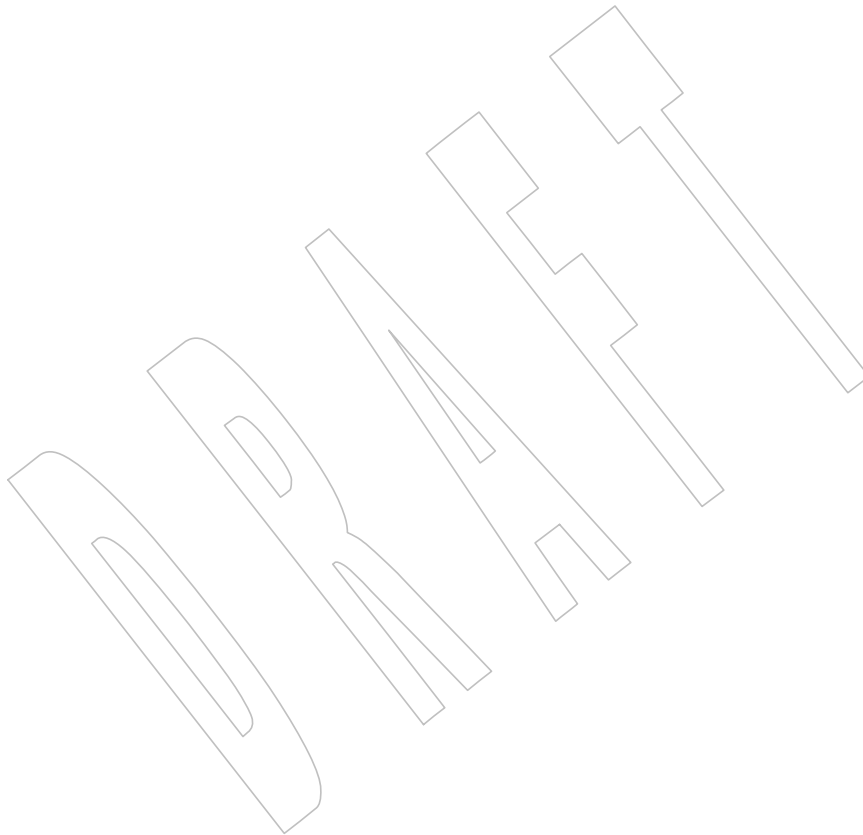
38195 **SEE ALSO**
 38196 `isalnum()`, `isalpha()`, `isblank()`, `iscntrl()`, `isdigit()`, `isgraph()`, `islower()`, `isprint()`, `ispunct()`, `isspace()`,
 38197 `isxdigit()`, `setlocale()`, `uselocale()`
 38198 XBD Chapter 7 (on page 121), `<ctype.h>`, `<locale.h>`

38199 **CHANGE HISTORY**
 38200 First released in Issue 1. Derived from Issue 1 of the SVID.

38201 **Issue 6**
 38202 The normative text is updated to avoid use of the term “must” for application requirements.

38203
38204
38205**Issue 7**

The *isupper_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38206 **NAME**

38207 iswalnum, iswalnum_l — test for an alphanumeric wide-character code

38208 **SYNOPSIS**

38209 #include <wctype.h>

38210 int iswalnum(wint_t wc);

38211 CX int iswalnum_l(wint_t wc, locale_t locale);

38212 **DESCRIPTION**38213 CX For *iswalnum()*: The functionality described on this reference page is aligned with the ISO C
38214 standard. Any conflict between the requirements described here and the ISO C standard is
38215 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38216 CX The *iswalnum()* and *iswalnum_l()* functions shall test whether *wc* is a wide-character code
38217 representing a character of class **alpha** or **digit** in the current locale of the process, or in the
38218 locale represented by *locale*, respectively; see XBD Chapter 7 (on page 121).38219 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38220 code corresponding to a valid character in the current locale, or equal to the value of the macro
38221 WEOF. If the argument has any other value, the behavior is undefined.38222 **RETURN VALUE**38223 CX The *iswalnum()* and *iswalnum_l()* functions shall return non-zero if *wc* is an alphanumeric
38224 wide-character code; otherwise, they shall return 0.38225 **ERRORS**38226 The *iswalnum_l()* function may fail if:38227 CX [EINVAL] *locale* is not a valid locale object handle.38228 **EXAMPLES**

38229 None.

38230 **APPLICATION USAGE**38231 To ensure applications portability, especially across natural languages, only these functions and
38232 the functions in the reference pages listed in the SEE ALSO section should be used for character
38233 classification.38234 **RATIONALE**

38235 None.

38236 **FUTURE DIRECTIONS**

38237 None.

38238 **SEE ALSO**38239 *iswalphabet()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
38240 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38241 XBD Chapter 7 (on page 121), <locale.h>, <stdio.h>, <wctype.h>

38242 **CHANGE HISTORY**

38243 First released as a World-wide Portability Interface in Issue 4.

38244 **Issue 5**38245 The following change has been made in this version for alignment with
38246 ISO/IEC 9899:1990/Amendment 1:1995 (E):

iswalnum()

38247

38248

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38249

Issue 6

38250

The normative text is updated to avoid use of the term “must” for application requirements.

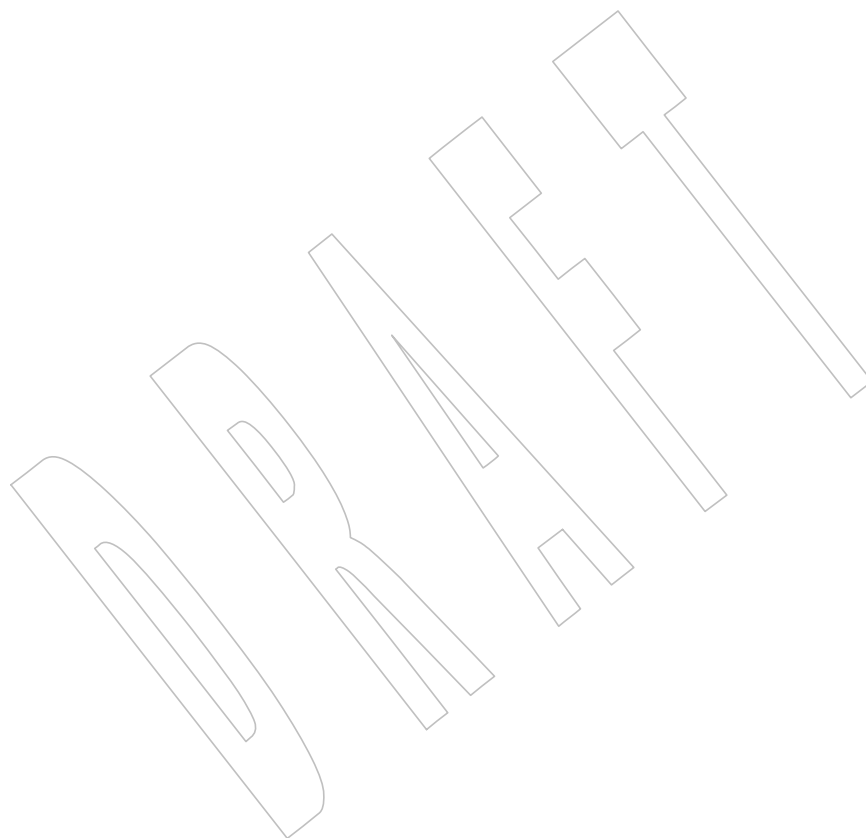
38251

Issue 7

38252

The `iswalnum_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38253



38254 **NAME**

38255 iswalpha, iswalpha_l — test for an alphabetic wide-character code

38256 **SYNOPSIS**

38257 #include <wctype.h>

38258 int iswalpha(wint_t wc);

38259 CX int iswalpha_l(wint_t wc, locale_t locale);

38260 **DESCRIPTION**38261 CX For *iswalpha()*: The functionality described on this reference page is aligned with the ISO C
38262 standard. Any conflict between the requirements described here and the ISO C standard is
38263 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38264 CX The *iswalpha()* and *iswalpha_l()* functions shall test whether *wc* is a wide-character code
38265 representing a character of class **alpha** in the current locale of the process, or in the locale
38266 represented by *locale*, respectively; see XBD Chapter 7 (on page 121).38267 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38268 code corresponding to a valid character in the current locale, or equal to the value of the macro
38269 WEOF. If the argument has any other value, the behavior is undefined.38270 **RETURN VALUE**38271 CX The *iswalpha()* and *iswalpha_l()* functions shall return non-zero if *wc* is an alphabetic wide-
38272 character code; otherwise, they shall return 0.38273 **ERRORS**38274 The *iswalpha_l()* function may fail if:38275 CX [EINVAL] *locale* is not a valid locale object handle.38276 **EXAMPLES**

38277 None.

38278 **APPLICATION USAGE**38279 To ensure applications portability, especially across natural languages, only these functions and
38280 the functions in the reference pages listed in the SEE ALSO section should be used for character
38281 classification.38282 **RATIONALE**

38283 None.

38284 **FUTURE DIRECTIONS**

38285 None.

38286 **SEE ALSO**38287 *iswalnum()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*,
38288 *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38289 XBD Chapter 7 (on page 121), <locale.h>, <stdio.h>, <wctype.h>

38290 **CHANGE HISTORY**

38291 First released in Issue 4.

38292 **Issue 5**38293 The following change has been made in this version for alignment with
38294 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38295

38296

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38297

Issue 6

38298

The normative text is updated to avoid use of the term “must” for application requirements.

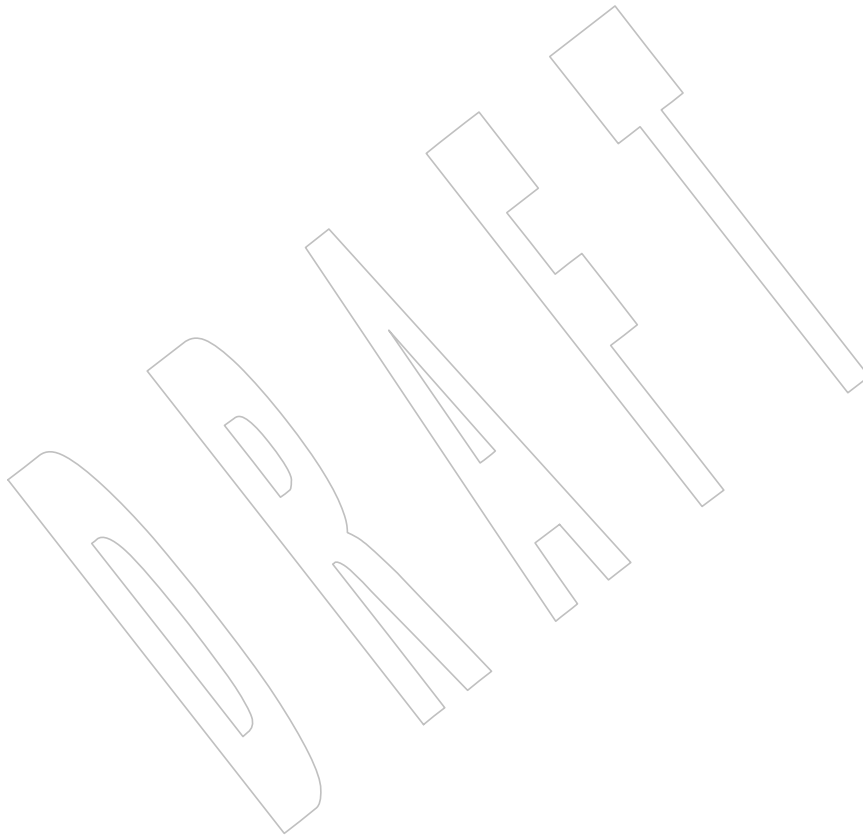
38299

Issue 7

38300

38301

The `iswalpha_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38302 **NAME**

38303 iswblank, iswblank_l — test for a blank wide-character code

38304 **SYNOPSIS**

38305 #include <wctype.h>

38306 int iswblank(wint_t wc);

38307 CX int iswblank_l(wint_t wc, locale_t locale);

38308 **DESCRIPTION**38309 CX For *iswblank()*: The functionality described on this reference page is aligned with the ISO C
38310 standard. Any conflict between the requirements described here and the ISO C standard is
38311 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38312 CX The *iswblank()* and *iswblank_l()* functions shall test whether *wc* is a wide-character code
38313 representing a character of class **blank** in the current locale of the process, or in the locale
38314 represented by *locale*, respectively; see XBD Chapter 7 (on page 121).38315 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38316 code corresponding to a valid character in the current locale, or equal to the value of the macro
38317 WEOF. If the argument has any other value, the behavior is undefined.38318 **RETURN VALUE**38319 CX The *iswblank()* and *iswblank_l()* functions shall return non-zero if *wc* is a blank wide-character
38320 code; otherwise, they shall return 0.38321 **ERRORS**38322 The *iswblank_l()* function may fail if:38323 CX [EINVAL] *locale* is not a valid locale object handle.38324 **EXAMPLES**

38325 None.

38326 **APPLICATION USAGE**38327 To ensure applications portability, especially across natural languages, only these functions and
38328 the functions in the reference pages listed in the SEE ALSO section should be used for character
38329 classification.38330 **RATIONALE**

38331 None.

38332 **FUTURE DIRECTIONS**

38333 None.

38334 **SEE ALSO**38335 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
38336 *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*, *setlocale()*, *uselocale()*

38337 XBD Chapter 7 (on page 121), <locale.h>, <stdio.h>, <wctype.h>

38338 **CHANGE HISTORY**

38339 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

38340 **Issue 7**38341 The *iswblank_l()* function is added from The Open Group Technical Standard, 2006, Extended
38342 API Set Part 4.

38343 **NAME**
 38344 `iswcntrl, iswcntrl_l` — test for a control wide-character code

38345 **SYNOPSIS**
 38346 `#include <wctype.h>`
 38347 `int iswcntrl(wint_t wc);`
 38348 CX `int iswcntrl_l(wint_t wc, locale_t locale);`

38349 **DESCRIPTION**
 38350 CX For `iswcntrl()`: The functionality described on this reference page is aligned with the ISO C
 38351 standard. Any conflict between the requirements described here and the ISO C standard is
 38352 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38353 CX The `iswcntrl()` and `iswcntrl_l()` functions shall test whether `wc` is a wide-character code
 38354 CX representing a character of class **cntrl** in the current locale of the process, or in the locale
 38355 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).

38356 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 38357 code corresponding to a valid character in the current locale, or equal to the value of the macro
 38358 WEOF. If the argument has any other value, the behavior is undefined.

38359 **RETURN VALUE**
 38360 CX The `iswcntrl()` and `iswcntrl_l()` functions shall return non-zero if `wc` is a control wide-character
 38361 code; otherwise, they shall return 0.

38362 **ERRORS**
 38363 The `iswcntrl_l()` function may fail if:
 38364 CX **[EINVAL]** `locale` is not a valid locale object handle.

38365 **EXAMPLES**
 38366 None.

38367 **APPLICATION USAGE**
 38368 To ensure applications portability, especially across natural languages, only these functions and
 38369 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38370 classification.

38371 **RATIONALE**
 38372 None.

38373 **FUTURE DIRECTIONS**
 38374 None.

38375 **SEE ALSO**
 38376 `iswalnum()`, `iswalpha()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,
 38377 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`
 38378 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`

38379 **CHANGE HISTORY**
 38380 First released in Issue 4.

38381 **Issue 5**
 38382 The following change has been made in this version for alignment with
 38383 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38384

38385

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38386

Issue 6

38387

The normative text is updated to avoid use of the term “must” for application requirements.

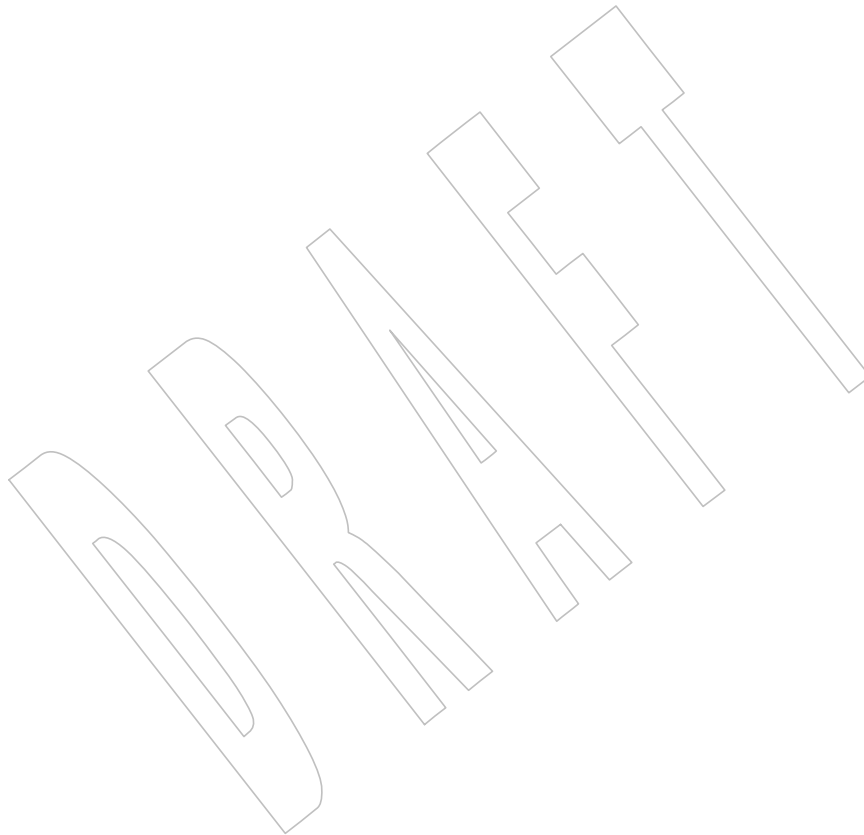
38388

Issue 7

38389

The `iswcntrl_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38390



38391 **NAME**
 38392 `iswctype`, `iswctype_l` — test character for a specified class

SYNOPSIS

```
38394 #include <wctype.h>
38395
38395 int iswctype(wint_t wc, wctype_t charclass);
38396 CX int iswctype_l(wint_t wc, wctype_t charclass,
38397 locale_t locale);
```

DESCRIPTION

38398
 38399 CX For `iswctype()`: The functionality described on this reference page is aligned with the ISO C
 38400 standard. Any conflict between the requirements described here and the ISO C standard is
 38401 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38402 CX The `iswctype()` and `iswctype_l()` functions shall determine whether the wide-character code `wc`
 38403 has the character class `charclass`, returning true or false. The `iswctype()` and `iswctype_l()`
 38404 functions are defined on WEOF and wide-character codes corresponding to the valid character
 38405 encodings in the current locale, or in the locale represented by `locale`, respectively. If the `wc`
 38406 argument is not in the domain of the function, the result is undefined. If the value of `charclass` is
 38407 invalid (that is, not obtained by a call to `wctype()` or `charclass` is invalidated by a subsequent call
 38408 to `setlocale()` that has affected category `LC_CTYPE`) the result is unspecified.

RETURN VALUE

38409 CX The `iswctype()` and `iswctype_l()` functions shall return non-zero (true) if and only if `wc` has the
 38410 property described by `charclass`. If `charclass` is 0, these functions shall return 0.

ERRORS

38412 The `iswctype_l()` function may fail if:

38413
 38414 CX [EINVAL] `locale` is not a valid locale object handle.

EXAMPLES**Testing for a Valid Character**

```
38416 #include <wctype.h>
38417 ...
38418 int yes_or_no;
38419 wint_t wc;
38420 wctype_t valid_class;
38421 ...
38422 if ((valid_class=wctype("vowel")) == (wctype_t)0)
38423     /* Invalid character class. */
38424     yes_or_no=iswctype(wc,valid_class);
```

APPLICATION USAGE

38426 The twelve strings "alnum", "alpha", "blank", "cntrl", "digit", "graph", "lower",
 38427 "print", "punct", "space", "upper", and "xdigit" are reserved for the standard
 38428 character classes. In the table below, the functions in the left column are equivalent to the
 38429 functions in the right column.

38431	<code>iswalnum(wc)</code>	<code>iswctype(wc, wctype("alnum"))</code>
38432	<code>iswalnum_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alnum"), locale)</code>
38433	<code>iswalpha(wc)</code>	<code>iswctype(wc, wctype("alpha"))</code>
38434	<code>iswalpha_l(wc, locale)</code>	<code>iswctype_l(wc, wctype("alpha"), locale)</code>

```

38435      iswblank(wc)          iswctype(wc, wctype("blank"))
38436      iswblank_l(wc, locale) iswctype_l(wc, wctype("blank"), locale)
38437      iswcntrl(wc)         iswctype(wc, wctype("cntrl"))
38438      iswcntrl_l(wc, locale) iswctype_l(wc, wctype("cntrl"), locale)
38439      iswdigit(wc)         iswctype(wc, wctype("digit"))
38440      iswdigit_l(wc, locale) iswctype_l(wc, wctype("digit"), locale)
38441      iswgraph(wc)        iswctype(wc, wctype("graph"))
38442      iswgraph_l(wc, locale) iswctype_l(wc, wctype("graph"), locale)
38443      iswlower(wc)        iswctype(wc, wctype("lower"))
38444      iswlower_l(wc, locale) iswctype_l(wc, wctype("lower"), locale)
38445      iswprint(wc)        iswctype(wc, wctype("print"))
38446      iswprint_l(wc, locale) iswctype_l(wc, wctype("print"), locale)
38447      iswpunct(wc)        iswctype(wc, wctype("punct"))
38448      iswpunct_l(wc, locale) iswctype_l(wc, wctype("punct"), locale)
38449      iswspace(wc)        iswctype(wc, wctype("space"))
38450      iswspace_l(wc, locale) iswctype_l(wc, wctype("space"), locale)
38451      iswupper(wc)        iswctype(wc, wctype("upper"))
38452      iswupper_l(wc, locale) iswctype_l(wc, wctype("upper"), locale)
38453      iswxdigit(wc)       iswctype(wc, wctype("xdigit"))
38454      iswxdigit_l(wc, locale) iswctype_l(wc, wctype("xdigit"), locale)

```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum(), iswalpha(), iswcntrl(), iswdigit(), iswgraph(), iswlower(), iswprint(), iswpunct(), iswspace(), iswupper(), iswxdigit(), setlocale(),uselocale(), wctype()

XBD **<locale.h>**, **<wctype.h>**

CHANGE HISTORY

First released as World-wide Portability Interfaces in Issue 4.

Issue 5

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

Issue 6

The behavior of $n=0$ is now described.

An example is added.

A new function, *iswblank()*, is added to the list in the APPLICATION USAGE.

Issue 7

The *iswctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38477 **NAME**38478 `iswdigit, iswdigit_l` — test for a decimal digit wide-character code38479 **SYNOPSIS**38480 `#include <wctype.h>`38481 `int iswdigit(wint_t wc);`38482 CX `int iswdigit_l(wint_t wc, locale_t locale);`38483 **DESCRIPTION**38484 CX For `iswdigit()`: The functionality described on this reference page is aligned with the ISO C
38485 standard. Any conflict between the requirements described here and the ISO C standard is
38486 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38487 CX The `iswdigit()` and `iswdigit_l()` functions shall test whether `wc` is a wide-character code
38488 representing a character of class **digit** in the current locale of the process, or in the locale
38489 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).38490 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38491 code corresponding to a valid character in the current locale, or equal to the value of the macro
38492 WEOF. If the argument has any other value, the behavior is undefined.38493 **RETURN VALUE**38494 CX The `iswdigit()` and `iswdigit_l()` functions shall return non-zero if `wc` is a decimal digit wide-
38495 character code; otherwise, they shall return 0.38496 **ERRORS**38497 The `iswdigit_l()` function may fail if:38498 CX **[EINVAL]** `locale` is not a valid locale object handle.38499 **EXAMPLES**

38500 None.

38501 **APPLICATION USAGE**38502 To ensure applications portability, especially across natural languages, only these functions and
38503 the functions in the reference pages listed in the SEE ALSO section should be used for character
38504 classification.38505 **RATIONALE**

38506 None.

38507 **FUTURE DIRECTIONS**

38508 None.

38509 **SEE ALSO**38510 `iswalnum()`, `iswalpunct()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswgraph()`, `iswlower()`, `iswprint()`, `iswpunct()`,
38511 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`38512 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`38513 **CHANGE HISTORY**

38514 First released in Issue 4.

38515 **Issue 5**38516 The following change has been made in this version for alignment with
38517 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38518

38519

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38520

Issue 6

38521

The normative text is updated to avoid use of the term “must” for application requirements.

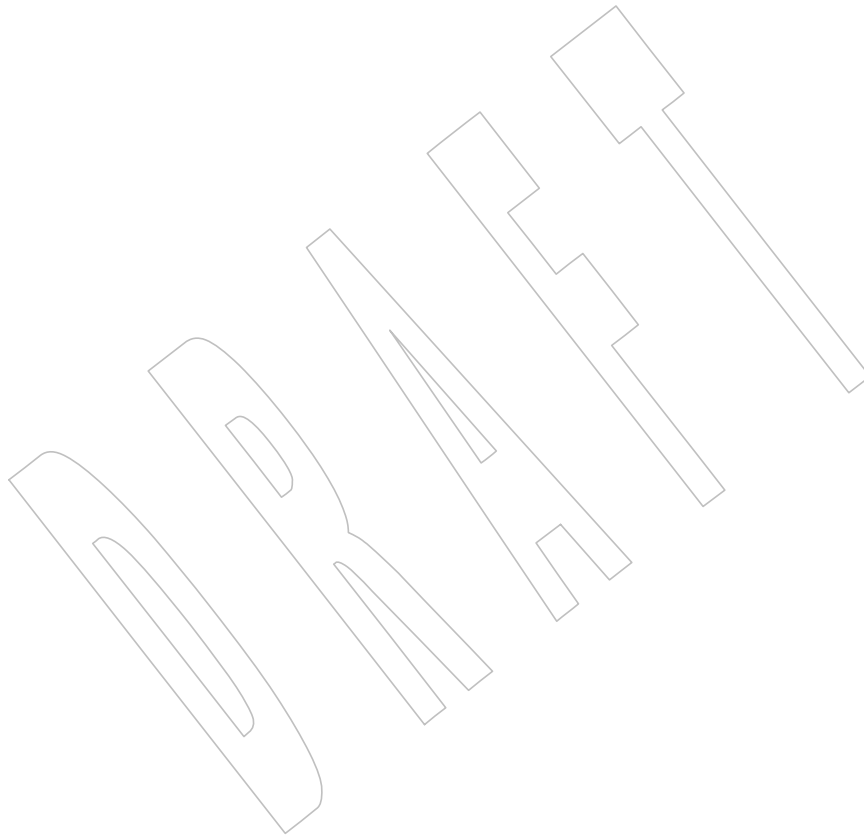
38522

Issue 7

38523

The `iswdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38524



38525 **NAME**38526 `iswgraph, iswgraph_l` — test for a visible wide-character code38527 **SYNOPSIS**38528 `#include <wctype.h>`38529 `int iswgraph(wint_t wc);`38530 CX `int iswgraph_l(wint_t wc, locale_t locale);`38531 **DESCRIPTION**38532 CX For `iswgraph()`: The functionality described on this reference page is aligned with the ISO C
38533 standard. Any conflict between the requirements described here and the ISO C standard is
38534 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38535 CX The `iswgraph()` and `iswgraph_l()` functions shall test whether `wc` is a wide-character code
38536 representing a character of class **graph** in the current locale of the process, or in the locale
38537 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).38538 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38539 code corresponding to a valid character in the current locale, or equal to the value of the macro
38540 WEOF. If the argument has any other value, the behavior is undefined.38541 **RETURN VALUE**38542 CX The `iswgraph()` and `iswgraph_l()` functions shall return non-zero if `wc` is a wide-character code
38543 with a visible representation; otherwise, they shall return 0.38544 **ERRORS**38545 The `iswgraph_l()` function may fail if:38546 CX **[EINVAL]** `locale` is not a valid locale object handle.38547 **EXAMPLES**

38548 None.

38549 **APPLICATION USAGE**38550 To ensure applications portability, especially across natural languages, only these functions and
38551 the functions in the reference pages listed in the SEE ALSO section should be used for character
38552 classification.38553 **RATIONALE**

38554 None.

38555 **FUTURE DIRECTIONS**

38556 None.

38557 **SEE ALSO**38558 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswlower()`, `iswprint()`, `iswpunct()`,
38559 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`38560 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`38561 **CHANGE HISTORY**

38562 First released in Issue 4.

38563 **Issue 5**38564 The following change has been made in this version for alignment with
38565 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38566

38567

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38568

Issue 6

38569

The normative text is updated to avoid use of the term “must” for application requirements.

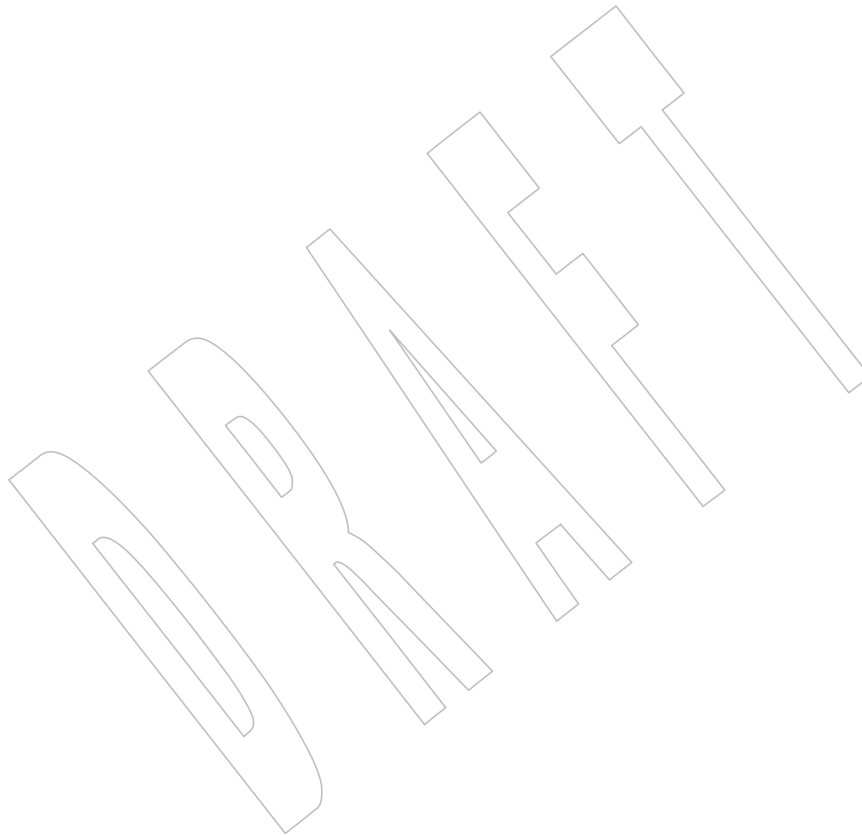
38570

Issue 7

38571

The `iswgraph_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38572



38573 **NAME**
 38574 `iswlower, iswlower_l` — test for a lowercase letter wide-character code

38575 **SYNOPSIS**
 38576 `#include <wctype.h>`
 38577 `int iswlower(wint_t wc);`
 38578 CX `int iswlower_l(wint_t wc, locale_t locale);`

38579 **DESCRIPTION**
 38580 CX For `iswlower()`: The functionality described on this reference page is aligned with the ISO C
 38581 standard. Any conflict between the requirements described here and the ISO C standard is
 38582 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38583 CX The `iswlower()` and `iswlower_l()` functions shall test whether `wc` is a wide-character code
 38584 CX representing a character of class **lower** in the current locale of the process, or in the locale
 38585 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).

38586 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 38587 code corresponding to a valid character in the current locale, or equal to the value of the macro
 38588 WEOF. If the argument has any other value, the behavior is undefined.

38589 **RETURN VALUE**
 38590 CX The `iswlower()` and `iswlower_l()` functions shall return non-zero if `wc` is a lowercase letter wide-
 38591 character code; otherwise, they shall return 0.

38592 **ERRORS**
 38593 The `iswlower_l()` function may fail if:
 38594 CX **[EINVAL]** `locale` is not a valid locale object handle.

38595 **EXAMPLES**
 38596 None.

38597 **APPLICATION USAGE**
 38598 To ensure applications portability, especially across natural languages, only these functions and
 38599 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38600 classification.

38601 **RATIONALE**
 38602 None.

38603 **FUTURE DIRECTIONS**
 38604 None.

38605 **SEE ALSO**
 38606 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswprint()`, `iswpunct()`,
 38607 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()` (on page 2112) 1
 38608 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`

38609 **CHANGE HISTORY**
 38610 First released in Issue 4.

38611 **Issue 5**
 38612 The following change has been made in this version for alignment with
 38613 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38614

38615

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38616

Issue 6

38617

The normative text is updated to avoid use of the term “must” for application requirements.

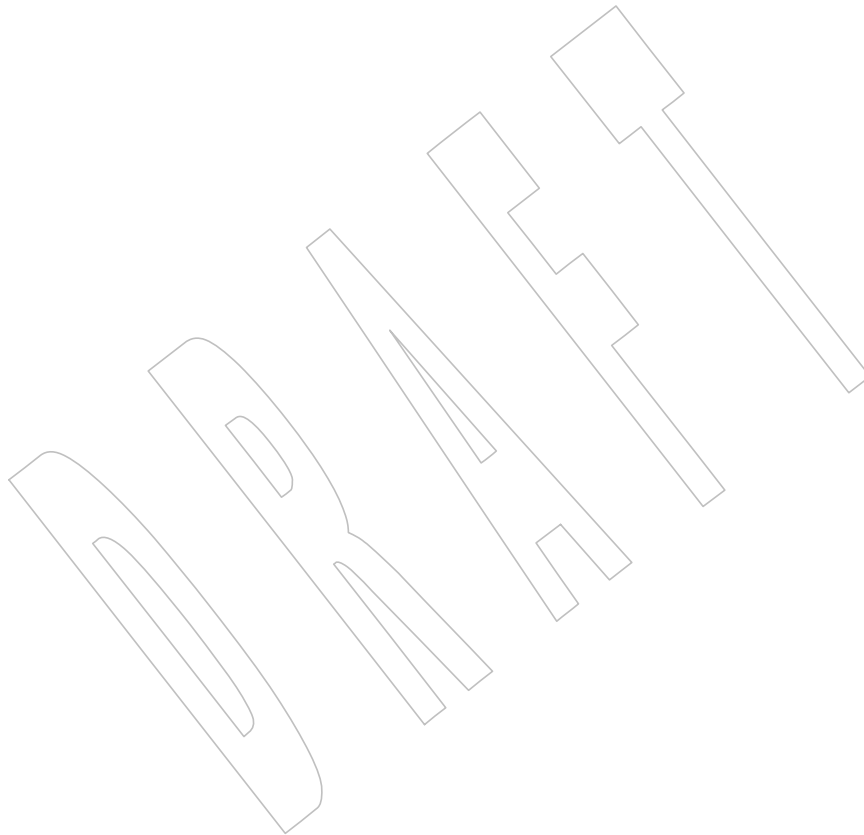
38618

Issue 7

38619

The `iswlower_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38620



38621 **NAME**
 38622 `iswprint, iswprint_l` — test for a printable wide-character code

38623 **SYNOPSIS**

38624 `#include <wctype.h>`
 38625 `int iswprint(wint_t wc);`
 38626 CX `int iswprint_l(wint_t wc, locale_t locale);`

38627 **DESCRIPTION**

38628 CX For `iswprint()`: The functionality described on this reference page is aligned with the ISO C
 38629 standard. Any conflict between the requirements described here and the ISO C standard is
 38630 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38631 CX The `iswprint()` and `iswprint_l()` functions shall test whether `wc` is a wide-character code
 38632 representing a character of class **print** in the current locale of the process, or in the locale
 38633 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).

38634 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 38635 code corresponding to a valid character in the current locale, or equal to the value of the macro
 38636 WEOF. If the argument has any other value, the behavior is undefined.

38637 **RETURN VALUE**

38638 CX The `iswprint()` and `iswprint_l()` functions shall return non-zero if `wc` is a printable wide-
 38639 character code; otherwise, they shall return 0.

38640 **ERRORS**

38641 The `iswprint_l()` function may fail if:

38642 CX **[EINVAL]** `locale` is not a valid locale object handle.

38643 **EXAMPLES**

38644 None.

38645 **APPLICATION USAGE**

38646 To ensure applications portability, especially across natural languages, only these functions and
 38647 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38648 classification.

38649 **RATIONALE**

38650 None.

38651 **FUTURE DIRECTIONS**

38652 None.

38653 **SEE ALSO**

38654 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswpunct()`,
 38655 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`

38656 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`

38657 **CHANGE HISTORY**

38658 First released in Issue 4.

38659 **Issue 5**

38660 The following change has been made in this version for alignment with
 38661 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38662

38663

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38664

Issue 6

38665

The normative text is updated to avoid use of the term “must” for application requirements.

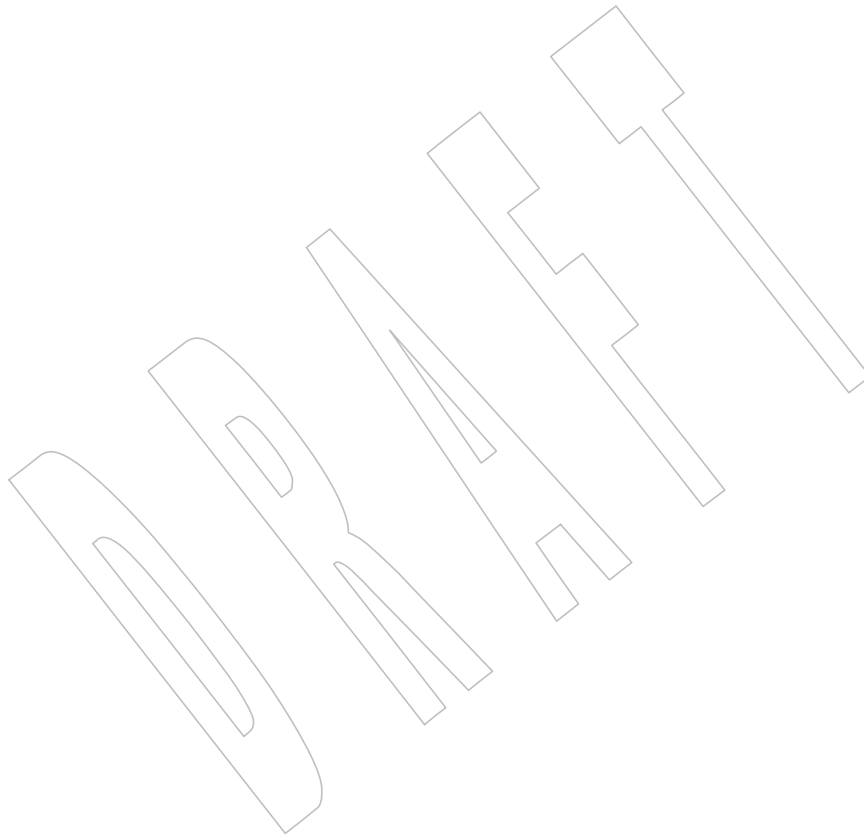
38666

Issue 7

38667

The `iswprint_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38668



38669 **NAME**38670 `iswpunct`, `iswpunct_l` — test for a punctuation wide-character code38671 **SYNOPSIS**38672 `#include <wctype.h>`38673 `int iswpunct(wint_t wc);`38674 CX `int iswpunct_l(wint_t wc, locale_t locale);`38675 **DESCRIPTION**38676 CX For `iswpunct()`: The functionality described on this reference page is aligned with the ISO C
38677 standard. Any conflict between the requirements described here and the ISO C standard is
38678 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38679 CX The `iswpunct()` and `iswpunct_l()` functions shall test whether `wc` is a wide-character code
38680 CX representing a character of class **punct** in the current locale of the process, or in the locale
38681 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).38682 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38683 code corresponding to a valid character in the current locale, or equal to the value of the macro
38684 WEOF. If the argument has any other value, the behavior is undefined.38685 **RETURN VALUE**38686 CX The `iswpunct()` and `iswpunct_l()` functions shall return non-zero if `wc` is a punctuation wide-
38687 character code; otherwise, they shall return 0.38688 **ERRORS**38689 The `iswpunct_l()` function may fail if:38690 CX **[EINVAL]** `locale` is not a valid locale object handle.38691 **EXAMPLES**

38692 None.

38693 **APPLICATION USAGE**38694 To ensure applications portability, especially across natural languages, only these functions and
38695 the functions in the reference pages listed in the SEE ALSO section should be used for character
38696 classification.38697 **RATIONALE**

38698 None.

38699 **FUTURE DIRECTIONS**

38700 None.

38701 **SEE ALSO**38702 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`,
38703 `iswspace()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`38704 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`38705 **CHANGE HISTORY**

38706 First released in Issue 4.

38707 **Issue 5**38708 The following change has been made in this version for alignment with
38709 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38710

38711

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38712

Issue 6

38713

The normative text is updated to avoid use of the term “must” for application requirements.

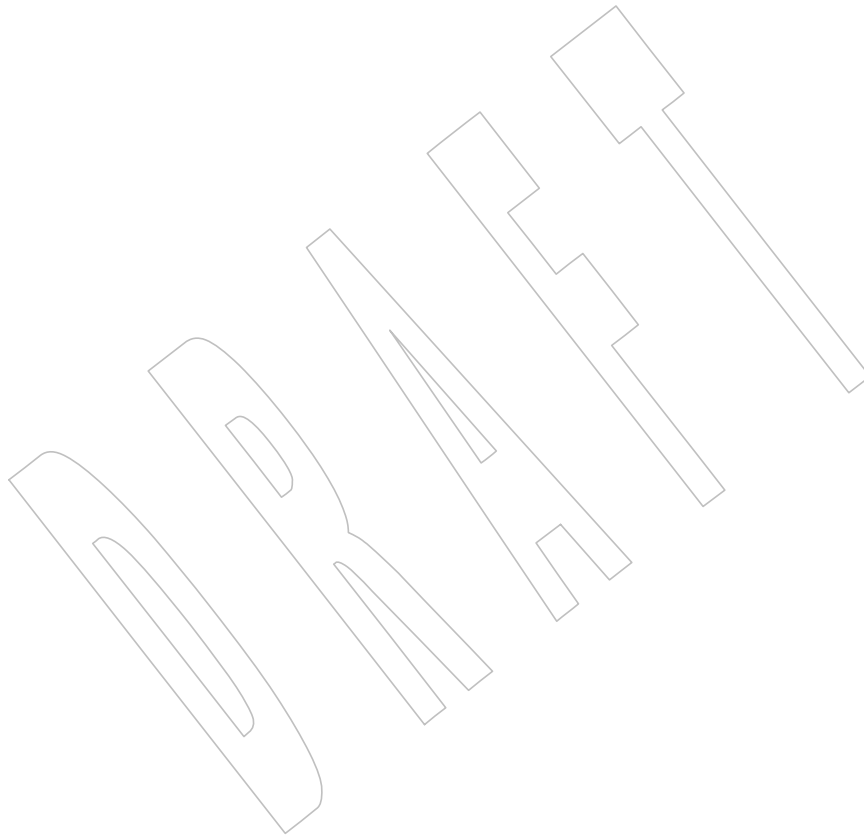
38714

Issue 7

38715

38716

The `iswpunct_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38717 **NAME**38718 `iswspace`, `iswspace_l` — test for a white-space wide-character code38719 **SYNOPSIS**38720 `#include <wctype.h>`38721 `int iswspace(wint_t wc);`38722 CX `int iswspace_l(wint_t wc, locale_t locale);`38723 **DESCRIPTION**38724 CX For `iswspace()`: The functionality described on this reference page is aligned with the ISO C
38725 standard. Any conflict between the requirements described here and the ISO C standard is
38726 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38727 CX The `iswspace()` and `iswspace_l()` functions shall test whether `wc` is a wide-character code
38728 representing a character of class **space** in the current locale of the process, or in the locale
38729 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).38730 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38731 code corresponding to a valid character in the current locale, or equal to the value of the macro
38732 WEOF. If the argument has any other value, the behavior is undefined.38733 **RETURN VALUE**38734 CX The `iswspace()` and `iswspace_l()` functions shall return non-zero if `wc` is a white-space wide-
38735 character code; otherwise, they shall return 0.38736 **ERRORS**38737 The `iswspace_l()` function may fail if:38738 CX **[EINVAL]** `locale` is not a valid locale object handle.38739 **EXAMPLES**

38740 None.

38741 **APPLICATION USAGE**38742 To ensure applications portability, especially across natural languages, only these functions and
38743 the functions in the reference pages listed in the SEE ALSO section should be used for character
38744 classification.38745 **RATIONALE**

38746 None.

38747 **FUTURE DIRECTIONS**

38748 None.

38749 **SEE ALSO**38750 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`,
38751 `iswpunct()`, `iswupper()`, `iswxdigit()`, `setlocale()`, `uselocale()`38752 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`38753 **CHANGE HISTORY**

38754 First released in Issue 4.

38755 **Issue 5**38756 The following change has been made in this version for alignment with
38757 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38758

38759

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38760

Issue 6

38761

The normative text is updated to avoid use of the term “must” for application requirements.

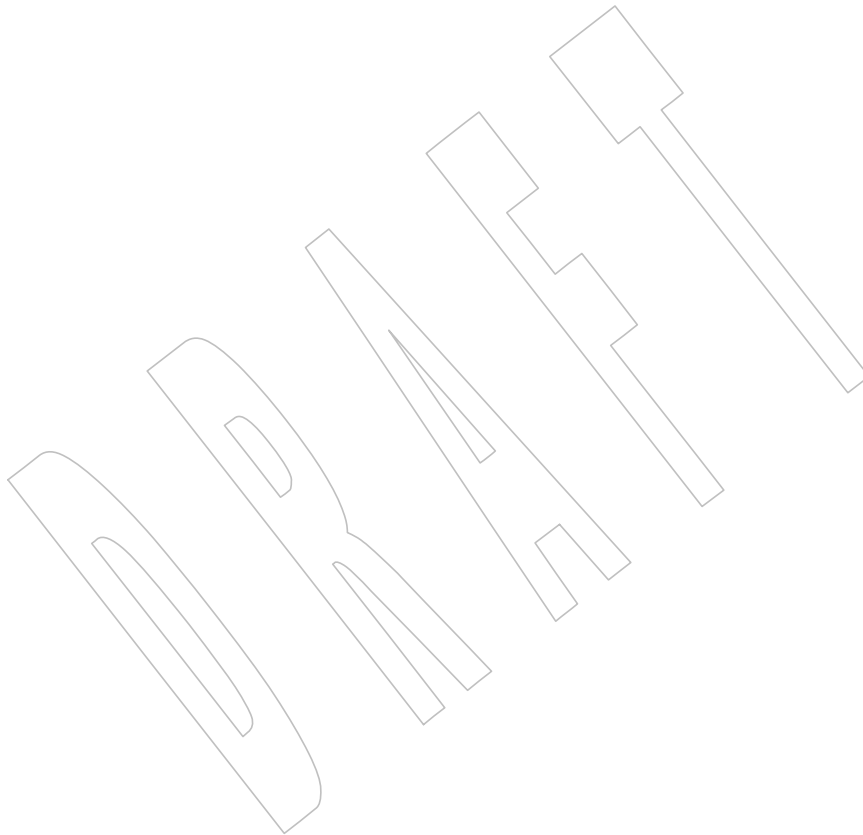
38762

Issue 7

38763

38764

The `iswspace_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38765 **NAME**
 38766 `iswupper, iswupper_l` — test for an uppercase letter wide-character code

38767 **SYNOPSIS**
 38768 `#include <wctype.h>`
 38769 `int iswupper(wint_t wc);`
 38770 CX `int iswupper_l(wint_t wc, locale_t locale);`

38771 **DESCRIPTION**
 38772 CX For `iswupper()`: The functionality described on this reference page is aligned with the ISO C
 38773 standard. Any conflict between the requirements described here and the ISO C standard is
 38774 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38775 CX The `iswupper()` and `iswupper_l()` functions shall test whether `wc` is a wide-character code
 38776 representing a character of class **upper** in the current locale of the process, or in the locale
 38777 represented by `locale`, respectively; see XBD Chapter 7 (on page 121).

38778 The `wc` argument is a **wint_t**, the value of which the application shall ensure is a wide-character
 38779 code corresponding to a valid character in the current locale, or equal to the value of the macro
 38780 WEOF. If the argument has any other value, the behavior is undefined.

38781 **RETURN VALUE**
 38782 CX The `iswupper()` and `iswupper_l()` functions shall return non-zero if `wc` is an uppercase letter
 38783 wide-character code; otherwise, they shall return 0.

38784 **ERRORS**
 38785 The `iswupper_l()` function may fail if:

38786 CX **[EINVAL]** `locale` is not a valid locale object handle.

38787 **EXAMPLES**
 38788 None.

38789 **APPLICATION USAGE**
 38790 To ensure applications portability, especially across natural languages, only these functions and
 38791 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38792 classification.

38793 **RATIONALE**
 38794 None.

38795 **FUTURE DIRECTIONS**
 38796 None.

38797 **SEE ALSO**
 38798 `iswalnum()`, `iswalpha()`, `iswcntrl()`, `iswctype()`, `iswdigit()`, `iswgraph()`, `iswlower()`, `iswprint()`,
 38799 `iswpunct()`, `iswspace()`, `iswxdigit()`, `setlocale()`, `uselocale()`
 38800 XBD Chapter 7 (on page 121), `<locale.h>`, `<wctype.h>`

38801 **CHANGE HISTORY**
 38802 First released in Issue 4.

38803 **Issue 5**
 38804 The following change has been made in this version for alignment with
 38805 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38806

38807

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38808

Issue 6

38809

The normative text is updated to avoid use of the term “must” for application requirements.

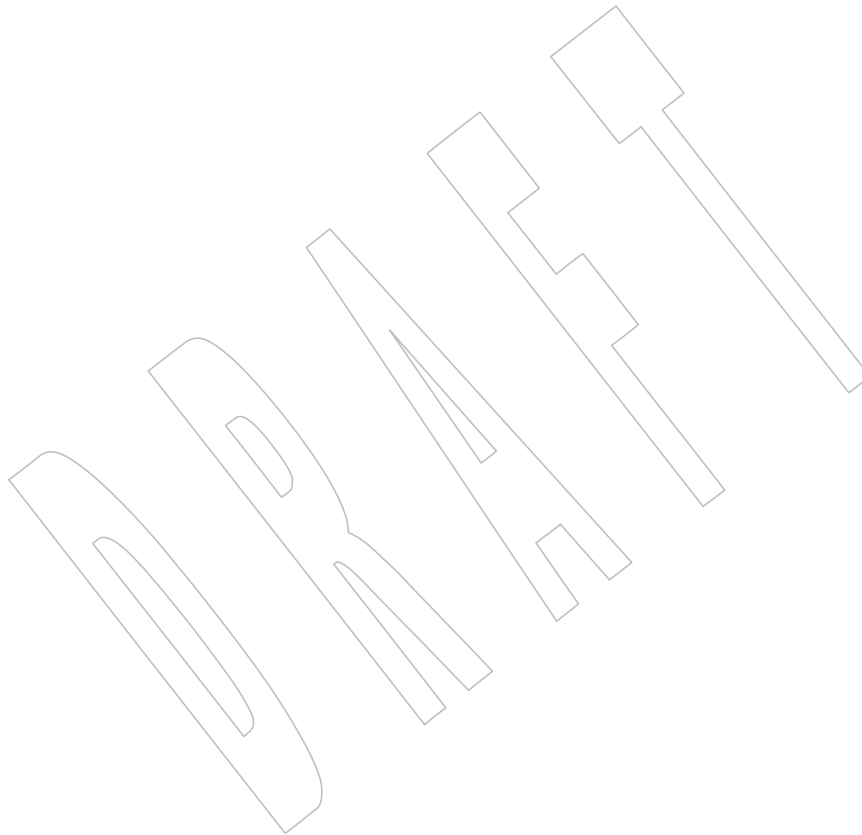
38810

Issue 7

38811

The `iswupper_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38812



38813 **NAME**

38814 iswxdigit, iswxdigit_l — test for a hexadecimal digit wide-character code

38815 **SYNOPSIS**

38816 #include <wctype.h>

38817 int iswxdigit(wint_t wc);

38818 CX int iswxdigit_l(wint_t wc, locale_t locale);

38819 **DESCRIPTION**38820 CX For *iswxdigit()*: The functionality described on this reference page is aligned with the ISO C
38821 standard. Any conflict between the requirements described here and the ISO C standard is
38822 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.38823 CX The *iswxdigit()* and *iswxdigit_l()* functions shall test whether *wc* is a wide-character code
38824 CX representing a character of class **xdigit** in the current locale of the process, or in the locale
38825 represented by *locale*, respectively; see XBD Chapter 7 (on page 121).38826 The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character
38827 code corresponding to a valid character in the current locale, or equal to the value of the macro
38828 WEOF. If the argument has any other value, the behavior is undefined.38829 **RETURN VALUE**38830 CX The *iswxdigit()* and *iswxdigit_l()* functions shall return non-zero if *wc* is a hexadecimal digit
38831 wide-character code; otherwise, they shall return 0.38832 **ERRORS**38833 The *iswxdigit_l()* function may fail if:38834 CX [EINVAL] *locale* is not a valid locale object handle.38835 **EXAMPLES**

38836 None.

38837 **APPLICATION USAGE**38838 To ensure applications portability, especially across natural languages, only these functions and
38839 the functions in the reference pages listed in the SEE ALSO section should be used for character
38840 classification.38841 **RATIONALE**

38842 None.

38843 **FUTURE DIRECTIONS**

38844 None.

38845 **SEE ALSO**38846 *iswalnum()*, *iswalpha()*, *iswcntrl()*, *iswctype()*, *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*,
38847 *iswpunct()*, *iswspace()*, *iswupper()*, *setlocale()*, *uselocale()*

38848 XBD Chapter 7 (on page 121), <locale.h>, <wctype.h>

38849 **CHANGE HISTORY**

38850 First released in Issue 4.

38851 **Issue 5**38852 The following change has been made in this version for alignment with
38853 ISO/IEC 9899:1990/Amendment 1:1995 (E):

38854

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

38855

38856

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

38857

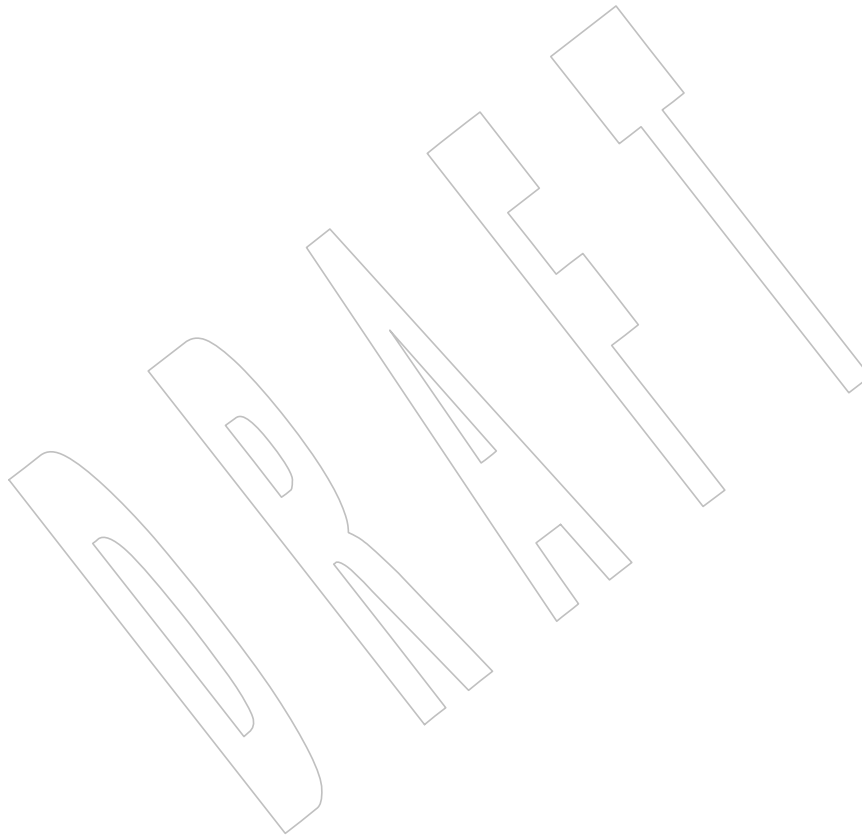
38858

Issue 7

The `iswxdigit_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

38859

38860



38861 **NAME**
 38862 isxdigit, isxdigit_l — test for a hexadecimal digit

38863 **SYNOPSIS**
 38864 #include <ctype.h>
 38865 int isxdigit(int c);
 38866 CX int isxdigit_l(int c, locale_t locale);

38867 **DESCRIPTION**
 38868 CX For *isxdigit()*: The functionality described on this reference page is aligned with the ISO C
 38869 standard. Any conflict between the requirements described here and the ISO C standard is
 38870 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

38871 CX The *isxdigit()* and *isxdigit_l()* functions shall test whether *c* is a character of class **xdigit** in the
 38872 current locale of the process, or in the locale represented by *locale*, respectively; see XBD
 38873 Chapter 7 (on page 121).

38874 The *c* argument is an **int**, the value of which the application shall ensure is a character
 38875 representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has
 38876 any other value, the behavior is undefined.

38877 **RETURN VALUE**
 38878 CX The *isxdigit()* and *isxdigit_l()* functions shall return non-zero if *c* is a hexadecimal digit;
 38879 otherwise, they shall return 0.

38880 **ERRORS**
 38881 The *isxdigit_l()* function may fail if:

38882 CX [EINVAL] *locale* is not a valid locale object handle.

38883 **EXAMPLES**
 38884 None.

38885 **APPLICATION USAGE**
 38886 To ensure applications portability, especially across natural languages, only these functions and
 38887 the functions in the reference pages listed in the SEE ALSO section should be used for character
 38888 classification.

38889 **RATIONALE**
 38890 None.

38891 **FUTURE DIRECTIONS**
 38892 None.

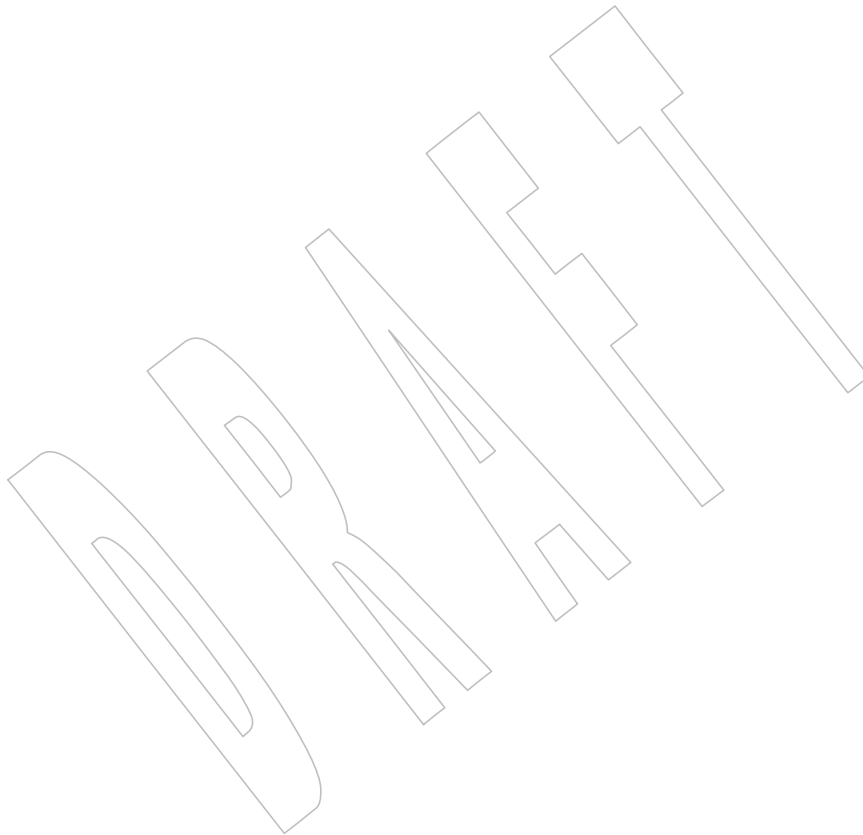
38893 **SEE ALSO**
 38894 *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*, *isprint()*, *ispunct()*, *isspace()*,
 38895 *isupper()*
 38896 XBD Chapter 7 (on page 121), <ctype.h>

38897 **CHANGE HISTORY**
 38898 First released in Issue 1. Derived from Issue 1 of the SVID.

38899 **Issue 6**
 38900 The normative text is updated to avoid use of the term “must” for application requirements.

38901
38902
38903**Issue 7**

The *isxdigit_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



38904 **NAME**
 38905 `j0, j1, jn` — Bessel functions of the first kind

38906 **SYNOPSIS**

```
38907 XSI      #include <math.h>
38908
38908 double j0(double x);
38909 double j1(double x);
38910 double jn(int n, double x);
```

38911 **DESCRIPTION**

38912 The `j0()`, `j1()`, and `jn()` functions shall compute Bessel functions of x of the first kind of orders 0,
 38913 1, and n , respectively.

38914 An application wishing to check for error situations should set `errno` to zero and call
 38915 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 38916 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 38917 zero, an error has occurred.

38918 **RETURN VALUE**

38919 Upon successful completion, these functions shall return the relevant Bessel value of x of the
 38920 first kind.

38921 If the x argument is too large in magnitude, or the correct result would cause underflow, 0 shall
 38922 be returned and a range error may occur.

38923 If x is NaN, a NaN shall be returned.

38924 **ERRORS**

38925 These functions may fail if:

38926 **Range Error** The value of x was too large in magnitude, or an underflow occurred.

38927 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 38928 then `errno` shall be set to [ERANGE]. If the integer expression
 38929 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the underflow
 38930 floating-point exception shall be raised.

38931 No other errors shall occur.

38932 **EXAMPLES**

38933 None.

38934 **APPLICATION USAGE**

38935 On error, the expressions `(math_errhandling & MATH_ERRNO)` and `(math_errhandling &`
 38936 `MATH_ERREXCEPT)` are independent of each other, but at least one of them must be non-zero.

38937 **RATIONALE**

38938 None.

38939 **FUTURE DIRECTIONS**

38940 None.

38941 **SEE ALSO**

38942 [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#), [y0\(\)](#)
 38943 [XBD Section 4.19](#) (on page 104), [<math.h>](#)

38944

CHANGE HISTORY

38945

First released in Issue 1. Derived from Issue 1 of the SVID.

38946

Issue 5

38947

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

38948

38949

Issue 6

38950

The may fail [EDOM] error is removed for the case for NaN.

38951

The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling with the ISO/IEC 9899:1999 standard.

38952



jrand48()38953 **NAME**

38954 jrand48 — generate a uniformly distributed pseudo-random long signed integer

38955 **SYNOPSIS**38956 XSI #include <stdlib.h>
38957 long jrand48(unsigned short xsubi[3]);38958 **DESCRIPTION**38959 Refer to *drand48()*.

38960 **NAME**
 38961 kill — send a signal to a process or a group of processes

38962 **SYNOPSIS**

```
38963 CX #include <signal.h>
38964 int kill(pid_t pid, int sig);
```

38965 **DESCRIPTION**

38966 The *kill()* function shall send a signal to a process or a group of processes specified by *pid*. The
 38967 signal to be sent is specified by *sig* and is either one from the list given in **<signal.h>** or 0. If *sig* is
 38968 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can
 38969 be used to check the validity of *pid*.

38970 For a process to have permission to send a signal to a process designated by *pid*, unless the
 38971 sending process has appropriate privileges, the real or effective user ID of the sending process
 38972 shall match the real or saved set-user-ID of the receiving process.

38973 If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

38974 If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes)
 38975 whose process group ID is equal to the process group ID of the sender, and for which the process
 38976 has permission to send a signal.

38977 If *pid* is -1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for
 38978 which the process has permission to send that signal.

38979 If *pid* is negative, but not -1, *sig* shall be sent to all processes (excluding an unspecified set of
 38980 system processes) whose process group ID is equal to the absolute value of *pid*, and for which
 38981 the process has permission to send a signal.

38982 If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for
 38983 the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait()* function
 38984 for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending
 38985 thread before *kill()* returns.

38986 The user ID tests described above shall not be applied when sending SIGCONT to a process that
 38987 is a member of the same session as the sending process.

38988 An implementation that provides extended security controls may impose further
 38989 implementation-defined restrictions on the sending of signals, including the null signal. In
 38990 particular, the system may deny the existence of some or all of the processes specified by *pid*.

38991 The *kill()* function is successful if the process has permission to send *sig* to any of the processes
 38992 specified by *pid*. If *kill()* fails, no signal shall be sent.

38993 **RETURN VALUE**

38994 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 38995 indicate the error.

38996 **ERRORS**

38997 The *kill()* function shall fail if:

- | | | |
|-------|----------|---|
| 38998 | [EINVAL] | The value of the <i>sig</i> argument is an invalid or unsupported signal number. |
| 38999 | [EPERM] | The process does not have permission to send the signal to any receiving process. |
| 39000 | | |

39001 [ESRCH] No process or process group can be found corresponding to that specified by
 39002 *pid*.

EXAMPLES

39003 None.
 39004

APPLICATION USAGE

39005 None.
 39006

RATIONALE

39007 The semantics for permission checking for *kill()* differed between System V and most other
 39008 implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of
 39009 POSIX.1-200x agree with System V. Specifically, a set-user-ID process cannot protect itself against
 39010 signals (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the
 39011 user who starts an application to send it signals even if it changes its effective user ID. The other
 39012 semantics give more power to an application that wants to protect itself from the user who ran
 39013 it.
 39014

39015 Some implementations provide semantic extensions to the *kill()* function when the absolute
 39016 value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a
 39017 flag to *kill()*. Since most implementations return [ESRCH] in this case, this behavior is not
 39018 included in this volume of POSIX.1-200x, although a conforming implementation could provide
 39019 such an extension.

39020 The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

39021 There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the
 39022 calling process and that signal is not blocked, that signal would be delivered before *kill()*
 39023 returns. This would permit a process to call *kill()* and be guaranteed that the call never return.
 39024 However, historical implementations that provide only the *signal()* function make only the
 39025 weaker guarantee in this volume of POSIX.1-200x, because they only deliver one signal each
 39026 time a process enters the kernel. Modifications to such implementations to support the
 39027 *sigaction()* function generally require entry to the kernel following return from a signal-catching
 39028 function, in order to restore the signal mask. Such modifications have the effect of satisfying the
 39029 stronger requirement, at least when *sigaction()* is used, but not necessarily when *signal()* is used.
 39030 The standard developers considered making the stronger requirement except when *signal()* is
 39031 used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the
 39032 stronger requirement whenever possible. In practice, the weaker requirement is the same, except
 39033 in the rare case when two signals arrive during a very short window. This reasoning also applies
 39034 to a similar requirement for *sigprocmask()*.

39035 In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID
 39036 security checks. This allows a job control shell to continue a job even if processes in the job have
 39037 altered their user IDs (as in the *su* command). In keeping with the addition of the concept of
 39038 sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any
 39039 process in the same session regardless of user ID security checks. This is less restrictive than BSD
 39040 in the sense that ancestor processes (in the same session) can now be the recipient. It is more
 39041 restrictive than BSD in the sense that descendant processes that form new sessions are now
 39042 subject to the user ID checks. A similar relaxation of security is not necessary for the other job
 39043 control signals since those signals are typically sent by the terminal driver in recognition of
 39044 special characters being typed; the terminal driver bypasses all security checks.

39045 In secure implementations, a process may be restricted from sending a signal to a process having
 39046 a different security label. In order to prevent the existence or nonexistence of a process from
 39047 being used as a covert channel, such processes should appear nonexistent to the sender; that is,
 39048 [ESRCH] should be returned, rather than [EPERM], if *pid* refers only to such processes.

39049 Existing implementations vary on the result of a *kill()* with *pid* indicating an inactive process (a
 39050 terminated process that has not been waited for by its parent). Some indicate success on such a

39051 call (subject to permission checking), while others give an error of [ESRCH]. Since the definition
 39052 of process lifetime in this volume of POSIX.1-200x covers inactive processes, the [ESRCH] error
 39053 as described is inappropriate in this case. In particular, this means that an application cannot
 39054 have a parent process check for termination of a particular child with *kill()*. (Usually this is done
 39055 with the null signal; this can be done reliably with *waitpid()*.)

39056 There is some belief that the name *kill()* is misleading, since the function is not always intended
 39057 to cause process termination. However, the name is common to all historical implementations,
 39058 and any change would be in conflict with the goal of minimal changes to existing application
 39059 code.

39060 FUTURE DIRECTIONS

39061 None.

39062 SEE ALSO

39063 *getpid()*, *raise()*, *setsid()*, *sigaction()*, *sigqueue()*, *wait*

39064 XBD [<signal.h>](#), [<sys/types.h>](#)

39065 CHANGE HISTORY

39066 First released in Issue 1. Derived from Issue 1 of the SVID.

39067 Issue 5

39068 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

39069 Issue 6

39070 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

39071 The following new requirements on POSIX implementations derive from alignment with the
 39072 Single UNIX Specification:

- 39073 • In the DESCRIPTION, the second paragraph is reworded to indicate that the saved set-
 39074 user-ID of the calling process is checked in place of its effective user ID. This is a FIPS
 39075 requirement.
- 39076 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 39077 required for conforming implementations of previous POSIX specifications, it was not
 39078 required for UNIX applications.
- 39079 • The behavior when *pid* is -1 is now specified. It was previously explicitly unspecified in
 39080 the POSIX.1-1988 standard.

39081 The normative text is updated to avoid use of the term “must” for application requirements.

39082 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/51 is applied, correcting the RATIONALE
 39083 section.

39084 **NAME**
 39085 `killpg` — send a signal to a process group

39086 **SYNOPSIS**
 39087 XSI

```
#include <signal.h>
```


 39088

```
int killpg(pid_t pgrp, int sig);
```

39089 **DESCRIPTION**
 39090 The `killpg()` function shall send the signal specified by `sig` to the process group specified by `pgrp`.
 39091 If `pgrp` is greater than 1, `killpg(pgrp, sig)` shall be equivalent to `kill(-pgrp, sig)`. If `pgrp` is less than or
 39092 equal to 1, the behavior of `killpg()` is undefined.

39093 **RETURN VALUE**
 39094 Refer to *kill*.

39095 **ERRORS**
 39096 Refer to *kill*.

39097 **EXAMPLES**

39098 **Sending a Signal to All Other Members of a Process Group**

39099 The following example shows how the calling process could send a signal to all other members
 39100 of its process group. To prevent itself from receiving the signal it first makes itself immune to the
 39101 signal by ignoring it.

```
39102 #include <signal.h>
39103 #include <unistd.h>
39104 ...
39105     if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
39106         /* Handle error */;
39107
39108     if (killpg(getpgrp(), SIGUSR1) == -1)
39109         /* Handle error */;
```

39109 **APPLICATION USAGE**
 39110 None.

39111 **RATIONALE**
 39112 None.

39113 **FUTURE DIRECTIONS**
 39114 None.

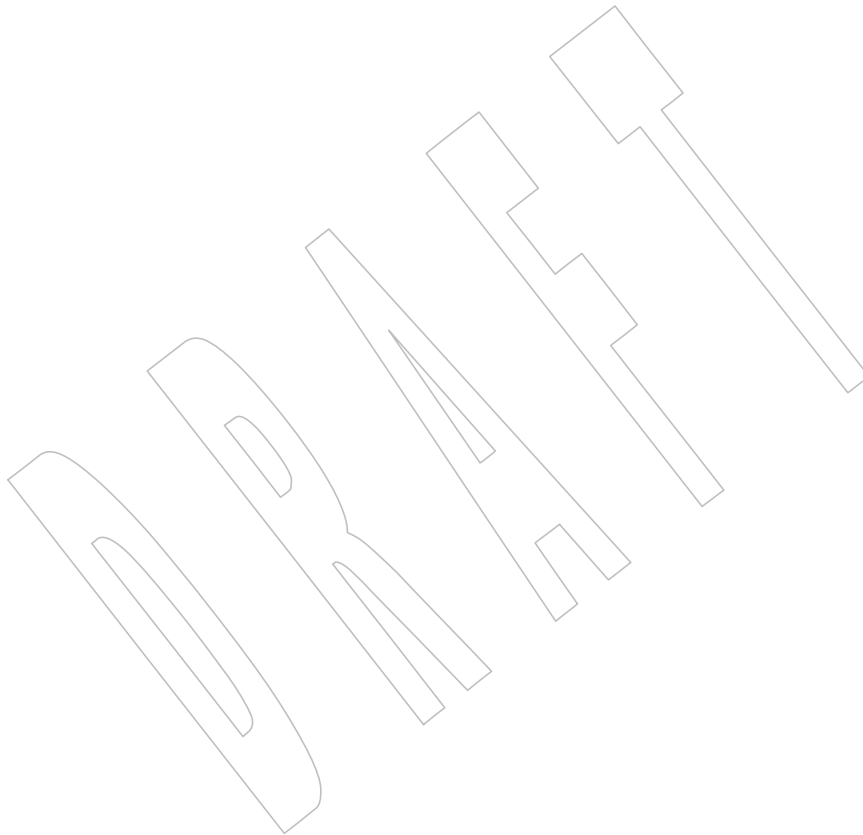
39115 **SEE ALSO**
 39116 *getpgid(), getpid(), kill, raise()*
 39117 XBD *<signal.h>*

39118 **CHANGE HISTORY**
 39119 First released in Issue 4, Version 2.

39120 **Issue 5**
 39121 Moved from X/OPEN UNIX extension to BASE.

39122
39123
39124**Issue 6**

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/52 is applied, adding the example to the EXAMPLES section.



l64a()

39125 **NAME**
39126 l64a — convert a 32-bit integer to a radix-64 ASCII string

39127 **SYNOPSIS**

39128 XSI `#include <stdlib.h>`
39129 `char *l64a(long value);`

39130 **DESCRIPTION**

39131 Refer to [a64l\(\)](#).

39132 **NAME**

39133 labs, llabs — return a long integer absolute value

39134 **SYNOPSIS**

39135 #include <stdlib.h>

39136 long labs(long *i*);39137 long long llabs(long long *i*);39138 **DESCRIPTION**

39139 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39140 conflict between the requirements described here and the ISO C standard is unintentional. This
 39141 volume of POSIX.1-200x defers to the ISO C standard.

39142 The *labs()* function shall compute the absolute value of the **long** integer operand *i*. The *llabs()*
 39143 function shall compute the absolute value of the **long long** integer operand *i*. If the result
 39144 cannot be represented, the behavior is undefined.

39145 **RETURN VALUE**39146 The *labs()* function shall return the absolute value of the **long** integer operand.39147 The *llabs()* function shall return the absolute value of the **long long** integer operand.39148 **ERRORS**

39149 No errors are defined.

39150 **EXAMPLES**

39151 None.

39152 **APPLICATION USAGE**

39153 None.

39154 **RATIONALE**

39155 None.

39156 **FUTURE DIRECTIONS**

39157 None.

39158 **SEE ALSO**39159 [abs\(\)](#)39160 XBD [<stdlib.h>](#) |39161 **CHANGE HISTORY**

39162 First released in Issue 4. Derived from the ISO C standard.

39163 **Issue 6**39164 The *llabs()* function is added for alignment with the ISO/IEC 9899:1999 standard.39165 **Issue 7**

39166 SD5-XSH-ERN-152 is applied, correcting the RETURN VALUE section. +

39167 **NAME**

39168 lchown — change the owner and group of a symbolic link

39169 **SYNOPSIS**

39170 #include <unistd.h>

39171 int lchown(const char *path, uid_t owner, gid_t group);

39172 **DESCRIPTION**

39173 The *lchown()* function shall be equivalent to *chown()*, except in the case where the named file is a
 39174 symbolic link. In this case, *lchown()* shall change the ownership of the symbolic link file itself,
 39175 while *chown()* changes the ownership of the file or directory to which the symbolic link refers.

39176 **RETURN VALUE**

39177 Upon successful completion, *lchown()* shall return 0. Otherwise, it shall return -1 and set *errno* to
 39178 indicate an error.

39179 **ERRORS**39180 The *lchown()* function shall fail if:

- 39181 [EACCES] Search permission is denied on a component of the path prefix of *path*.
- 39182 [EINVAL] The owner or group ID is not a value supported by the implementation.
- 39183 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 39184 argument.
- 39185 [ENAMETOOLONG]
 39186 The length of a pathname exceeds {PATH_MAX} or a pathname component is
 39187 longer than {NAME_MAX}.
- 39188 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.
- 39189 [ENOTDIR] A component of the path prefix of *path* is not a directory.
- 39190 [EOPNOTSUPP] The *path* argument names a symbolic link and the implementation does not
 39191 support setting the owner or group of a symbolic link.
- 39192 [EPERM] The effective user ID does not match the owner of the file and the process does
 39193 not have appropriate privileges.
- 39194 [EROFS] The file resides on a read-only file system.
- 39195 The *lchown()* function may fail if:
- 39196 [EIO] An I/O error occurred while reading or writing to the file system.
- 39197 [EINTR] A signal was caught during execution of the function.
- 39198 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 39199 resolution of the *path* argument.
- 39200 [ENAMETOOLONG]
 39201 Pathname resolution of a symbolic link produced an intermediate result
 39202 whose length exceeds {PATH_MAX}.

39203 **EXAMPLES**39204 **Changing the Current Owner of a File**

39205 The following example shows how to change the ownership of the symbolic link named
39206 **/modules/pass1** to the user ID associated with "jones" and the group ID associated with "cnd".

39207 The numeric value for the user ID is obtained by using the *getpwnam()* function. The numeric
39208 value for the group ID is obtained by using the *getgrnam()* function.

```
39209 #include <sys/types.h>
39210 #include <unistd.h>
39211 #include <pwd.h>
39212 #include <grp.h>

39213 struct passwd *pwd;
39214 struct group *grp;
39215 char          *path = "/modules/pass1";
39216 ...
39217 pwd = getpwnam("jones");
39218 grp = getgrnam("cnd");
39219 lchown(path, pwd->pw_uid, grp->gr_gid);
```

39220 **APPLICATION USAGE**

39221 On implementations which support symbolic links as directory entries rather than files, *lchown()*
39222 may fail.

39223 **RATIONALE**

39224 None.

39225 **FUTURE DIRECTIONS**

39226 None.

39227 **SEE ALSO**

39228 *chown*, *symlink()*

39229 XBD [<unistd.h>](#)

39230 **CHANGE HISTORY**

39231 First released in Issue 4, Version 2.

39232 **Issue 5**

39233 Moved from X/OPEN UNIX extension to BASE.

39234 **Issue 6**

39235 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
39236 [ELOOP] error condition is added.

39237 The Open Group Base Resolution bwg2001-013 is applied, adding wording to the
39238 APPLICATION USAGE.

39239 **Issue 7**

39240 The *lchown()* function is moved from the XSI option to the Base.

lcong48()39241 **NAME**

39242 lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

39243 **SYNOPSIS**39244 XSI #include <stdlib.h>
39245 void lcong48(unsigned short param[7]);39246 **DESCRIPTION**39247 Refer to *drand48()*.

39248 **NAME**
 39249 ldexp, ldexpf, ldexpl — load exponent of a floating-point number

39250 **SYNOPSIS**
 39251 #include <math.h>
 39252 double ldexp(double x, int exp);
 39253 float ldexpf(float x, int exp);
 39254 long double ldexpl(long double x, int exp);

39255 **DESCRIPTION**
 39256 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39257 conflict between the requirements described here and the ISO C standard is unintentional. This
 39258 volume of POSIX.1-200x defers to the ISO C standard.

39259 These functions shall compute the quantity $x * 2^{exp}$.
 39260 An application wishing to check for error situations should set *errno* to zero and call
 39261 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 39262 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 39263 zero, an error has occurred.

39264 **RETURN VALUE**
 39265 Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power
 39266 *exp*.

39267 If these functions would cause overflow, a range error shall occur and *ldexp()*, *ldexpf()*, and
 39268 *ldexpl()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL (according to the sign of
 39269 *x*), respectively.

39270 If the correct value would cause underflow, and is not representable, a range error may occur,
 39271 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

39272 MX If *x* is NaN, a NaN shall be returned.

39273 If *x* is ± 0 or \pm Inf, *x* shall be returned.

39274 If *exp* is 0, *x* shall be returned.

39275 If the correct value would cause underflow, and is representable, a range error may occur and
 39276 the correct value shall be returned.

39277 **ERRORS**

39278 These functions shall fail if:

39279 Range Error The result overflows.

39280 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39281 then *errno* shall be set to [ERANGE]. If the integer expression
 39282 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 39283 floating-point exception shall be raised.

39284 These functions may fail if:

39285 Range Error The result underflows.

39286 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39287 then *errno* shall be set to [ERANGE]. If the integer expression
 39288 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 39289 floating-point exception shall be raised.

39290
39291
39292
39293
39294
39295
39296
39297
39298
39299
39300
39301
39302
39303
39304
39305
39306
39307
39308
39309
39310
39311
39312
39313

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *frexp()*, *isnan()*

XBD Section 4.19 (on page 104), [<math.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *ldexpf()* and *ldexpl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

39314 **NAME**

39315 ldiv, lldiv — compute quotient and remainder of a long division

39316 **SYNOPSIS**

39317 #include <stdlib.h>

39318 ldiv_t ldiv(long numer, long denom);

39319 lldiv_t lldiv(long long numer, long long denom);

39320 **DESCRIPTION**

39321 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39322 conflict between the requirements described here and the ISO C standard is unintentional. This
 39323 volume of POSIX.1-200x defers to the ISO C standard.

39324 These functions shall compute the quotient and remainder of the division of the numerator
 39325 *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long**
 39326 integer (for the *ldiv()* function) or **long long** integer (for the *lldiv()* function) of lesser magnitude
 39327 that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is
 39328 undefined; otherwise, *quot * denom + rem* shall equal *numer*.

39329 **RETURN VALUE**

39330 The *ldiv()* function shall return a structure of type **ldiv_t**, comprising both the quotient and the
 39331 remainder. The structure shall include the following members, in any order:

```
39332 long    quot;    /* Quotient */
39333 long    rem;     /* Remainder */
```

39334 The *lldiv()* function shall return a structure of type **lldiv_t**, comprising both the quotient and the
 39335 remainder. The structure shall include the following members, in any order:

```
39336 long long quot;    /* Quotient */
39337 long long rem;     /* Remainder */
```

39338 **ERRORS**

39339 No errors are defined.

39340 **EXAMPLES**

39341 None.

39342 **APPLICATION USAGE**

39343 None.

39344 **RATIONALE**

39345 None.

39346 **FUTURE DIRECTIONS**

39347 None.

39348 **SEE ALSO**39349 [div\(\)](#)39350 XBD [<stdlib.h>](#)39351 **CHANGE HISTORY**

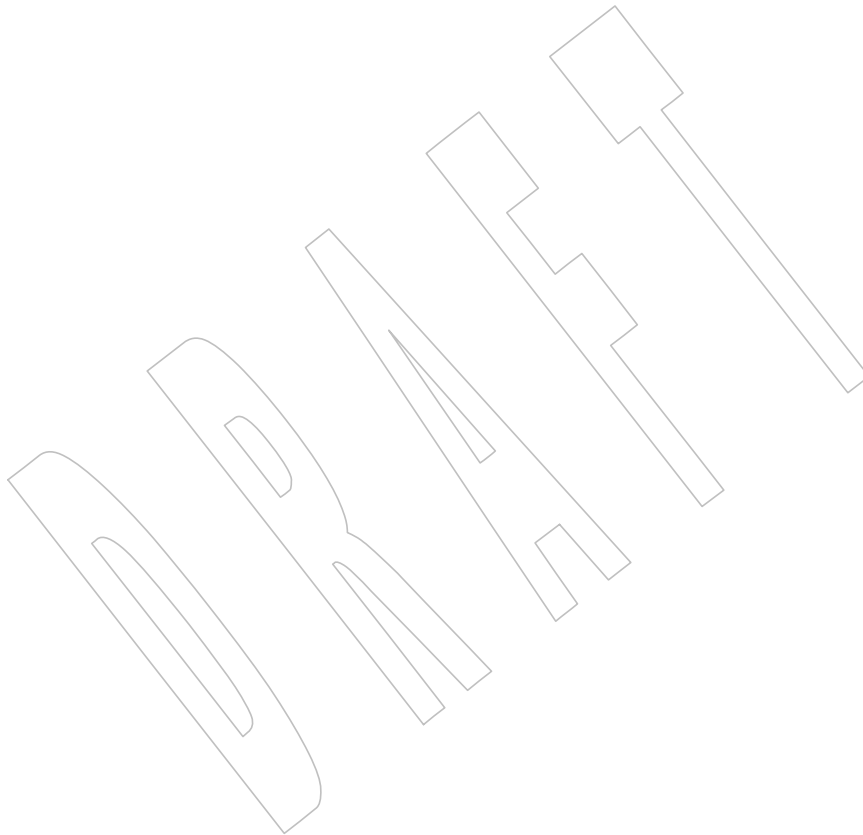
39352 First released in Issue 4. Derived from the ISO C standard.

39353

39354

Issue 6

The *lldiv()* function is added for alignment with the ISO/IEC 9899:1999 standard.



39355 **NAME**
39356 lfind — find entry in a linear search table

39357 **SYNOPSIS**

```
39358 XSI #include <search.h>  
39359 void *lfind(const void *key, const void *base, size_t *nelp,  
39360           size_t width, int (*compar)(const void *, const void *));
```

39361 **DESCRIPTION**

39362 Refer to [lsearch\(\)](#).

39363 **NAME**
 39364 lgamma, lgammaf, lgammal — log gamma function

39365 **SYNOPSIS**
 39366 #include <math.h>
 39367 double lgamma(double x);
 39368 float lgammaf(float x);
 39369 long double lgammal(long double x);
 39370 XSI extern int signgam;

39371 DESCRIPTION

39372 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39373 conflict between the requirements described here and the ISO C standard is unintentional. This
 39374 volume of POSIX.1-200x defers to the ISO C standard.

39375 These functions shall compute $\log_e |\Gamma(x)|$ where $\Gamma(x)$ is defined as $\int_0^{\infty} e^{-t} t^{x-1} dt$. The argument x
 39376 need not be a non-positive integer ($\Gamma(x)$ is defined over the reals, except the non-positive
 39377 integers).

39378 XSI The sign of $\Gamma(x)$ is returned in the external integer *signgam*.

39379 CX These functions need not be thread-safe. A function that is not required to be thread-safe is not
 39380 required to be reentrant.

39381 An application wishing to check for error situations should set *errno* to zero and call
 39382 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 39383 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 39384 zero, an error has occurred.

39385 RETURN VALUE

39386 Upon successful completion, these functions shall return the logarithmic gamma of x .

39387 If x is a non-positive integer, a pole error shall occur and *lgamma()*, *lgammaf()*, and *lgammal()*
 39388 shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

39389 If the correct value would cause overflow, a range error shall occur and *lgamma()*, *lgammaf()*,
 39390 and *lgammal()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL (having the same
 39391 sign as the correct value), respectively.

39392 MX If x is NaN, a NaN shall be returned.

39393 If x is 1 or 2, +0 shall be returned.

39394 If x is \pm Inf, +Inf shall be returned.

39395 ERRORS

39396 These functions shall fail if:

39397 Pole Error The x argument is a negative integer or zero.

39398 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39399 then *errno* shall be set to [ERANGE]. If the integer expression
 39400 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 39401 floating-point exception shall be raised.

39402 Range Error The result overflows.

39403 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

39404 then *errno* shall be set to [ERANGE]. If the integer expression

39405 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow

39406 floating-point exception shall be raised.

EXAMPLES

39407 None.

39408

APPLICATION USAGE

39409 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &

39410 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

39411

RATIONALE

39412 None.

39413

FUTURE DIRECTIONS

39414 None.

39415

SEE ALSO

39416 *exp()*, *feclearexcept()*, *fetestexcept()*, *isnan()*

39417

39418 XBD Section 4.19 (on page 104), `<math.h>`

39419

CHANGE HISTORY

39420 First released in Issue 3.

Issue 5

39421 The DESCRIPTION is updated to indicate how an application should check for an error. This

39422 text was previously published in the APPLICATION USAGE section.

39423

39424 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

39425 The *lgamma()* function is no longer marked as an extension.

39426

39427 The *lgammaf()* and *lgammal()* functions are added for alignment with the ISO/IEC 9899:1999

39428 standard.

39429 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are

39430 revised to align with the ISO/IEC 9899:1999 standard.

39431 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are

39432 marked.

39433 Functionality relating to the XSI option is marked.

39434 **NAME**

39435 link, linkat — link one file to another file relative to two directory file descriptors

39436 **SYNOPSIS**

```
39437 #include <unistd.h>
39438
39438 int link(const char *path1, const char *path2);
39439 int linkat(int fd1, const char *path1, int fd2, const char *path2,
39440 int flag);
```

39441 **DESCRIPTION**39442 The *link()* function shall create a new link (directory entry) for the existing file, *path1*.

39443 The *path1* argument points to a pathname naming an existing file. The *path2* argument points to
 39444 a pathname naming the new directory entry to be created. The *link()* function shall atomically
 39445 create a new link for the existing file and the link count of the file shall be incremented by one.

39446 If *path1* names a directory, *link()* shall fail unless the process has appropriate privileges and the
 39447 implementation supports using *link()* on directories.

39448 If *path1* names a symbolic link, it is implementation-defined whether *link()* follows the symbolic +
 39449 link, or creates a new link to the symbolic link itself. +

39450 Upon successful completion, *link()* shall mark for update the last file status change timestamp of |
 39451 the file. Also, the last data modification and last file status change timestamps of the directory |
 39452 that contains the new entry shall be marked for update.

39453 If *link()* fails, no link shall be created and the link count of the file shall remain unchanged.

39454 The implementation may require that the calling process has permission to access the existing
 39455 file.

39456 The *linkat()* function shall be equivalent to the *link()* function except in the case where either
 39457 *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to
 39458 the directory associated with the file descriptor *fd1* instead of the current working directory and
 39459 similarly for *path2* and the file descriptor *fd2*. It is unspecified whether directory searches are
 39460 permitted based on whether the file was opened with search permission or on the current
 39461 permissions of the directory underlying the file descriptor.

39462 Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 39463 in **<fcntl.h>**:

39464 AT_SYMLINK_FOLLOW

39465 If *path1* names a symbolic link, a new link for the target of the symbolic link is
 39466 created.

39467 If *linkat()* is passed the special value AT_FDCWD in the *fd1* or *fd2* parameter, the current
 39468 working directory is used for the respective *path* argument. If both *fd1* and *fd2* have value
 39469 AT_FDCWD, the behavior shall be identical to a call to *link()*.

39470 If the AT_SYMLINK_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a |
 39471 symbolic link, a new link is created for the symbolic link *path1* and not its target, unless the |
 39472 implementation does not support making hard links to symbolic links in which case the *linkat()* |
 39473 call shall fail.

39474 **RETURN VALUE**

39475 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 39476 return `-1` and set `errno` to indicate the error.

39477 **ERRORS**

39478 These functions shall fail if:

39479 [EACCES] A component of either path prefix denies search permission, or the requested
 39480 link requires writing in a directory that denies write permission, or the calling
 39481 process does not have permission to access the existing file and this is required
 39482 by the implementation.

39483 [EEXIST] The `path2` argument resolves to an existing file or refers to a symbolic link.

39484 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path1` or
 39485 `path2` argument.

39486 [EMLINK] The number of links to the file named by `path1` would exceed `{LINK_MAX}`.

39487 [ENAMETOOLONG]

39488 The length of the `path1` or `path2` argument exceeds `{PATH_MAX}` or a
 39489 pathname component is longer than `{NAME_MAX}`.

39490 [ENOENT] A component of either path prefix does not exist; the file named by `path1` does
 39491 not exist; or `path1` or `path2` points to an empty string.

39492 [ENOSPC] The directory to contain the link cannot be extended.

39493 [ENOTDIR] A component of either path prefix is not a directory.

39494 [EPERM] The file named by `path1` is a directory and either the calling process does not
 39495 have appropriate privileges or the implementation prohibits using `link()` on
 39496 directories.

39497 [EROFS] The requested link requires writing in a directory on a read-only file system.

39498 [EXDEV] The link named by `path2` and the file named by `path1` are on different file
 39499 systems and the implementation does not support links between file systems.

39500 OB XSR [EXDEV] `path1` refers to a named STREAM.

39501 The `linkat()` function shall fail if:

39502 [EBADF] The `path1` or `path2` argument does not specify an absolute path and the `fd1` or
 39503 `fd2` argument, respectively, is neither `AT_FDCWD` nor a valid file descriptor
 39504 open for reading.

39505 [EOPNOTSUPP]

39506 The `AT_SYMLINK_FOLLOW` flag is clear in the `flag` argument, the `path1`
 39507 argument names a symbolic link, and the implementation does not support
 39508 making hard links to symbolic links.

39509 These functions may fail if:

39510 [ELOOP] More than `{SYMLOOP_MAX}` symbolic links were encountered during
 39511 resolution of the `path1` or `path2` argument.

39512 [ENAMETOOLONG]

39513 As a result of encountering a symbolic link in resolution of the `path1` or `path2`
 39514 argument, the length of the substituted pathname string exceeded
 39515 `{PATH_MAX}`.

39516 The `linkat()` function may fail if:

39517 [EINVAL] The value of the *flag* argument is not valid.

39518 [ENOTDIR] The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*,
39519 respectively, is neither AT_FDCWD nor a file descriptor associated with a
39520 directory.

EXAMPLES**Creating a Link to a File**

The following example shows how to create a link to a file named `/home/cnd/mod1` by creating a new directory entry named `/modules/pass1`.

```
39521 #include <unistd.h>
39522
39523 char *path1 = "/home/cnd/mod1";
39524 char *path2 = "/modules/pass1";
39525 int status;
39526 ...
39527 status = link (path1, path2);
```

Creating a Link to a File Within a Program

In the following program example, the `link()` function links the `/etc/passwd` file (defined as `PASSWDFILE`) to a file named `/etc/opasswd` (defined as `SAVEFILE`), which is used to save the current password file. Then, after removing the current password file (defined as `PASSWDFILE`), the new password file is saved as the current password file using the `link()` function again.

```
39531 #include <unistd.h>
39532 #define LOCKFILE "/etc/ptmp"
39533 #define PASSWDFILE "/etc/passwd"
39534 #define SAVEFILE "/etc/opasswd"
39535 ...
39536 /* Save current password file */
39537 link (PASSWDFILE, SAVEFILE);
39538
39539 /* Remove current password file. */
39540 unlink (PASSWDFILE);
39541
39542 /* Save new password file as current password file. */
39543 link (LOCKFILE, PASSWDFILE);
```

APPLICATION USAGE

Some implementations do allow links between file systems.

If *path1* refers to a symbolic link, application developers should use `linkat()` with appropriate + flags to select whether or not the symbolic link should be resolved.

RATIONALE

Linking to a directory is restricted to the superuser in most historical implementations because this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This volume of POSIX.1-200x continues that philosophy by prohibiting `link()` and `unlink()` from doing this. Other functions could do it if the implementor designed such an extension.

Some historical implementations allow linking of files on different file systems. Wording was added to explicitly allow this optional behavior.

The exception for cross-file system links is intended to apply only to links that are programmatically indistinguishable from "hard" links.

39561 The purpose of the *linkat()* function is to link files in directories other than the current working
 39562 directory without exposure to race conditions. Any part of the path of a file could be changed in
 39563 parallel to a call to *link()*, resulting in unspecified behavior. By opening a file descriptor for the
 39564 directory of both the existing file and the target location and using the *linkat()* function it can be
 39565 guaranteed that the both filenames are in the desired directories.

39566 The AT_SYMLINK_FOLLOW flag allows for implementing both common behaviors of the
 39567 *link()* function. The POSIX specification requires that if *path1* is a symbolic link, a new link for
 39568 the target of the symbolic link is created. Many systems by default or as an alternative provide a
 39569 mechanism to avoid the implicit symlink lookup and create a new link for the symbolic link
 39570 itself.

39571 Earlier versions of this standard specified only the *link()* function, and required it to behave like
 39572 *linkat()* with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and
 39573 Linux kernels had *link()* behaving like *linkat()* with no flags, and many systems that attempted
 39574 to provide a conforming *link()* function did so in a way that was rarely used, and when it was
 39575 used did not conform to the standard (e.g., by not being atomic, or by dereferencing the
 39576 symbolic link incorrectly). Since applications could not rely on *link()* following links in practice,
 39577 the *linkat()* function was added taking a flag to specify the desired behavior for the application.

39578 FUTURE DIRECTIONS

39579 None.

39580 SEE ALSO

39581 *rename()*, *symlink()*, *unlink*

39582 XBD `<fcntl.h>`, `<unistd.h>`

39583 CHANGE HISTORY

39584 First released in Issue 1. Derived from Issue 1 of the SVID.

39585 Issue 6

39586 The following new requirements on POSIX implementations derive from alignment with the
 39587 Single UNIX Specification:

- 39588 • The [ELOOP] mandatory error condition is added.
- 39589 • A second [ENAMETOOLONG] is added as an optional error condition.

39590 The following changes were made to align with the IEEE P1003.1a draft standard:

- 39591 • An explanation is added of the action when *path2* refers to a symbolic link.
- 39592 • The [ELOOP] optional error condition is added.

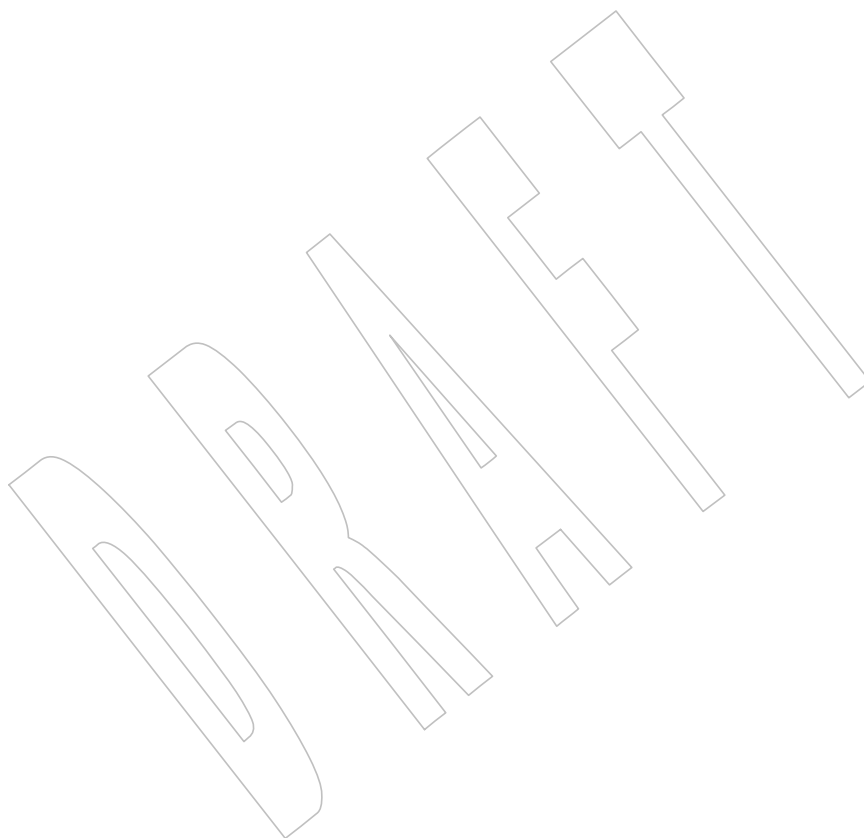
39593 Issue 7

39594 SD5-XSH-ERN-93 is applied, adding RATIONALE.

39595 The *linkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set
 39596 Part 2.

39597 Functionality relating to XSI STREAMS is marked obsolescent.

39598 Changes are made related to support for finegrained timestamps.

39599 **NAME**39600 `linkat` — link one file to another file relative to two directory file descriptors39601 **SYNOPSIS**39602 `#include <unistd.h>`39603 `int linkat(int fd1, const char *path1, int fd2, const char *path2,
39604 int flag);`39605 **DESCRIPTION**39606 Refer to [link](#).

39607 **NAME**

39608 lio_listio — list directed I/O

39609 **SYNOPSIS**

39610 #include <aio.h>

39611 int lio_listio(int mode, struct aiocb *restrict const list[restrict],
39612 int nent, struct sigevent *restrict sig);39613 **DESCRIPTION**39614 The *lio_listio()* function shall initiate a list of I/O requests with a single function call.39615 The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <aio.h> and
39616 determines whether the function returns when the I/O operations have been completed, or as
39617 soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall
39618 wait until all I/O is complete and the *sig* argument shall be ignored.39619 If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous
39620 notification shall occur, according to the *sig* argument, when all the I/O operations complete. If
39621 *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous
39622 notification occurs as specified in Section 2.4.1 (on page 463) when all the requests in *list* have
39623 completed.39624 The I/O requests enumerated by *list* are submitted in an unspecified order.39625 The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements.
39626 The array may contain NULL elements, which shall be ignored.39627 If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list*
39628 become illegal addresses before all asynchronous I/O completed and, if necessary, the
39629 notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf*
39630 member of the **aiocb** structure pointed to by the elements of the array *list* become illegal
39631 addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed,
39632 the behavior is undefined.39633 The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The
39634 supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in
39635 <aio.h>. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode*
39636 element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read()*
39637 with the *aio_cbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to
39638 LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write()* with the *aio_cbp*
39639 equal to the address of the **aiocb** structure.39640 The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.39641 The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.39642 The *aio_nbytes* member specifies the number of bytes of data to be transferred.39643 The members of the **aiocb** structure further describe the I/O operation to be performed, in a
39644 manner identical to that of the corresponding **aiocb** structure when used by the *aio_read()* and
39645 *aio_write()* functions.39646 The *nent* argument specifies how many elements are members of the list; that is, the length of the
39647 array.39648 The behavior of this function is altered according to the definitions of synchronized I/O data
39649 integrity completion and synchronized I/O file integrity completion if synchronized I/O is
39650 enabled on the file associated with *aio_fildes*.

39651 For regular files, no data transfer shall occur past the offset maximum established in the open
39652 file description associated with *aio*cbp->*aio_fildes*.

39653 If *sig*->*sigev_notify* is SIGEV_THREAD and *sig*->*sigev_notify_attributes* is a non-NULL pointer
39654 and the block pointed to by this pointer becomes an illegal address prior to all asynchronous
39655 I/O being completed, then the behavior is undefined.

RETURN VALUE

39656 If the *mode* argument has the value LIO_NOWAIT, the *lio_listio()* function shall return the value
39657 zero if the I/O operations are successfully queued; otherwise, the function shall return the value
39658 -1 and set *errno* to indicate the error.
39659

39660 If the *mode* argument has the value LIO_WAIT, the *lio_listio()* function shall return the value zero
39661 when all the indicated I/O has completed successfully. Otherwise, *lio_listio()* shall return a value
39662 of -1 and set *errno* to indicate the error.

39663 In either case, the return value only indicates the success or failure of the *lio_listio()* call itself,
39664 not the status of the individual I/O requests. In some cases one or more of the I/O requests
39665 contained in the list may fail. Failure of an individual request does not prevent completion of
39666 any other individual request. To determine the outcome of each I/O request, the application
39667 shall examine the error status associated with each **aio**cb control block. The error statuses so
39668 returned are identical to those returned as the result of an *aio_read()* or *aio_write()* function.

ERRORS

39669 The *lio_listio()* function shall fail if:
39670

39671 [EAGAIN] The resources necessary to queue all the I/O requests were not available. The
39672 application may check the error status for each **aio**cb to determine the
39673 individual request(s) that failed.

39674 [EAGAIN] The number of entries indicated by *nent* would cause the system-wide limit
39675 {AIO_MAX} to be exceeded.

39676 [EINVAL] The *mode* argument is not a proper value, or the value of *nent* was greater than
39677 {AIO_LISTIO_MAX}.

39678 [EINTR] A signal was delivered while waiting for all I/O requests to complete during
39679 an LIO_WAIT operation. Note that, since each I/O operation invoked by
39680 *lio_listio()* may possibly provoke a signal when it completes, this error return
39681 may be caused by the completion of one (or more) of the very I/O operations
39682 being awaited. Outstanding I/O requests are not canceled, and the application
39683 shall examine each list element to determine whether the request was
39684 initiated, canceled, or completed.

39685 [EIO] One or more of the individual I/O operations failed. The application may
39686 check the error status for each **aio**cb structure to determine the individual
39687 request(s) that failed.

39688 In addition to the errors returned by the *lio_listio()* function, if the *lio_listio()* function succeeds
39689 or fails with errors of [EAGAIN], [EINTR], or [EIO], then some of the I/O specified by the list
39690 may have been initiated. If the *lio_listio()* function fails with an error code other than [EAGAIN],
39691 [EINTR], or [EIO], no operations from the list shall have been initiated. The I/O operation
39692 indicated by each list element can encounter errors specific to the individual read or write
39693 function being performed. In this event, the error status for each **aio**cb control block contains the
39694 associated error code. The error codes that can be set are the same as would be set by a *read()* or
39695 *write()* function, with the following additional error codes possible:

39696 [EAGAIN] The requested I/O operation was not queued due to resource limitations.

- 39697 [ECANCELED] The requested I/O was canceled before the I/O completed due to an explicit
39698 *aio_cancel()* request.
- 39699 [EFBIG] The *aiocbp->aio_lio_opcode* is LIO_WRITE, the file is a regular file,
39700 *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is greater than or
39701 equal to the offset maximum in the open file description associated with
39702 *aiocbp->aio_fildes*.
- 39703 [EINPROGRESS] The requested I/O is in progress.
- 39704 [EOVERFLOW] The *aiocbp->aio_lio_opcode* is LIO_READ, the file is a regular file,
39705 *aiocbp->aio_nbytes* is greater than 0, and the *aiocbp->aio_offset* is before the
39706 end-of-file and is greater than or equal to the offset maximum in the open file
39707 description associated with *aiocbp->aio_fildes*.

EXAMPLES

39708 None.
39709

APPLICATION USAGE

39710 None.
39711

RATIONALE

39712 Although it may appear that there are inconsistencies in the specified circumstances for error
39713 codes, the [EIO] error condition applies when any circumstance relating to an individual
39714 operation makes that operation fail. This might be due to a badly formulated request (for
39715 example, the *aio_lio_opcode* field is invalid, and *aio_error()* returns [EINVAL]) or might arise from
39716 application behavior (for example, the file descriptor is closed before the operation is initiated,
39717 and *aio_error()* returns [EBADF]).
39718

39719 The limitation on the set of error codes returned when operations from the list shall have been
39720 initiated enables applications to know when operations have been started and whether
39721 *aio_error()* is valid for a specific operation.

FUTURE DIRECTIONS

39722 None.
39723

SEE ALSO

39724 *aio_read()*, *aio_write()*, *aio_error()*, *aio_return()*, *aio_cancel()*, *close()*, *exec*, *exit()*, *fork()*, *lseek()*,
39725 *read*
39726

39727 XBD <*aio.h*>
39728

CHANGE HISTORY

39729 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

39730 Large File Summit extensions are added.
39731

Issue 6

39732 The [ENOSYS] error condition has been removed as stubs need not be provided if an
39733 implementation does not support the Asynchronous Input and Output option.

39734 The *lio_listio()* function is marked as part of the Asynchronous Input and Output option.

39735 The following new requirements on POSIX implementations derive from alignment with the
39736 Single UNIX Specification:

- 39737 • In the DESCRIPTION, text is added to indicate that for regular files no data transfer occurs
39738 past the offset maximum established in the open file description associated with
39739 *aiocbp->aio_fildes*. This change is to support large files.
- 39740 • The [EBIG] and [EOVERFLOW] error conditions are defined. This change is to support
39741 large files.

39742 The normative text is updated to avoid use of the term “must” for application requirements.

39743 The **restrict** keyword is added to the *lio_listio()* prototype for alignment with the
39744 ISO/IEC 9899:1999 standard.

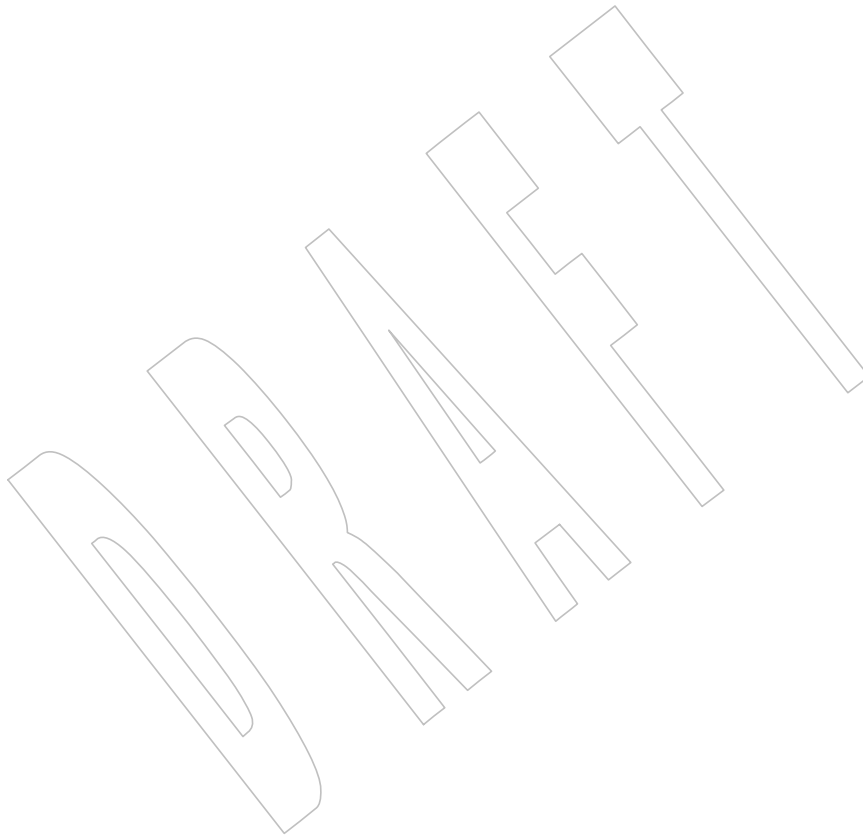
39745 **Issue 6**

39746 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/53 is applied, adding new text for
39747 symmetry with the *aio_read()* and *aio_write()* functions to the DESCRIPTION.

39748 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/54 is applied, adding text to the
39749 DESCRIPTION making it explicit that the user is required to keep the structure pointed to by
39750 *sig->sigev_notify_attributes* valid until the last asynchronous operation finished and the
39751 notification has been sent.

39752 **Issue 7**

39753 The *lio_listio()* function is moved from the Asynchronous Input and Output option to the Base.



39754 **NAME**

39755 listen — listen for socket connections and limit the queue of incoming connections

39756 **SYNOPSIS**

39757 #include <sys/socket.h>

39758 int listen(int *socket*, int *backlog*);39759 **DESCRIPTION**39760 The *listen()* function shall mark a connection-mode socket, specified by the *socket* argument, as
39761 accepting connections.39762 The *backlog* argument provides a hint to the implementation which the implementation shall use
39763 to limit the number of outstanding connections in the socket's listen queue. Implementations
39764 may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog*
39765 argument value shall result in a larger or equal length of the listen queue. Implementations shall
39766 support values of *backlog* up to SOMAXCONN, defined in <sys/socket.h>.39767 The implementation may include incomplete connections in its listen queue. The limits on the
39768 number of incomplete connections and completed connections queued may be different.39769 The implementation may have an upper limit on the length of the listen queue—either global or
39770 per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.39771 If *listen()* is called with a *backlog* argument value that is less than 0, the function behaves as if it
39772 had been called with a *backlog* argument value of 0.39773 A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of
39774 the listen queue may be set to an implementation-defined minimum value.39775 The socket in use may require the process to have appropriate privileges to use the *listen()*
39776 function.39777 **RETURN VALUE**39778 Upon successful completions, *listen()* shall return 0; otherwise, -1 shall be returned and *errno* set
39779 to indicate the error.39780 **ERRORS**39781 The *listen()* function shall fail if:39782 [EBADF] The *socket* argument is not a valid file descriptor.

39783 [EDESTADDRREQ]

39784 The socket is not bound to a local address, and the protocol does not support
39785 listening on an unbound socket.39786 [EINVAL] The *socket* is already connected.39787 [ENOTSOCK] The *socket* argument does not refer to a socket.39788 [EOPNOTSUPP] The socket protocol does not support *listen()*.39789 The *listen()* function may fail if:

39790 [EACCES] The calling process does not have the appropriate privileges.

39791 [EINVAL] The *socket* has been shut down.

39792 [ENOBUFS] Insufficient resources are available in the system to complete the call.

listen()

39793
39794
39795
39796
39797
39798
39799
39800
39801
39802
39803
39804
39805
39806
39807

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

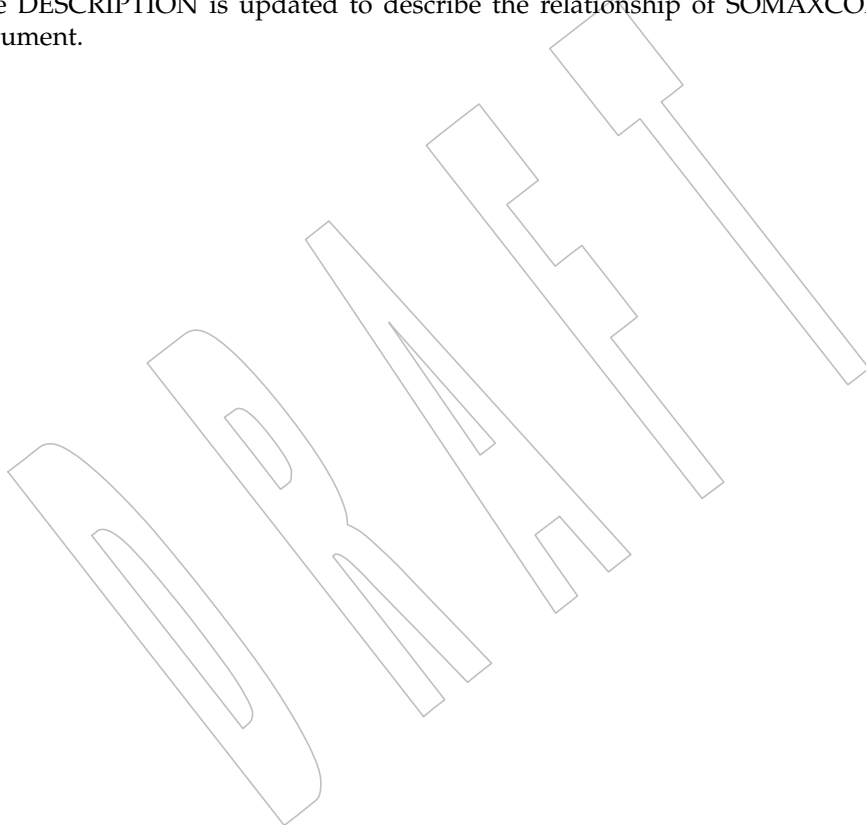
accept(), *connect()*, *socket()*

XBD <[sys/socket.h](#)>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

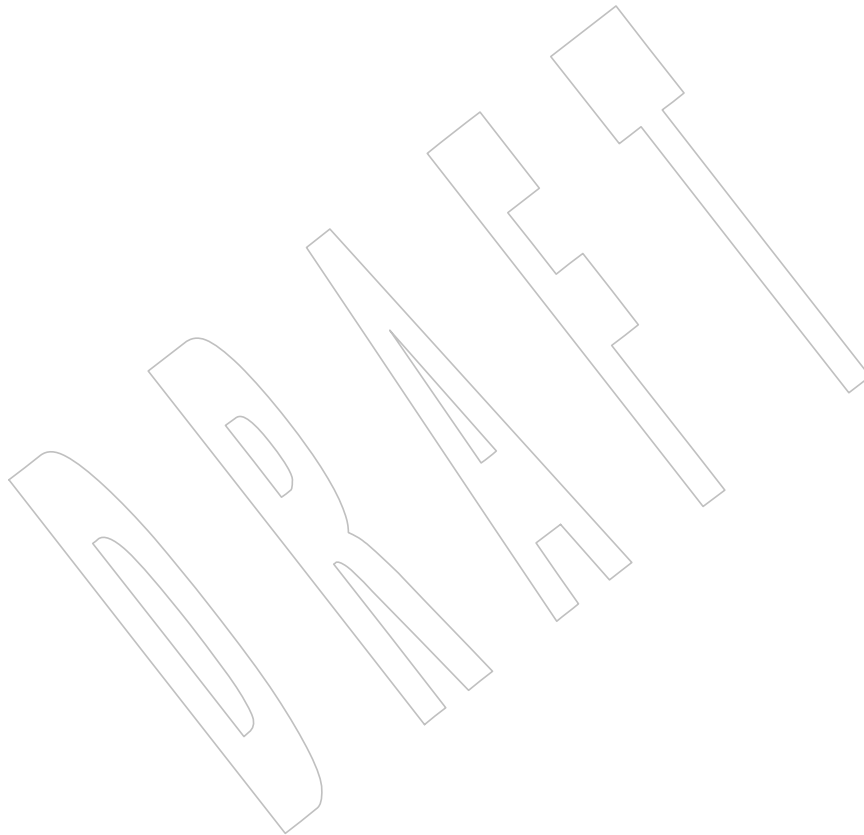
The DESCRIPTION is updated to describe the relationship of SOMAXCONN and the *backlog* argument.



39808 **NAME**
39809 llabs — return a long integer absolute value

39810 **SYNOPSIS**
39811 #include <stdlib.h>
39812 long long llabs(long long i);

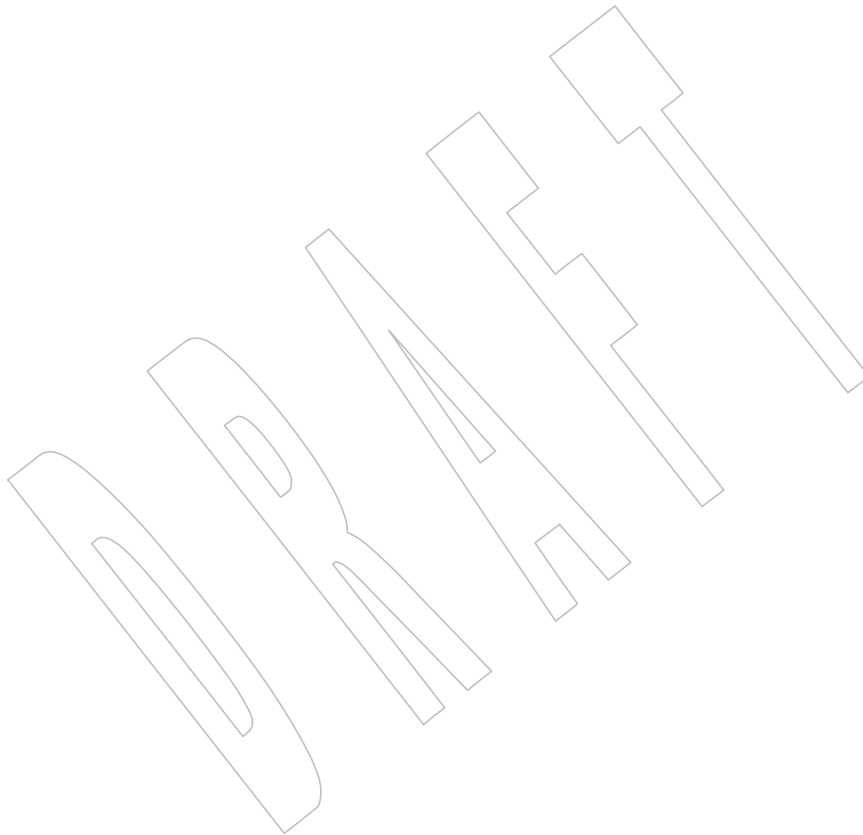
39813 **DESCRIPTION**
39814 Refer to *labs()*.



39815 **NAME**
39816 `lldiv` — compute quotient and remainder of a long division

39817 **SYNOPSIS**
39818 `#include <stdlib.h>`
39819 `lldiv_t lldiv(long long numer, long long denom);`

39820 **DESCRIPTION**
39821 Refer to *ldiv()*.



39822 **NAME**

39823 llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

39824 **SYNOPSIS**

```
39825 #include <math.h>
39826
39826 long long llrint(double x);
39827 long long llrintf(float x);
39828 long long llrintl(long double x);
```

39829 **DESCRIPTION**

39830 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39831 conflict between the requirements described here and the ISO C standard is unintentional. This
 39832 volume of POSIX.1-200x defers to the ISO C standard.

39833 These functions shall round their argument to the nearest integer value, rounding according to
 39834 the current rounding direction.

39835 An application wishing to check for error situations should set *errno* to zero and call
 39836 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 39837 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 39838 zero, an error has occurred.

39839 **RETURN VALUE**

39840 Upon successful completion, these functions shall return the rounded integer value.

39841 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

39842 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

39843 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

39844 If the correct value is positive and too large to represent as a **long long**, an unspecified value
 39845 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 39846 shall occur;

39847 CX otherwise, a domain error may occur.

39848 If the correct value is negative and too large to represent as a **long long**, an unspecified value
 39849 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 39850 shall occur;

39851 CX otherwise, a domain error may occur.

39852 **ERRORS**

39853 These functions shall fail if:

39854 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 39855 integer.

39856 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39857 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 39858 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 39859 shall be raised.

39860 These functions may fail if:

39861 **Domain Error** The correct value is not representable as an integer.

39862 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39863 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 39864 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

39865 shall be raised.

39866 **EXAMPLES**

39867 None.

39868 **APPLICATION USAGE**

39869 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
39870 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

39871 **RATIONALE**

39872 These functions provide floating-to-integer conversions. They round according to the current
39873 rounding direction. If the rounded value is outside the range of the return type, the numeric
39874 result is unspecified and the invalid floating-point exception is raised. When they raise no other
39875 floating-point exception and the result differs from the argument, they raise the inexact floating-
39876 point exception.

39877 **FUTURE DIRECTIONS**

39878 None.

39879 **SEE ALSO**

39880 *feclearexcept()*, *fetestexcept()*, *lrint()*

39881 XBD Section 4.19 (on page 104), <math.h>

39882 **CHANGE HISTORY**

39883 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

39884 **Issue 7**

39885 ISO C TC2 #53 is applied.

39886 **NAME**
 39887 llround, llroundf, llroundl — round to nearest integer value

39888 **SYNOPSIS**
 39889 #include <math.h>
 39890 long long llround(double x);
 39891 long long llroundf(float x);
 39892 long long llroundl(long double x);

39893 **DESCRIPTION**
 39894 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39895 conflict between the requirements described here and the ISO C standard is unintentional. This
 39896 volume of POSIX.1-200x defers to the ISO C standard.

39897 These functions shall round their argument to the nearest integer value, rounding halfway cases
 39898 away from zero, regardless of the current rounding direction.

39899 An application wishing to check for error situations should set *errno* to zero and call
 39900 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 39901 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 39902 zero, an error has occurred.

39903 **RETURN VALUE**
 39904 Upon successful completion, these functions shall return the rounded integer value.

39905 MX If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

39906 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

39907 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

39908 If the correct value is positive and too large to represent as a **long long**, an unspecified value
 39909 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 39910 shall occur;

39911 CX otherwise, a domain error may occur.

39912 If the correct value is negative and too large to represent as a **long long**, an unspecified value
 39913 MX shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error
 39914 shall occur;

39915 CX otherwise, a domain error may occur.

39916 **ERRORS**
 39917 These functions shall fail if:

39918 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 39919 integer.

39920 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39921 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 39922 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 39923 shall be raised.

39924 These functions may fail if:

39925 **Domain Error** The correct value is not representable as an integer.

39926 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 39927 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 39928 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions *(math_errhandling & MATH_ERRNO)* and *(math_errhandling & MATH_ERREXCEPT)* are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *llrint()* functions in that the default rounding direction for the *llround()* functions round halfway cases away from zero and need not raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *lround()*

XBD Section 4.19 (on page 104), [<math.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-75) is applied.

39948 **NAME**
 39949 localeconv — return locale-specific information

39950 **SYNOPSIS**
 39951 #include <locale.h>
 39952 struct lconv *localeconv(void);

39953 **DESCRIPTION**

39954 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 39955 conflict between the requirements described here and the ISO C standard is unintentional. This
 39956 volume of POSIX.1-200x defers to the ISO C standard.

39957 The *localeconv()* function shall set the components of an object with the type **struct lconv** with
 39958 the values appropriate for the formatting of numeric quantities (monetary and otherwise)
 39959 according to the rules of the current locale.

39960 The members of the structure with type **char *** are pointers to strings, any of which (except
 39961 **decimal_point**) can point to " ", to indicate that the value is not available in the current locale or
 39962 is of zero length. The members with type **char** are non-negative numbers, any of which can be
 39963 {CHAR_MAX} to indicate that the value is not available in the current locale.

39964 The members include the following:

39965 **char *decimal_point**
 39966 The radix character used to format non-monetary quantities.

39967 **char *thousands_sep**
 39968 The character used to separate groups of digits before the decimal-point character in
 39969 formatted non-monetary quantities.

39970 **char *grouping**
 39971 A string whose elements taken as one-byte integer values indicate the size of each group of
 39972 digits in formatted non-monetary quantities.

39973 **char *int_curr_symbol**
 39974 The international currency symbol applicable to the current locale. The first three characters
 39975 contain the alphabetic international currency symbol in accordance with those specified in
 39976 the ISO 4217:2001 standard. The fourth character (immediately preceding the null byte) is
 39977 the character used to separate the international currency symbol from the monetary
 39978 quantity.

39979 **char *currency_symbol**
 39980 The local currency symbol applicable to the current locale.

39981 **char *mon_decimal_point**
 39982 The radix character used to format monetary quantities.

39983 **char *mon_thousands_sep**
 39984 The separator for groups of digits before the decimal-point in formatted monetary
 39985 quantities.

39986 **char *mon_grouping**
 39987 A string whose elements taken as one-byte integer values indicate the size of each group of
 39988 digits in formatted monetary quantities.

39989 **char *positive_sign**
 39990 The string used to indicate a non-negative valued formatted monetary quantity.

- 39991 **char *negative_sign**
39992 The string used to indicate a negative valued formatted monetary quantity.
- 39993 **char int_frac_digits**
39994 The number of fractional digits (those after the decimal-point) to be displayed in an
39995 internationally formatted monetary quantity.
- 39996 **char frac_digits**
39997 The number of fractional digits (those after the decimal-point) to be displayed in a
39998 formatted monetary quantity.
- 39999 **char p_cs_precedes**
40000 Set to 1 if the **currency_symbol** precedes the value for a non-negative formatted monetary
40001 quantity. Set to 0 if the symbol succeeds the value.
- 40002 **char p_sep_by_space**
40003 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
40004 value for a non-negative formatted monetary quantity.
- 40005 **char n_cs_precedes**
40006 Set to 1 if the **currency_symbol** precedes the value for a negative formatted monetary
40007 quantity. Set to 0 if the symbol succeeds the value.
- 40008 **char n_sep_by_space**
40009 Set to a value indicating the separation of the **currency_symbol**, the sign string, and the
40010 value for a negative formatted monetary quantity.
- 40011 **char p_sign_posn**
40012 Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted
40013 monetary quantity.
- 40014 **char n_sign_posn**
40015 Set to a value indicating the positioning of the **negative_sign** for a negative formatted
40016 monetary quantity.
- 40017 **char int_p_cs_precedes**
40018 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a non-
40019 negative internationally formatted monetary quantity.
- 40020 **char int_n_cs_precedes**
40021 Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a
40022 negative internationally formatted monetary quantity.
- 40023 **char int_p_sep_by_space**
40024 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
40025 value for a non-negative internationally formatted monetary quantity.
- 40026 **char int_n_sep_by_space**
40027 Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the
40028 value for a negative internationally formatted monetary quantity.
- 40029 **char int_p_sign_posn**
40030 Set to a value indicating the positioning of the **positive_sign** for a non-negative
40031 internationally formatted monetary quantity.
- 40032 **char int_n_sign_posn**
40033 Set to a value indicating the positioning of the **negative_sign** for a negative internationally
40034 formatted monetary quantity.
- 40035 The elements of **grouping** and **mon_grouping** are interpreted according to the following:

- 40036 {CHAR_MAX} No further grouping is to be performed.
- 40037 0 The previous element is to be repeatedly used for the remainder of the digits.
- 40038 *other* The integer value is the number of digits that comprise the current group. The
40039 next element is examined to determine the size of the next group of digits
40040 before the current group.

40041 The values of `p_sep_by_space`, `n_sep_by_space`, `int_p_sep_by_space`, and `int_n_sep_by_space`
40042 are interpreted according to the following:

- 40043 0 No space separates the currency symbol and value.
- 40044 1 If the currency symbol and sign string are adjacent, a space separates them from the value;
40045 otherwise, a space separates the currency symbol from the value.
- 40046 2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a
40047 space separates the sign string from the value.

40048 For `int_p_sep_by_space` and `int_n_sep_by_space`, the fourth character of `int_curr_symbol` is
40049 used instead of a space.

40050 The values of `p_sign_posn`, `n_sign_posn`, `int_p_sign_posn`, and `int_n_sign_posn` are
40051 interpreted according to the following:

- 40052 0 Parentheses surround the quantity and `currency_symbol` or `int_curr_symbol`.
- 40053 1 The sign string precedes the quantity and `currency_symbol` or `int_curr_symbol`.
- 40054 2 The sign string succeeds the quantity and `currency_symbol` or `int_curr_symbol`.
- 40055 3 The sign string immediately precedes the `currency_symbol` or `int_curr_symbol`.
- 40056 4 The sign string immediately succeeds the `currency_symbol` or `int_curr_symbol`.

40057 The implementation shall behave as if no function in this volume of POSIX.1-200x calls
40058 `localeconv()`.

40059 CX The `localeconv()` function need not be thread-safe. A function that is not required to be thread-
40060 safe is not required to be reentrant.

40061 RETURN VALUE

40062 The `localeconv()` function shall return a pointer to the filled-in object. The application shall not
40063 modify the structure pointed to by the return value which may be overwritten by a subsequent
40064 call to `localeconv()`. In addition, calls to `setlocale()` with the categories `LC_ALL`, `LC_MONETARY`,
40065 or `LC_NUMERIC` or calls to `uselocale()` which change the categories `LC_MONETARY` or
40066 `LC_NUMERIC` may overwrite the contents of the structure.

40067 ERRORS

40068 No errors are defined.

40069 EXAMPLES

40070 None.

40071 APPLICATION USAGE

40072 The following table illustrates the rules which may be used by four countries to format
40073 monetary quantities.

Country	Positive Format	Negative Format	International Format
Italy	L.1.230	-L.1.230	ITL.1.230
Netherlands	F 1.234,56	F -1.234,56	NLG 1.234,56
Norway	kr1.234,56	kr1.234,56-	NOK 1.234,56
Switzerland	SFrs.1,234.56	SFrs.1,234.56C	CHF 1,234.56

40079
40080

For these four countries, the respective values for the monetary members of the structure returned by *localeconv()* are:

40081
40082
40083
40084
40085
40086
40087
40088
40089
40090
40091
40092
40093
40094
40095
40096
40097
40098
40099
40100
40101
40102

	Italy	Netherlands	Norway	Switzerland
int_curr_symbol	"EUR "	"EUR "	"NOK "	"CHF "
currency_symbol	"€"	"€"	"kr"	"Fr."
mon_decimal_point	"."	","	","	"."
mon_thousands_sep	"."	"."	"."	","
mon_grouping	"\3"	"\3"	"\3"	"\3"
positive_sign	" "	" "	" "	" "
negative_sign	"-"	"-"	"-"	"C"
int_frac_digits	0	2	2	2
frac_digits	0	2	2	2
p_cs_precedes	1	1	1	1
p_sep_by_space	0	1	0	0
n_cs_precedes	1	1	1	1
n_sep_by_space	0	1	0	0
p_sign_posn	1	1	1	1
n_sign_posn	1	4	2	2
int_p_cs_precedes	1	1	1	1
int_n_cs_precedes	1	1	1	1
int_p_sep_by_space	0	0	0	0
int_n_sep_by_space	0	0	0	0
int_p_sign_posn	1	1	1	1
int_n_sign_posn	1	4	4	2

40103
40104**RATIONALE**

None.

40105
40106**FUTURE DIRECTIONS**

None.

40107
40108**SEE ALSO**

fprintf(), *fscanf()*, *isalpha()*, *isascii()*, *nl_langinfo()*, *setlocale()*, *strcat()*, *strchr()*, *strcmp()*, *strcoll()*, *strcpy()*, *strftime()*, *strlen()*, *strpbrk()*, *strspn()*, *strtok()*, *strxfrm()*, *strtod()*, *uselocale()*

40109
40110XBD [<langinfo.h>](#), [<locale.h>](#)40111
40112**CHANGE HISTORY**

First released in Issue 4. Derived from the ANSI C standard.

40113
40114**Issue 6**

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

40115

The RETURN VALUE section is rewritten to avoid use of the term "must".

40116

This reference page is updated for alignment with the ISO/IEC 9899:1999 standard.

40117

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

40118

40119

40120

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/31 is applied, removing references to **int_curr_symbol** and updating the descriptions of **p_sep_by_space** and **n_sep_by_space**. These changes are for alignment with the ISO C standard.

40121

Issue 7

40122

The definitions of **int_curr_symbol** and **currency_symbol** are updated.

40123 **NAME**

40124 localtime, localtime_r — convert a time value to a broken-down local time

40125 **SYNOPSIS**

40126 #include <time.h>

40127 struct tm *localtime(const time_t *timer);

40128 CX struct tm *localtime_r(const time_t *restrict timer,
40129 struct tm *restrict result);40130 **DESCRIPTION**40131 CX For *localtime()*: The functionality described on this reference page is aligned with the ISO C
40132 standard. Any conflict between the requirements described here and the ISO C standard is
40133 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.40134 The *localtime()* function shall convert the time in seconds since the Epoch pointed to by *timer*
40135 into a broken-down time, expressed as a local time. The function corrects for the timezone and
40136 any seasonal time adjustments. Local timezone information is used as though *localtime()* calls
40137 *tzset()*.40138 The relationship between a time in seconds since the Epoch used as an argument to *localtime()*
40139 and the **tm** structure (defined in the **<time.h>** header) is that the result shall be as specified in
40140 the expression given in the definition of seconds since the Epoch (see XBD Section 4.15, on page
40141 100) corrected for timezone and any seasonal time adjustments, where the names in the structure
40142 and in the expression correspond.40143 The same relationship shall apply for *localtime_r()*.40144 The *localtime()* function need not be thread-safe. A function that is not required to be thread-safe
40145 is not required to be reentrant.40146 The *asctime()*, *ctime()*, *gmtime()*, and *localtime()* functions shall return values in one of two static
40147 objects: a broken-down time structure and an array of type **char**. Execution of any of the
40148 functions may overwrite the information returned in either of these objects by any of the other
40149 functions.40150 The *localtime_r()* function shall convert the time in seconds since the Epoch pointed to by *timer*
40151 into a broken-down time stored in the structure to which *result* points. The *localtime_r()* function
40152 shall also return a pointer to that same structure.40153 Unlike *localtime()*, the reentrant version is not required to set *tzname*.40154 If the reentrant version does not set *tzname*, it shall not set *daylight* and shall not set *timezone*.40155 **RETURN VALUE**40156 Upon successful completion, the *localtime()* function shall return a pointer to the broken-down
40157 CX time structure. If an error is detected, *localtime()* shall return a null pointer and set *errno* to
40158 indicate the error.40159 Upon successful completion, *localtime_r()* shall return a pointer to the structure pointed to by the
40160 argument *result*. If an error is detected, *localtime_r()* shall return a null pointer and set *errno* to
40161 indicate the error.40162 **ERRORS**40163 CX The *localtime()* and *localtime_r()* functions shall fail if:

40164 CX [Eoverflow] The result cannot be represented.

40165 EXAMPLES

40166 Getting the Local Date and Time

40167 The following example uses the *time()* function to calculate the time elapsed, in seconds, since
40168 January 1, 1970 0:00 UTC (the Epoch), *localtime()* to convert that value to a broken-down time,
40169 and *asctime()* to convert the broken-down time values into a printable string.

```
40170 #include <stdio.h>
40171 #include <time.h>
40172 int main(void)
40173 {
40174     time_t result;
40175     result = time(NULL);
40176     printf("%s%ju secs since the Epoch\n",
40177           asctime(localtime(&result)),
40178           (uintmax_t)result);
40179     return(0);
40180 }
```

40181 This example writes the current time to *stdout* in a form like this:

```
40182 Wed Jun 26 10:32:15 1996
40183 835810335 secs since the Epoch
```

40184 Getting the Modification Time for a File

40185 The following example prints the last data modification timestamp in the local timezone for a
40186 given file.

```
40187 #include <stdio.h>
40188 #include <time.h>
40189 #include <sys/stat.h>
40190 int
40191 print_file_time(const char *filename)
40192 {
40193     struct stat statbuf;
40194     struct tm *tm;
40195     char timestr[BUFSIZ];
40196     if(stat(filename, &statbuf) == -1)
40197         return -1;
40198     if((tm = localtime(&statbuf.st_mtime)) == NULL)
40199         return -1;
40200     if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
40201         return -1;
40202     printf("%s: %s.%09ld\n", filename, timestr, statbuf.st_mtim.tv_nsec);
40203     return 0;
40204 }
```

40205 **Timing an Event**

40206 The following example gets the current time, converts it to a string using *localtime()* and
 40207 *asctime()*, and prints it to standard output using *fputs()*. It then prints the number of minutes to
 40208 an event being timed.

```
40209 #include <time.h>
40210 #include <stdio.h>
40211 ...
40212 time_t now;
40213 int minutes_to_event;
40214 ...
40215 time(&now);
40216 printf("The time is ");
40217 fputs(asctime(localtime(&now)), stdout);
40218 printf("There are still %d minutes to the event.\n",
40219     minutes_to_event);
40220 ...
```

40221 **APPLICATION USAGE**

40222 The *localtime_r()* function is thread-safe and returns values in a user-supplied buffer instead of
 40223 possibly using a static data area that may be overwritten by each call.

40224 **RATIONALE**

40225 None.

40226 **FUTURE DIRECTIONS**

40227 None.

40228 **SEE ALSO**

40229 *asctime()*, *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *mktime()*, *strftime()*, *strptime()*, *time*,
 40230 *tzset()*, *utime()*

40231 XBD Section 4.15 (on page 100), [<time.h>](#)

40232 **CHANGE HISTORY**

40233 First released in Issue 1. Derived from Issue 1 of the SVID.

40234 **Issue 5**

40235 A note indicating that the *localtime()* function need not be reentrant is added to the
 40236 DESCRIPTION.

40237 The *localtime_r()* function is included for alignment with the POSIX Threads Extension.

40238 **Issue 6**

40239 The *localtime_r()* function is marked as part of the Thread-Safe Functions option.

40240 Extensions beyond the ISO C standard are marked.

40241 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
 40242 its avoidance of possibly using a static data area.

40243 The **restrict** keyword is added to the *localtime_r()* prototype for alignment with the
 40244 ISO/IEC 9899:1999 standard.

40245 Examples are added.

40246 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/32 is applied, adding the
 40247 [EOverflow] error.

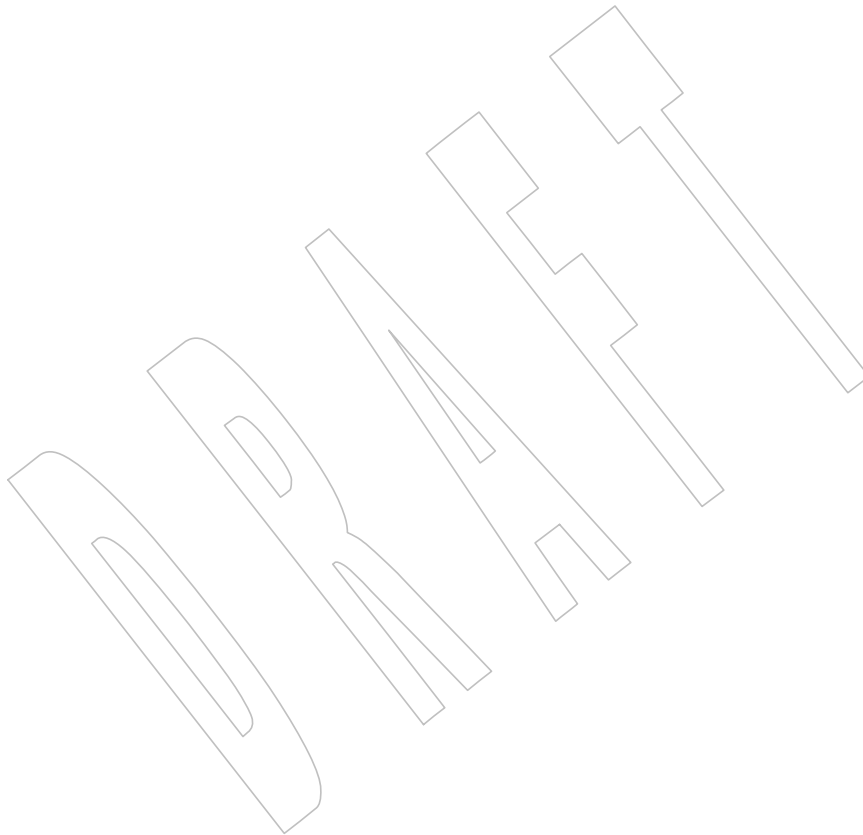
40248 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/55 is applied, updating the error handling
 40249 for *localtime_r()*.

localtime()

40250 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/56 is applied, adding a requirement that if
40251 *localtime_r()* does not set the *tzname* variable, it shall not set the *daylight* or *timezone* variables. On
40252 systems supporting XSI, the *daylight*, *timezone*, and *tzname* variables should all be set to provide
40253 information for the same timezone. This updates the description of *localtime_r()* to mention
40254 *daylight* and *timezone* as well as *tzname*. The SEE ALSO section is updated.

Issue 7

40255 The *localtime_r()* function is moved from the Thread-Safe Functions option to the Base.
40256 Changes are made to the EXAMPLES section related to support for finegrained timestamps. +
40257



40258 **NAME**
 40259 lockf — record locking on files

40260 **SYNOPSIS**

```
40261 XSI #include <unistd.h>
40262 int lockf(int fildes, int function, off_t size);
```

40263 **DESCRIPTION**

40264 The *lockf()* function shall lock sections of a file with advisory-mode locks. Calls to *lockf()* from
 40265 threads in other processes which attempt to lock the locked file section shall either return an
 40266 error value or block until the section becomes unlocked. All the locks for a process are removed
 40267 when the process terminates. Record locking with *lockf()* shall be supported for regular files and
 40268 may be supported for other files.

40269 The *fildes* argument is an open file descriptor. To establish a lock with this function, the file
 40270 descriptor shall be opened with write-only permission (O_WRONLY) or with read/write
 40271 permission (O_RDWR).

40272 The *function* argument is a control value which specifies the action to be taken. The permissible
 40273 values for *function* are defined in **<unistd.h>** as follows:

Function	Description
F_ULOCK	Unlock locked sections.
F_LOCK	Lock a section for exclusive use.
F_TLOCK	Test and lock a section for exclusive use.
F_TEST	Test a section for locks by other processes.

40274
 40275
 40276
 40277
 40278
 40279 F_TEST shall detect if a lock by another process is present on the specified section.

40280 F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

40281 F_ULOCK shall remove locks from a section of the file.

40282 The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be
 40283 locked or unlocked starts at the current offset in the file and extends forward for a positive size
 40284 or backward for a negative size (the preceding bytes up to but not including the current offset).
 40285 If *size* is 0, the section from the current offset through the largest possible file offset shall be
 40286 locked (that is, from the current offset through the present or any future end-of-file). An area
 40287 need not be allocated to the file to be locked because locks may exist past the end-of-file.

40288 The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained
 40289 by a previously locked section for the same process. When this occurs, or if adjacent locked
 40290 sections would occur, the sections shall be combined into a single locked section. If the request
 40291 would cause the number of locks to exceed a system-imposed limit, the request shall fail.

40292 F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available.
 40293 F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the
 40294 function to fail if the section is already locked by another process.

40295 File locks shall be released on first close by the locking process of any file descriptor for the file.

40296 F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the
 40297 process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or
 40298 to the end-of-file if *size* is (off_t)0. When all of a locked section is not released (that is, when the
 40299 beginning or end of the area to be unlocked falls within a locked section), the remaining portions
 40300 of that section shall remain locked by the process. Releasing the center portion of a locked

lockf()

40301 section shall cause the remaining locked beginning and end portions to become two separate
 40302 locked sections. If the request would cause the number of locks in the system to exceed a system-
 40303 imposed limit, the request shall fail.

40304 A potential for deadlock occurs if the threads of a process controlling a locked section are
 40305 blocked by accessing a locked section of another process. If the system detects that deadlock
 40306 would occur, *lockf()* shall fail with an [EDEADLK] error.

40307 The interaction between *fcntl()* and *lockf()* locks is unspecified.

40308 Blocking on a section shall be interrupted by any signal.

40309 An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested
 40310 section is the maximum value for an object of type **off_t**, when the process has an existing lock
 40311 in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a
 40312 request to unlock from the start of the requested section with a size equal to 0. Otherwise, an
 40313 F_ULOCK request shall attempt to unlock only the requested section.

40314 Attempting to lock a section of a file that is associated with a buffered stream produces
 40315 unspecified results.

RETURN VALUE

40316 Upon successful completion, *lockf()* shall return 0. Otherwise, it shall return -1, set *errno* to
 40317 indicate an error, and existing locks shall not be changed.

ERRORS

40319 The *lockf()* function shall fail if:

40321 [EBADF] The *fildev* argument is not a valid open file descriptor; or *function* is F_LOCK
 40322 and F_TLOCK and *fildev* is not a valid file descriptor open for writing.

40323 [EACCES] or [EAGAIN]
 40324 The *function* argument is F_TLOCK or F_TEST and the section is already
 40325 locked by another process.

40326 [EDEADLK] The *function* argument is F_LOCK and a deadlock is detected.

40327 [EINTR] A signal was caught during execution of the function.

40328 [EINVAL] The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or
 40329 F_ULOCK; or *size* plus the current file offset is less than 0.

40330 [EOVERFLOW] The offset of the first, or if *size* is not 0 then the last, byte in the requested
 40331 section cannot be represented correctly in an object of type **off_t**.

40332 The *lockf()* function may fail if:

40333 [EAGAIN] The *function* argument is F_LOCK or F_TLOCK and the file is mapped with
 40334 *mmap()*.

40335 [EDEADLK] or [ENOLCK]
 40336 The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request
 40337 would cause the number of locks to exceed a system-imposed limit.

40338 [EOPNOTSUPP] or [EINVAL]
 40339 The implementation does not support the locking of files of the type indicated
 40340 by the *fildev* argument.

EXAMPLES**Locking a Portion of a File**

In the following example, a file named `/home/cnd/mod1` is being modified. Other processes that use locking are prevented from changing it during this process. Only the first 10 000 bytes are locked, and the lock call fails if another process has any part of this area locked already.

```
#include <fcntl.h>
#include <unistd.h>

int fildes;
int status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fildes, F_TLOCK, (off_t)10000);
```

APPLICATION USAGE

Record-locking should not be used in combination with the `fopen()`, `fread()`, `fwrite()`, and other `stdio` functions. Instead, the more primitive, non-buffered functions (such as `open()`) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The `stdio` functions are the most common source of unexpected buffering.

The `alarm()` function may be used to provide a timeout facility in applications requiring it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*alarm\(\)*](#), [*chmod*](#), [*close\(\)*](#), [*creat\(\)*](#), [*fcntl\(\)*](#), [*fopen\(\)*](#), [*mmap\(\)*](#), [*open\(\)*](#), [*read*](#), [*write*](#)

XBD [*<unistd.h>*](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Large File Summit extensions are added. In particular, the description of [EINVAL] is clarified and moved from optional to mandatory status.

A note is added to the DESCRIPTION indicating the effects of attempting to lock a section of a file that is associated with a buffered stream.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

Austin Group Interpretation 1003.1-2001 #054 is applied, updating the DESCRIPTION.

40379 **NAME**

40380 log, logf, logl — natural logarithm function

40381 **SYNOPSIS**

```
40382 #include <math.h>
40383
40383 double log(double x);
40384 float logf(float x);
40385 long double logl(long double x);
```

40386 **DESCRIPTION**

40387 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40388 conflict between the requirements described here and the ISO C standard is unintentional. This
 40389 volume of POSIX.1-200x defers to the ISO C standard.

40390 These functions shall compute the natural logarithm of their argument x , $\log_e(x)$.

40391 An application wishing to check for error situations should set *errno* to zero and call
 40392 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40393 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40394 zero, an error has occurred.

40395 **RETURN VALUE**

40396 Upon successful completion, these functions shall return the natural logarithm of x .

40397 If x is ± 0 , a pole error shall occur and *log()*, *logf()*, and *logl()* shall return `-HUGE_VAL`,
 40398 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

40399 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 40400 NaN (if supported), or an implementation-defined value shall be returned.

40401 MX If x is NaN, a NaN shall be returned.

40402 If x is 1, `+0` shall be returned.

40403 If x is `+Inf`, x shall be returned.

40404 **ERRORS**

40405 These functions shall fail if:

40406 MX Domain Error The finite value of x is negative, or x is `-Inf`.

40407 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40408 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40409 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40410 shall be raised.

40411 Pole Error The value of x is zero.

40412 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40413 then *errno* shall be set to [ERANGE]. If the integer expression
 40414 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 40415 floating-point exception shall be raised.

40416
40417
40418
40419
40420
40421
40422
40423
40424
40425
40426
40427
40428
40429
40430
40431
40432
40433
40434
40435
40436
40437
40438
40439

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp(), *feclearexcept()*, *fetestexcept()*, *isnan()*, *log10()*, *log1p()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The *logf()* and *logl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

40440 **NAME**
 40441 `log10`, `log10f`, `log10l` — base 10 logarithm function

40442 **SYNOPSIS**
 40443 `#include <math.h>`
 40444 `double log10(double x);`
 40445 `float log10f(float x);`
 40446 `long double log10l(long double x);`

40447 **DESCRIPTION**

40448 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40449 conflict between the requirements described here and the ISO C standard is unintentional. This
 40450 volume of POSIX.1-200x defers to the ISO C standard.

40451 These functions shall compute the base 10 logarithm of their argument x , $\log_{10}(x)$.

40452 An application wishing to check for error situations should set *errno* to zero and call
 40453 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40454 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40455 zero, an error has occurred.

40456 **RETURN VALUE**

40457 Upon successful completion, these functions shall return the base 10 logarithm of x .

40458 If x is ± 0 , a pole error shall occur and *log10()*, *log10f()*, and *log10l()* shall return `-HUGE_VAL`,
 40459 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

40460 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 40461 NaN (if supported), or an implementation-defined value shall be returned.

40462 MX If x is NaN, a NaN shall be returned.

40463 If x is 1, `+0` shall be returned.

40464 If x is `+Inf`, `+Inf` shall be returned.

40465 **ERRORS**

40466 These functions shall fail if:

40467 MX Domain Error The finite value of x is negative, or x is `-Inf`.

40468 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40469 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40470 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40471 shall be raised.

40472 Pole Error The value of x is zero.

40473 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40474 then *errno* shall be set to [ERANGE]. If the integer expression
 40475 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 40476 floating-point exception shall be raised.

40477
40478
40479
40480
40481
40482
40483
40484
40485
40486
40487
40488
40489
40490
40491
40492
40493
40494
40495
40496
40497
40498
40499
40500
40501

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *isnan()*, *log()*, *pow()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The *log10f()* and *log10l()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

40502 **NAME**

40503 log1p, log1pf, log1pl — compute a natural logarithm

40504 **SYNOPSIS**

```
40505 #include <math.h>
40506
40506 double log1p(double x);
40507 float log1pf(float x);
40508 long double log1pl(long double x);
```

40509 **DESCRIPTION**

40510 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40511 conflict between the requirements described here and the ISO C standard is unintentional. This
 40512 volume of POSIX.1-200x defers to the ISO C standard.

40513 These functions shall compute $\log_e(1.0 + x)$.

40514 An application wishing to check for error situations should set *errno* to zero and call
 40515 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40516 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40517 zero, an error has occurred.

40518 **RETURN VALUE**40519 Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

40520 If x is -1 , a pole error shall occur and *log1p()*, *log1pf()*, and *log1pl()* shall return $-\text{HUGE_VAL}$,
 40521 $-\text{HUGE_VALF}$, and $-\text{HUGE_VALL}$, respectively.

40522 MX For finite values of x that are less than -1 , or if x is $-\text{Inf}$, a domain error shall occur, and either a
 40523 NaN (if supported), or an implementation-defined value shall be returned.

40524 MX If x is NaN, a NaN shall be returned.40525 If x is ± 0 , or $+\text{Inf}$, x shall be returned.40526 If x is subnormal, a range error may occur and x should be returned.40527 **ERRORS**

40528 These functions shall fail if:

40529 MX Domain Error The finite value of x is less than -1 , or x is $-\text{Inf}$.

40530 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40531 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40532 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40533 shall be raised.

40534 Pole Error The value of x is -1 .

40535 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40536 then *errno* shall be set to [ERANGE]. If the integer expression
 40537 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 40538 floating-point exception shall be raised.

40539 These functions may fail if:

40540 MX Range Error The value of x is subnormal.

40541 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40542 then *errno* shall be set to [ERANGE]. If the integer expression
 40543 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

floating-point exception shall be raised.

40544

EXAMPLES

40545

None.

40546

APPLICATION USAGE

40547

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

40548

40549

RATIONALE

40550

None.

40551

FUTURE DIRECTIONS

40552

None.

40553

SEE ALSO

40554

feclearexcept(), *fetestexcept()*, *log()*

40555

XBD Section 4.19 (on page 104), [<math.h>](#)

40556

CHANGE HISTORY

40557

First released in Issue 4, Version 2.

40558

Issue 5

40559

Moved from X/OPEN UNIX extension to BASE.

40560

Issue 6

40561

The normative text is updated to avoid use of the term “must” for application requirements.

40562

The *log1p()* function is no longer marked as an extension.

40563

The *log1pf()* and *log1pl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

40564

40565

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

40566

40567

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

40568

40569

40570 **NAME**
 40571 `log2, log2f, log2l` — compute base 2 logarithm functions

40572 **SYNOPSIS**
 40573 `#include <math.h>`
 40574 `double log2(double x);`
 40575 `float log2f(float x);`
 40576 `long double log2l(long double x);`

40577 **DESCRIPTION**
 40578 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40579 conflict between the requirements described here and the ISO C standard is unintentional. This
 40580 volume of POSIX.1-200x defers to the ISO C standard.

40581 These functions shall compute the base 2 logarithm of their argument x , $\log_2(x)$.
 40582 An application wishing to check for error situations should set *errno* to zero and call
 40583 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40584 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40585 zero, an error has occurred.

40586 **RETURN VALUE**
 40587 Upon successful completion, these functions shall return the base 2 logarithm of x .
 40588 If x is ± 0 , a pole error shall occur and *log2()*, *log2f()*, and *log2l()* shall return `-HUGE_VAL`,
 40589 `-HUGE_VALF`, and `-HUGE_VALL`, respectively.

40590 MX For finite values of x that are less than 0, or if x is `-Inf`, a domain error shall occur, and either a
 40591 NaN (if supported), or an implementation-defined value shall be returned.

40592 MX If x is NaN, a NaN shall be returned.

40593 If x is 1, +0 shall be returned.

40594 If x is `+Inf`, x shall be returned.

40595 **ERRORS**
 40596 These functions shall fail if:

40597 MX Domain Error The finite value of x is less than zero, or x is `-Inf`.
 40598 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40599 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40600 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40601 shall be raised.

40602 Pole Error The value of x is zero.
 40603 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40604 then *errno* shall be set to [ERANGE]. If the integer expression
 40605 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 40606 floating-point exception shall be raised.

40607
40608
40609
40610
40611
40612
40613
40614
40615
40616
40617
40618
40619
40620

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *log()*

XBD Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

DRAFT

40621 **NAME**
 40622 `logb`, `logbf`, `logbl` — radix-independent exponent

40623 **SYNOPSIS**
 40624 `#include <math.h>`
 40625 `double logb(double x);`
 40626 `float logbf(float x);`
 40627 `long double logbl(long double x);`

40628 DESCRIPTION

40629 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40630 conflict between the requirements described here and the ISO C standard is unintentional. This
 40631 volume of POSIX.1-200x defers to the ISO C standard.

40632 These functions shall compute the exponent of x , which is the integral part of $\log_r |x|$, as a
 40633 signed floating-point value, for non-zero x , where r is the radix of the machine's floating-point
 40634 arithmetic, which is the value of `FLT_RADIX` defined in the `<float.h>` header.

40635 If x is subnormal it is treated as though it were normalized; thus for finite positive x :

40636
$$1 \leq x * FLT_RADIX^{-\logb(x)} < FLT_RADIX$$

40637 An application wishing to check for error situations should set `errno` to zero and call
 40638 `feclearexcept(FE_ALL_EXCEPT)` before calling these functions. On return, if `errno` is non-zero or
 40639 `fetestexcept(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW)` is non-
 40640 zero, an error has occurred.

40641 RETURN VALUE

40642 Upon successful completion, these functions shall return the exponent of x .

40643 If x is ± 0 , `logb()`, `logbf()`, and `logbl()` shall return `-HUGE_VAL`, `-HUGE_VALF`, and
 40644 MX `-HUGE_VALL`, respectively. On systems that support the IEC 60559 Floating-Point option, a
 40645 pole error shall occur;
 40646 CX otherwise, a pole error may occur.

40647 MX If x is NaN, a NaN shall be returned.

40648 MX If x is $\pm\text{Inf}$, $+\text{Inf}$ shall be returned.

40649 ERRORS

40650 These functions shall fail if:

40651 MX **Pole Error** The value of x is ± 0 .
 40652 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 40653 then `errno` shall be set to `[ERANGE]`. If the integer expression
 40654 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the divide-by-zero
 40655 floating-point exception shall be raised.

40656 These functions may fail if:

40657 **Pole Error** The value of x is 0.
 40658 If the integer expression `(math_errhandling & MATH_ERRNO)` is non-zero,
 40659 then `errno` shall be set to `[ERANGE]`. If the integer expression
 40660 `(math_errhandling & MATH_ERREXCEPT)` is non-zero, then the divide-by-zero
 40661 floating-point exception shall be raised.

40662
40663
40664
40665
40666
40667
40668
40669
40670
40671
40672
40673
40674
40675
40676
40677
40678
40679
40680
40681
40682
40683
40684
40685
40686

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *ilogb()*, *scalbln()*

XBD Section 4.19 (on page 104), `<float.h>`, `<math.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The *logb()* function is no longer marked as an extension.

The *logbf()* and *logbl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

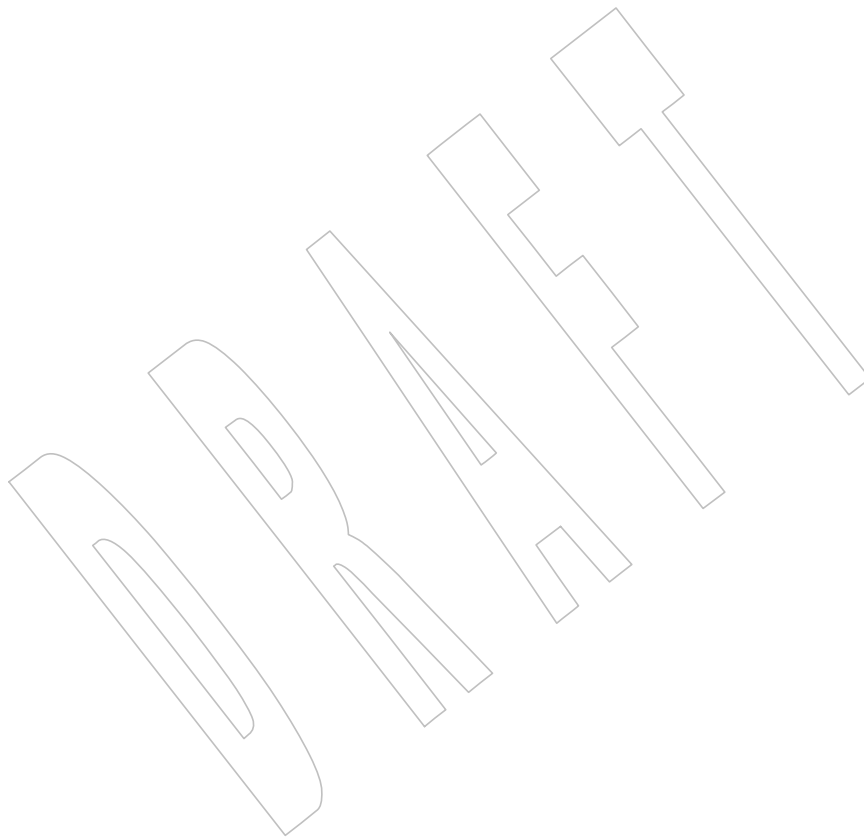
Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #50 (SD5-XSH-ERN-76) is applied.

40687 **NAME**
40688 `logf, logl` — natural logarithm function

40689 **SYNOPSIS**
40690 `#include <math.h>`
40691 `float logf(float x);`
40692 `long double logl(long double x);`

40693 **DESCRIPTION**
40694 Refer to *log()*.



40695 **NAME**40696 `longjmp` — non-local goto40697 **SYNOPSIS**40698 `#include <setjmp.h>`40699 `void longjmp(jmp_buf env, int val);`40700 **DESCRIPTION**

40701 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40702 conflict between the requirements described here and the ISO C standard is unintentional. This
 40703 volume of POSIX.1-200x defers to the ISO C standard.

40704 The *longjmp()* function shall restore the environment saved by the most recent invocation of
 40705 *setjmp()* in the same thread, with the corresponding **jmp_buf** argument. If there is no such
 40706 invocation, or if the function containing the invocation of *setjmp()* has terminated execution in
 40707 the interim, or if the invocation of *setjmp()* was within the scope of an identifier with variably
 40708 CX modified type and execution has left that scope in the interim, the behavior is undefined. It is
 40709 unspecified whether *longjmp()* restores the signal mask, leaves the signal mask unchanged, or
 40710 restores it to its value at the time *setjmp()* was called.

40711 All accessible objects have values, and all other components of the abstract machine have state
 40712 (for example, floating-point status flags and open files), as of the time *longjmp()* was called,
 40713 except that the values of objects of automatic storage duration are unspecified if they meet all
 40714 the following conditions:

- 40715 • They are local to the function containing the corresponding *setjmp()* invocation.
- 40716 • They do not have volatile-qualified type.
- 40717 • They are changed between the *setjmp()* invocation and *longjmp()* call.

40718 CX As it bypasses the usual function call and return mechanisms, *longjmp()* shall execute correctly
 40719 in contexts of interrupts, signals, and any of their associated functions. However, if *longjmp()* is
 40720 invoked from a nested signal handler (that is, from a function invoked as a result of a signal
 40721 raised during the handling of another signal), the behavior is undefined.

40722 The effect of a call to *longjmp()* where initialization of the **jmp_buf** structure was not performed
 40723 in the calling thread is undefined.

40724 **RETURN VALUE**

40725 After *longjmp()* is completed, program execution continues as if the corresponding invocation of
 40726 *setjmp()* had just returned the value specified by *val*. The *longjmp()* function shall not cause
 40727 *setjmp()* to return 0; if *val* is 0, *setjmp()* shall return 1.

40728 **ERRORS**

40729 No errors are defined.

40730 **EXAMPLES**

40731 None.

40732 **APPLICATION USAGE**

40733 Applications whose behavior depends on the value of the signal mask should not use *longjmp()*
 40734 and *setjmp()*, since their effect on the signal mask is unspecified, but should instead use the
 40735 *siglongjmp()* and *sigsetjmp()* functions (which can save and restore the signal mask under
 40736 application control).

longjmp()

40737

RATIONALE

40738

None.

40739

FUTURE DIRECTIONS

40740

None.

40741

SEE ALSO

40742

setjmp(), *sigaction()*, *siglongjmp()*, *sigsetjmp()*

40743

XBD <*setjmp.h*>

40744

CHANGE HISTORY

40745

First released in Issue 1. Derived from Issue 1 of the SVID.

40746

Issue 5

40747

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

40748

Issue 6

40749

Extensions beyond the ISO C standard are marked.

40750

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

40751

- The DESCRIPTION now explicitly makes *longjmp()*'s effect on the signal mask unspecified.

40752

40753

40754

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

40755 **NAME**
40756 lrand48 — generate uniformly distributed pseudo-random non-negative long integers

40757 **SYNOPSIS**

```
40758 XSI #include <stdlib.h>  
40759 long lrand48(void);
```

40760 **DESCRIPTION**

40761 Refer to *drand48()*.

40762 **NAME**40763 `lrint, lrintf, lrintl` — round to nearest integer value using current rounding direction40764 **SYNOPSIS**

```
40765 #include <math.h>
40766
40766 long lrint(double x);
40767 long lrintf(float x);
40768 long lrintl(long double x);
```

40769 **DESCRIPTION**

40770 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40771 conflict between the requirements described here and the ISO C standard is unintentional. This
 40772 volume of POSIX.1-200x defers to the ISO C standard.

40773 These functions shall round their argument to the nearest integer value, rounding according to
 40774 the current rounding direction.

40775 An application wishing to check for error situations should set *errno* to zero and call
 40776 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40777 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40778 zero, an error has occurred.

40779 **RETURN VALUE**

40780 Upon successful completion, these functions shall return the rounded integer value.

40781 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.40782 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.40783 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

40784 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be
 40785 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall
 40786 occur;

40787 CX otherwise, a domain error may occur.

40788 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be
 40789 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall
 40790 occur;

40791 CX otherwise, a domain error may occur.

40792 **ERRORS**

40793 These functions shall fail if:

40794 MX **Domain Error** The *x* argument is NaN or ±Inf, or the correct value is not representable as an
 40795 integer.

40796 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40797 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40798 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40799 shall be raised.

40800 These functions may fail if:

40801 **Domain Error** The correct value is not representable as an integer.

40802 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40803 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40804 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

40805 shall be raised.

40806 **EXAMPLES**

40807 None.

40808 **APPLICATION USAGE**

40809 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
40810 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

40811 **RATIONALE**

40812 These functions provide floating-to-integer conversions. They round according to the current
40813 rounding direction. If the rounded value is outside the range of the return type, the numeric
40814 result is unspecified and the invalid floating-point exception is raised. When they raise no other
40815 floating-point exception and the result differs from the argument, they raise the inexact floating-
40816 point exception.

40817 **FUTURE DIRECTIONS**

40818 None.

40819 **SEE ALSO**

40820 *feclearexcept()*, *fetestexcept()*, *llrint()*

40821 XBD Section 4.19 (on page 104), <math.h>

40822 **CHANGE HISTORY**

40823 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

40824 **Issue 7**

40825 ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #53 (SD5-XSH-ERN-77) is applied.

40826 **NAME**
 40827 `lround, lroundf, lroundl` — round to nearest integer value

40828 **SYNOPSIS**
 40829 `#include <math.h>`
 40830 `long lround(double x);`
 40831 `long lroundf(float x);`
 40832 `long lroundl(long double x);`

40833 **DESCRIPTION**
 40834 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 40835 conflict between the requirements described here and the ISO C standard is unintentional. This
 40836 volume of POSIX.1-200x defers to the ISO C standard.

40837 These functions shall round their argument to the nearest integer value, rounding halfway cases
 40838 away from zero, regardless of the current rounding direction.

40839 An application wishing to check for error situations should set *errno* to zero and call
 40840 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 40841 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 40842 zero, an error has occurred.

40843 **RETURN VALUE**
 40844 Upon successful completion, these functions shall return the rounded integer value.

40845 MX If *x* is NaN, a domain error shall occur and an unspecified value is returned.

40846 If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

40847 If *x* is -Inf, a domain error shall occur and an unspecified value is returned.

40848 If the correct value is positive and too large to represent as a **long**, an unspecified value shall be
 40849 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;
 40850 CX otherwise, a **domain** error may occur.

40851 If the correct value is negative and too large to represent as a **long**, an unspecified value shall be
 40852 MX returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur;
 40853 CX otherwise, a **domain** error may occur.

40854 **ERRORS**
 40855 These functions shall fail if:

40856 MX **Domain Error** The *x* argument is NaN or \pm Inf, or the correct value is not representable as an
 40857 integer.

40858 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40859 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40860 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40861 shall be raised.

40862 These functions may fail if:

40863 **Domain Error** The correct value is not representable as an integer.

40864 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 40865 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 40866 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 40867 shall be raised.

40868

EXAMPLES

40869

None.

40870

APPLICATION USAGE

40871

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

40872

40873

RATIONALE

40874

These functions differ from the *lrint()* functions in the default rounding direction, with the *lround()* functions rounding halfway cases away from zero and needing not to raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

40875

40876

40877

40878

FUTURE DIRECTIONS

40879

None.

40880

SEE ALSO

40881

[*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*llround\(\)*](#)

40882

XBD [Section 4.19](#) (on page 104), [<math.h>](#)

40883

CHANGE HISTORY

40884

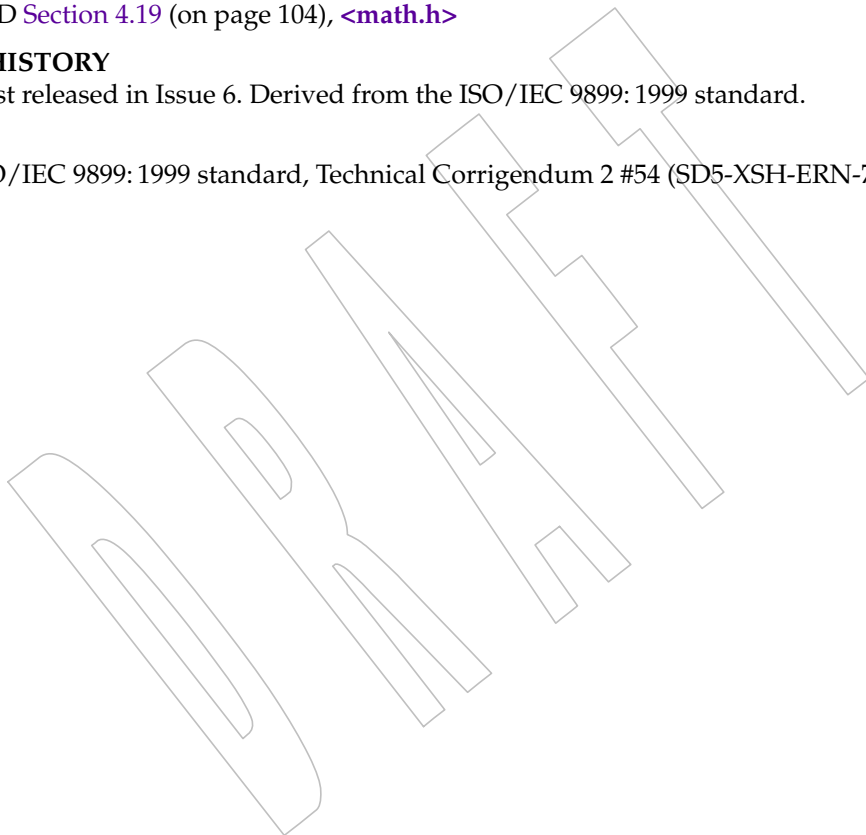
First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

40885

Issue 7

40886

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #54 (SD5-XSH-ERN-78) is applied.



40887 **NAME**

40888 lsearch, lfind — linear search and update

40889 **SYNOPSIS**

```
40890 XSI #include <search.h>
40891 void *lsearch(const void *key, void *base, size_t *nel, size_t width,
40892             int (*compar)(const void *, const void *));
40893 void *lfind(const void *key, const void *base, size_t *nel,
40894            size_t width, int (*compar)(const void *, const void *));
```

40895 **DESCRIPTION**

40896 The *lsearch()* function shall linearly search the table and return a pointer into the table for the
 40897 matching entry. If the entry does not occur, it shall be added at the end of the table. The *key*
 40898 argument points to the entry to be sought in the table. The *base* argument points to the first
 40899 element in the table. The *width* argument is the size of an element in bytes. The *nel* argument
 40900 points to an integer containing the current number of elements in the table. The integer to which
 40901 *nel* points shall be incremented if the entry is added to the table. The *compar* argument points to
 40902 a comparison function which the application shall supply (for example, *strcmp()*). It is called
 40903 with two arguments that point to the elements being compared. The application shall ensure
 40904 that the function returns 0 if the elements are equal, and non-zero otherwise.

40905 The *lfind()* function shall be equivalent to *lsearch()*, except that if the entry is not found, it is not
 40906 added to the table. Instead, a null pointer is returned.

40907 **RETURN VALUE**

40908 If the searched for entry is found, both *lsearch()* and *lfind()* shall return a pointer to it.
 40909 Otherwise, *lfind()* shall return a null pointer and *lsearch()* shall return a pointer to the newly
 40910 added element.

40911 Both functions shall return a null pointer in case of error.

40912 **ERRORS**

40913 No errors are defined.

40914 **EXAMPLES**40915 **Storing Strings in a Table**

40916 This fragment reads in less than or equal to TABSIZE strings of length less than or equal to
 40917 ELSIZE and stores them in a table, eliminating duplicates.

```
40918 #include <stdio.h>
40919 #include <string.h>
40920 #include <search.h>
40921 #define TABSIZE 50
40922 #define ELSIZE 120
40923 ...
40924     char line[ELSIZE], tab[TABSIZE][ELSIZE];
40925     size_t nel = 0;
40926     ...
40927     while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
40928         (void) lsearch(line, tab, &nel,
40929                      ELSIZE, (int (*)(const void *, const void *)) strcmp);
40930     ...
```

```

40931     Finding a Matching Entry
40932     The following example finds any line that reads "This is a test.".
40933     #include <search.h>
40934     #include <string.h>
40935     ...
40936     char line[ELSIZE], tab[TABSIZE][ELSIZE];
40937     size_t nel = 0;
40938     char *findline;
40939     void *entry;
40940
40941     findline = "This is a test.\n";
40942     entry = lfind(findline, tab, &nel, ELSIZE, (
40943         int (*)(const void *, const void *)) strcmp);

```

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate(), *tdelete()*

XBD [<search.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

40958
40959
40960
40961
40962
40963
40964
40965
40966
40967
40968
40969
40970
40971
40972
40973
40974
40975
40976
40977
40978
40979
40980
40981
40982
40983
40984
40985
40986
40987
40988
40989
40990
40991
40992
40993
40994
40995
40996
40997
40998

NAME

`lseek` — move the read/write file offset

SYNOPSIS

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPTION

The `lseek()` function shall set the file offset for the open file description associated with the file descriptor `fildes`, as follows:

- If `whence` is `SEEK_SET`, the file offset shall be set to `offset` bytes.
- If `whence` is `SEEK_CUR`, the file offset shall be set to its current location plus `offset`.
- If `whence` is `SEEK_END`, the file offset shall be set to the size of the file plus `offset`.

The symbolic constants `SEEK_SET`, `SEEK_CUR`, and `SEEK_END` are defined in `<unistd.h>`.

The behavior of `lseek()` on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The `lseek()` function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The `lseek()` function shall not, by itself, extend the size of a file.

SHM If `fildes` refers to a shared memory object, the result of the `lseek()` function is unspecified.

TYM If `fildes` refers to a typed memory object, the result of the `lseek()` function is unspecified.

RETURN VALUE

Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, `(off_t)-1` shall be returned, `errno` shall be set to indicate the error, and the file offset shall remain unchanged.

ERRORS

The `lseek()` function shall fail if:

- | | |
|-------------|--|
| [EBADF] | The <code>fildes</code> argument is not an open file descriptor. |
| [EINVAL] | The <code>whence</code> argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory. |
| [EOVERFLOW] | The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> . |
| [ESPIPE] | The <code>fildes</code> argument is associated with a pipe, FIFO, or socket. |

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the functions `fgetpos()` and `fsetpos()`, which work on very large files by use of a special positioning type.

Although `lseek()` may position the file offset beyond the end of the file, this function does not itself extend the size of the file. While the only function in POSIX.1-200x that may directly

40999 extend the size of the file is *write()*, *truncate()*, and *ftruncate()*, several functions originally
 41000 derived from the ISO C standard, such as *fwrite()*, *fprintf()*, and so on, may do so (by causing
 41001 calls on *write()*).

41002 An invalid file offset that would cause [EINVAL] to be returned may be both implementation-
 41003 defined and device-dependent (for example, memory may have few invalid values). A negative
 41004 file offset may be valid for some devices in some implementations.

41005 The POSIX.1-1990 standard did not specifically prohibit *lseek()* from returning a negative offset.
 41006 Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return
 41007 to determine whether a return value of (*off_t*)-1 is a negative offset or an indication of an error
 41008 condition. The standard developers did not wish to require this action on the part of a
 41009 conforming application, and chose to require that *errno* be set to [EINVAL] when the resulting
 41010 file offset would be negative for a regular file, block special file, or directory.

41011 FUTURE DIRECTIONS

41012 None.

41013 SEE ALSO

41014 *open()*

41015 XBD <*sys/types.h*>, <*unistd.h*>

41016 CHANGE HISTORY

41017 First released in Issue 1. Derived from Issue 1 of the SVID.

41018 Issue 5

41019 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension.

41020 Large File Summit extensions are added.

41021 Issue 6

41022 In the SYNOPSIS, the optional include of the <*sys/types.h*> header is removed.

41023 The following new requirements on POSIX implementations derive from alignment with the
 41024 Single UNIX Specification:

- 41025 • The requirement to include <*sys/types.h*> has been removed. Although <*sys/types.h*> was
 41026 required for conforming implementations of previous POSIX specifications, it was not
 41027 required for UNIX applications.
- 41028 • The [E_OVERFLOW] error condition is added. This change is to support large files.

41029 An additional [ESPIPE] error condition is added for sockets.

41030 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
 41031 *lseek()* results are unspecified for typed memory objects.

lstat()41032 **NAME**41033 `lstat` — get file status41034 **SYNOPSIS**41035 `#include <sys/stat.h>`41036 `int lstat(const char *restrict path, struct stat *restrict buf);`41037 **DESCRIPTION**41038 Refer to *fstatat()*.

DRAFT

41039 **NAME**

41040 malloc — a memory allocator

41041 **SYNOPSIS**

41042 #include <stdlib.h>

41043 void *malloc(size_t size);

41044 **DESCRIPTION**

41045 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41046 conflict between the requirements described here and the ISO C standard is unintentional. This
 41047 volume of POSIX.1-200x defers to the ISO C standard.

41048 The *malloc()* function shall allocate unused space for an object whose size in bytes is specified by
 41049 *size* and whose value is unspecified.

41050 The order and contiguity of storage allocated by successive calls to *malloc()* is unspecified. The
 41051 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 41052 a pointer to any type of object and then used to access such an object in the space allocated (until
 41053 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 41054 disjoint from any other object. The pointer returned points to the start (lowest byte address) of
 41055 the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of
 41056 the space requested is 0, the behavior is implementation-defined: the value returned shall be
 41057 either a null pointer or a unique pointer.

41058 **RETURN VALUE**

41059 Upon successful completion with *size* not equal to 0, *malloc()* shall return a pointer to the
 41060 allocated space. If *size* is 0, either a null pointer or a unique pointer that can be successfully
 41061 CX passed to *free()* shall be returned. Otherwise, it shall return a null pointer and set *errno* to
 41062 indicate the error.

41063 **ERRORS**41064 The *malloc()* function shall fail if:

41065 CX [ENOMEM] Insufficient storage space is available.

41066 **EXAMPLES**

41067 None.

41068 **APPLICATION USAGE**

41069 None.

41070 **RATIONALE**

41071 None.

41072 **FUTURE DIRECTIONS**

41073 None.

41074 **SEE ALSO**41075 *alphasort()*, *calloc()*, *fmemopen()*, *free()*, *realloc()*, *strdup()*

41076 XBD <stdlib.h>

41077 **CHANGE HISTORY**

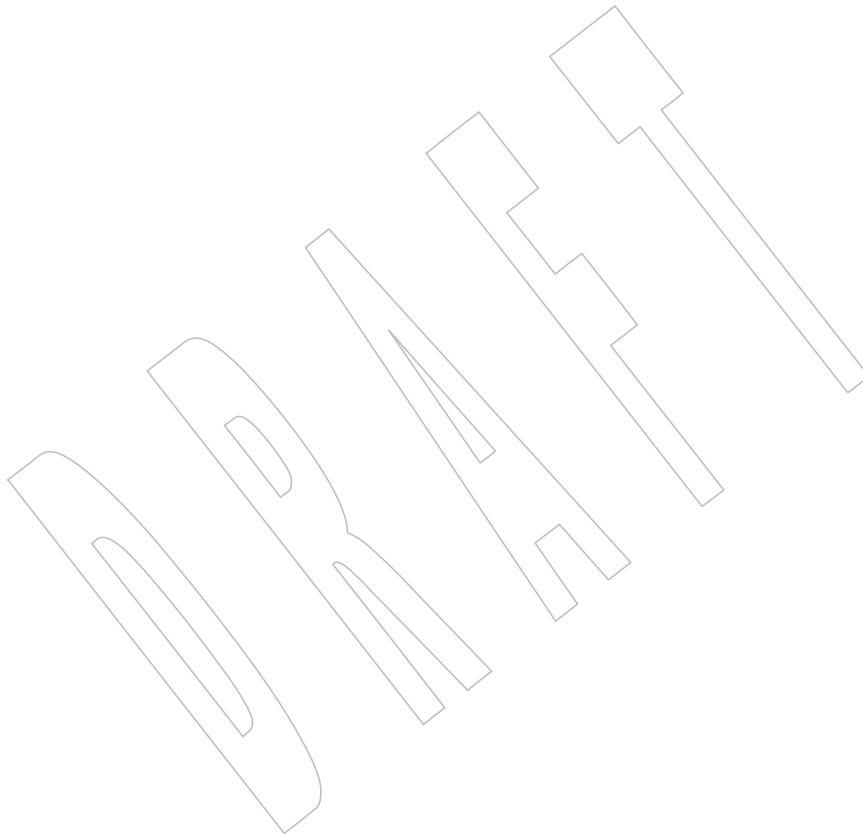
41078 First released in Issue 1. Derived from Issue 1 of the SVID.

41079
41080
41081
41082
41083
41084**Issue 6**

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, the requirement to set *errno* to indicate an error is added.
- The [ENOMEM] error condition is added.



41085 **NAME**
 41086 `mblen` — get number of bytes in a character

41087 **SYNOPSIS**
 41088 `#include <stdlib.h>`
 41089 `int mblen(const char *s, size_t n);`

41090 **DESCRIPTION**

41091 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41092 conflict between the requirements described here and the ISO C standard is unintentional. This
 41093 volume of POSIX.1-200x defers to the ISO C standard.

41094 If *s* is not a null pointer, *mblen()* shall determine the number of bytes constituting the character
 41095 pointed to by *s*. Except that the shift state of *mbtowc()* is not affected, it shall be equivalent to:

41096 `mbtowc((wchar_t *)0, s, n);`

41097 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 41098 *mblen()*.

41099 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 41100 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 41101 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 41102 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 41103 null pointer shall cause this function to return a non-zero value if encodings have state
 41104 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 41105 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 41106 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 41107 unspecified.

41108 **RETURN VALUE**

41109 If *s* is a null pointer, *mblen()* shall return a non-zero or 0 value, if character encodings,
 41110 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen()* shall
 41111 either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 41112 character (if the next *n* or fewer bytes form a valid character), or return -1 (if they do not form a
 41113 valid character) and may set *errno* to indicate the error. In no case shall the value returned be
 41114 greater than *n* or the value of the {MB_CUR_MAX} macro.

41115 **ERRORS**

41116 The *mblen()* function may fail if:

41117 XSI [EILSEQ] Invalid character sequence is detected.

41118 **EXAMPLES**

41119 None.

41120 **APPLICATION USAGE**

41121 None.

41122 **RATIONALE**

41123 None.

41124 **FUTURE DIRECTIONS**

41125 None.

mblen()

41126

SEE ALSO

41127

mbtowc(), *mbstowcs()*, *wctomb()*, *wcstombs()*

41128

XBD <stdlib.h>

41129

CHANGE HISTORY

41130

First released in Issue 4. Aligned with the ISO C standard.



41131 **NAME**
 41132 `mbrlen` — get number of bytes in a character (restartable)

41133 **SYNOPSIS**
 41134 `#include <wchar.h>`
 41135 `size_t mbrlen(const char *restrict s, size_t n,`
 41136 `mbstate_t *restrict ps);`

41137 **DESCRIPTION**

41138 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41139 conflict between the requirements described here and the ISO C standard is unintentional. This
 41140 volume of POSIX.1-200x defers to the ISO C standard.

41141 If *s* is not a null pointer, `mbrlen()` shall determine the number of bytes constituting the character
 41142 pointed to by *s*. It shall be equivalent to:

41143 `mbstate_t internal;`
 41144 `mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);`

41145 If *ps* is a null pointer, the `mbrlen()` function shall use its own internal `mbstate_t` object, which is
 41146 initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object
 41147 pointed to by *ps* shall be used to completely describe the current conversion state of the
 41148 associated character sequence. The implementation shall behave as if no function defined in this
 41149 volume of POSIX.1-200x calls `mbrlen()`.

41150 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.

41151 **RETURN VALUE**

41152 The `mbrlen()` function shall return the first of the following that applies:

41153 0 If the next *n* or fewer bytes complete the character that corresponds to the null
 41154 wide character.

41155 *positive* If the next *n* or fewer bytes complete a valid character; the value returned shall
 41156 be the number of bytes that complete the character.

41157 `(size_t)-2` If the next *n* bytes contribute to an incomplete but potentially valid character,
 41158 and all *n* bytes have been processed. When *n* has at least the value of the
 41159 `{MB_CUR_MAX}` macro, this case can only occur if *s* points at a sequence of
 41160 redundant shift sequences (for implementations with state-dependent
 41161 encodings).

41162 `(size_t)-1` If an encoding error occurs, in which case the next *n* or fewer bytes do not
 41163 contribute to a complete and valid character. In this case, `[EILSEQ]` shall be
 41164 stored in `errno` and the conversion state is undefined.

41165 **ERRORS**

41166 The `mbrlen()` function may fail if:

41167 `[EINVAL]` *ps* points to an object that contains an invalid conversion state.

41168 `[EILSEQ]` Invalid character sequence is detected.

mbrlen()

41169	EXAMPLES
41170	None.
41171	APPLICATION USAGE
41172	None.
41173	RATIONALE
41174	None.
41175	FUTURE DIRECTIONS
41176	None.
41177	SEE ALSO
41178	<i>mbstowc(), mbrtowc()</i>
41179	XBD < wchar.h >
41180	CHANGE HISTORY
41181	First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
41182	(E).
41183	Issue 6
41184	The <i>mbrlen()</i> prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41185 **NAME**41186 `mbrtowc` — convert a character to a wide-character code (restartable)41187 **SYNOPSIS**41188 `#include <wchar.h>`41189 `size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,`
41190 `size_t n, mbstate_t *restrict ps);`41191 **DESCRIPTION**41192 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
41193 conflict between the requirements described here and the ISO C standard is unintentional. This
41194 volume of POSIX.1-200x defers to the ISO C standard.41195 If *s* is a null pointer, the `mbrtowc()` function shall be equivalent to the call:41196 `mbrtowc(NULL, "", 1, ps)`41197 In this case, the values of the arguments *pwc* and *n* are ignored.41198 If *s* is not a null pointer, the `mbrtowc()` function shall inspect at most *n* bytes beginning at the
41199 byte pointed to by *s* to determine the number of bytes needed to complete the next character
41200 (including any shift sequences). If the function determines that the next character is completed,
41201 it shall determine the value of the corresponding wide character and then, if *pwc* is not a null
41202 pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide
41203 character is the null wide character, the resulting state described shall be the initial conversion
41204 state.41205 If *ps* is a null pointer, the `mbrtowc()` function shall use its own internal `mbstate_t` object, which
41206 shall be initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t`
41207 object pointed to by *ps* shall be used to completely describe the current conversion state of the
41208 associated character sequence. The implementation shall behave as if no function defined in this
41209 volume of POSIX.1-200x calls `mbrtowc()`.41210 The behavior of this function is affected by the `LC_CTYPE` category of the current locale.41211 **RETURN VALUE**41212 The `mbrtowc()` function shall return the first of the following that applies:41213 **0** If the next *n* or fewer bytes complete the character that corresponds to the null
41214 wide character (which is the value stored).41215 between 1 and *n* inclusive41216 If the next *n* or fewer bytes complete a valid character (which is the value
41217 stored); the value returned shall be the number of bytes that complete the
41218 character.41219 **(size_t)−2** If the next *n* bytes contribute to an incomplete but potentially valid character,
41220 and all *n* bytes have been processed (no value is stored). When *n* has at least
41221 the value of the `{MB_CUR_MAX}` macro, this case can only occur if *s* points at
41222 a sequence of redundant shift sequences (for implementations with state-
41223 dependent encodings).41224 **(size_t)−1** If an encoding error occurs, in which case the next *n* or fewer bytes do not
41225 contribute to a complete and valid character (no value is stored). In this case,
41226 `[EILSEQ]` shall be stored in `errno` and the conversion state is undefined.

mbrtowc()

41227

ERRORS

41228

The *mbrtowc()* function may fail if:

41229

CX

[EINVAL] *ps* points to an object that contains an invalid conversion state.

41230

[EILSEQ]

Invalid character sequence is detected.

41231

EXAMPLES

41232

None.

41233

APPLICATION USAGE

41234

None.

41235

RATIONALE

41236

None.

41237

FUTURE DIRECTIONS

41238

None.

41239

SEE ALSO

41240

mbsinit(), *mbsrtowcs()*

41241

XBD <[wchar.h](#)>

41242

CHANGE HISTORY

41243

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

41244

41245

Issue 6

41246

The *mbrtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41247

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

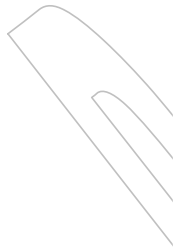
41248

- The **[EINVAL]** error condition is added.

41249

41250

ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.



41251 **NAME**

41252 mbsinit — determine conversion object status

41253 **SYNOPSIS**

41254 #include <wchar.h>

41255 int mbsinit(const mbstate_t *ps);

41256 **DESCRIPTION**

41257 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41258 conflict between the requirements described here and the ISO C standard is unintentional. This
 41259 volume of POSIX.1-200x defers to the ISO C standard.

41260 If *ps* is not a null pointer, the *mbsinit()* function shall determine whether the object pointed to by
 41261 *ps* describes an initial conversion state.

41262 **RETURN VALUE**

41263 The *mbsinit()* function shall return non-zero if *ps* is a null pointer, or if the pointed-to object
 41264 describes an initial conversion state; otherwise, it shall return zero.

41265 If an **mbstate_t** object is altered by any of the functions described as “restartable”, and is then
 41266 used with a different character sequence, or in the other conversion direction, or with a different
 41267 *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

41268 **ERRORS**

41269 No errors are defined.

41270 **EXAMPLES**

41271 None.

41272 **APPLICATION USAGE**

41273 The **mbstate_t** object is used to describe the current conversion state from a particular character
 41274 sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the
 41275 *LC_CTYPE* category of the current locale.

41276 The initial conversion state corresponds, for a conversion in either direction, to the beginning of
 41277 a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one
 41278 way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate
 41279 conversion involving any character sequence, in any *LC_CTYPE* category setting.

41280 **RATIONALE**

41281 None.

41282 **FUTURE DIRECTIONS**

41283 None.

41284 **SEE ALSO**41285 *mbrlen()*, *mbrtowc()*, *mbsrtowcs()*, *wcrtomb()*, *wcsrtombs()*

41286 XBD <wchar.h>

41287 **CHANGE HISTORY**

41288 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 41289 (E).

mbsnrtowcs()*System Interfaces*41290 **NAME**41291 `mbsnrtowcs` — convert a multi-byte string to a wide-character string41292 **SYNOPSIS**

```
41293 CX #include <wchar.h>  
41294 size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,  
41295 size_t nmc, size_t len, mbstate_t *restrict ps);
```

41296 **DESCRIPTION**41297 Refer to [mbsrtowcs\(\)](#).

41298 **NAME**41299 `mbsnrtowcs, mbsrtowcs` — convert a character string to a wide-character string (restartable)41300 **SYNOPSIS**41301 `#include <wchar.h>`

```
41302 CX size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,  

41303 size_t nmc, size_t len, mbstate_t *restrict ps);  

41304 size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,  

41305 size_t len, mbstate_t *restrict ps);
```

41306 **DESCRIPTION**

41307 CX For `mbsrtowcs()`: The functionality described on this reference page is aligned with the ISO C
41308 standard. Any conflict between the requirements described here and the ISO C standard is
41309 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

41310 The `mbsrtowcs()` function shall convert a sequence of characters, beginning in the conversion
41311 state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a
41312 sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters
41313 shall be stored into the array pointed to by `dst`. Conversion continues up to and including a
41314 terminating null character, which shall also be stored. Conversion shall stop early in either of the
41315 following cases:

- 41316 • A sequence of bytes is encountered that does not form a valid character.
- 41317 • `len` codes have been stored into the array pointed to by `dst` (and `dst` is not a null pointer).

41318 Each conversion shall take place as if by a call to the `mbrtowc()` function.

41319 If `dst` is not a null pointer, the pointer object pointed to by `src` shall be assigned either a null
41320 pointer (if conversion stopped due to reaching a terminating null character) or the address just
41321 past the last character converted (if any). If conversion stopped due to reaching a terminating
41322 null character, and if `dst` is not a null pointer, the resulting state described shall be the initial
41323 conversion state.

41324 If `ps` is a null pointer, the `mbsrtowcs()` function shall use its own internal `mbstate_t` object, which
41325 is initialized at program start-up to the initial conversion state. Otherwise, the `mbstate_t` object
41326 pointed to by `ps` shall be used to completely describe the current conversion state of the
41327 associated character sequence.

41328 CX The `mbsnrtowcs()` function shall be equivalent to the `mbsrtowcs()` function, except that the
41329 conversion of characters pointed to by `src` is limited to at most `nmc` bytes (the size of the input
41330 buffer).

41331 The behavior of these functions shall be affected by the `LC_CTYPE` category of the current locale.

41332 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
41333 these functions.

41334 **RETURN VALUE**

41335 If the input conversion encounters a sequence of bytes that do not form a valid character, an
41336 encoding error occurs. In this case, these functions shall store the value of the macro `[EILSEQ]` in
41337 `errno` and shall return `(size_t)-1`; the conversion state is undefined. Otherwise, these functions
41338 shall return the number of characters successfully converted, not including the terminating null
41339 (if any).

mbsrtowcs()41340 **ERRORS**

41341 These functions may fail if:

41342 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

41343 [EILSEQ] Invalid character sequence is detected.

41344 **EXAMPLES**

41345 None.

41346 **APPLICATION USAGE**

41347 None.

41348 **RATIONALE**

41349 None.

41350 **FUTURE DIRECTIONS**

41351 None.

41352 **SEE ALSO**41353 *iconv*, *mbrtowc()*, *mbsinit()*41354 XBD <[wchar.h](#)>41355 **CHANGE HISTORY**41356 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
41357 (E).41358 **Issue 6**41359 The *mbsrtowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41360 The [EINVAL] error condition is marked CX.

41361 **Issue 7**41362 The *mbsnrtowcs()* function is added from The Open Group Technical Standard, 2006, Extended
41363 API Set Part 1.

41364 **NAME**
 41365 `mbstowcs` — convert a character string to a wide-character string

41366 **SYNOPSIS**
 41367 `#include <stdlib.h>`
 41368 `size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,`
 41369 `size_t n);`

41370 DESCRIPTION

41371 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41372 conflict between the requirements described here and the ISO C standard is unintentional. This
 41373 volume of POSIX.1-200x defers to the ISO C standard.

41374 The `mbstowcs()` function shall convert a sequence of characters that begins in the initial shift
 41375 state from the array pointed to by `s` into a sequence of corresponding wide-character codes and
 41376 shall store not more than `n` wide-character codes into the array pointed to by `pwcs`. No
 41377 characters that follow a null byte (which is converted into a wide-character code with value 0)
 41378 shall be examined or converted. Each character shall be converted as if by a call to `mbtowlc()`,
 41379 except that the shift state of `mbtowlc()` is not affected.

41380 No more than `n` elements shall be modified in the array pointed to by `pwcs`. If copying takes
 41381 place between objects that overlap, the behavior is undefined.

41382 XSI The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.
 41383 If `pwcs` is a null pointer, `mbstowcs()` shall return the length required to convert the entire array
 41384 regardless of the value of `n`, but no values are stored.

41385 RETURN VALUE

41386 CX If an invalid character is encountered, `mbstowcs()` shall return `(size_t)-1` and may set `errno` to
 41387 indicate the error.

41388 XSI Otherwise, `mbstowcs()` shall return the number of the array elements modified (or required if
 41389 `pwcs` is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if
 41390 the value returned is `n`.

41391 ERRORS

41392 The `mbstowcs()` function may fail if:

41393 XSI `[EILSEQ]` Invalid byte sequence is detected.

41394 EXAMPLES

41395 None.

41396 APPLICATION USAGE

41397 None.

41398 RATIONALE

41399 None.

41400 FUTURE DIRECTIONS

41401 None.

41402 SEE ALSO

41403 [mblen\(\)](#), [mbtowlc\(\)](#), [wctomb\(\)](#), [wcstombs\(\)](#)

41404 XBD [<stdlib.h>](#)

41405

CHANGE HISTORY

41406

First released in Issue 4. Aligned with the ISO C standard.

41407

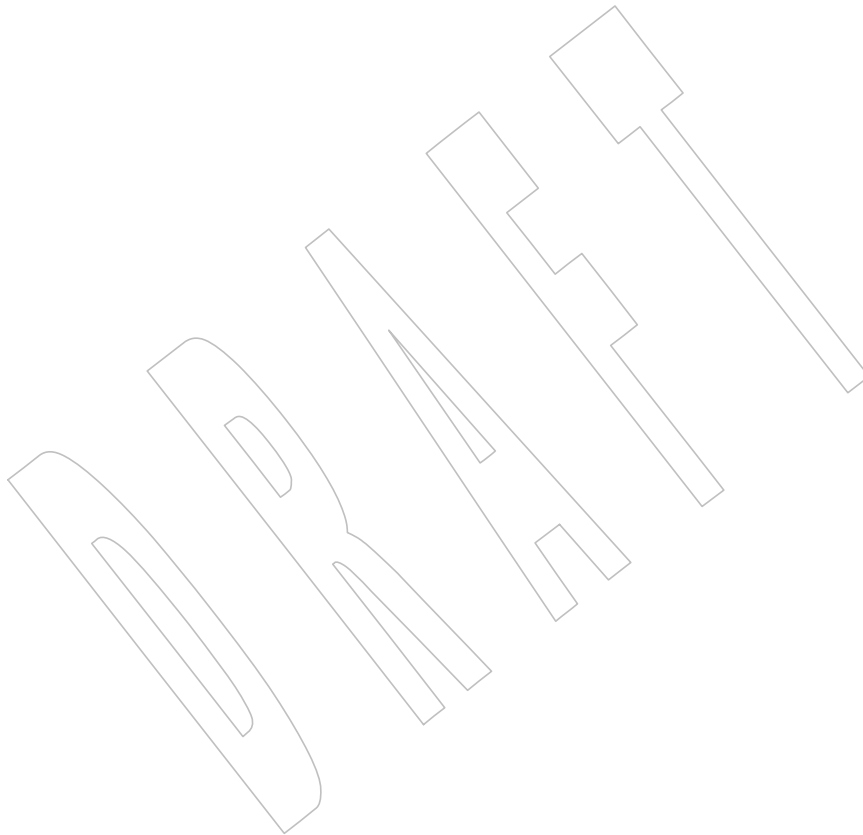
Issue 6

41408

The *mbstowcs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41409

Extensions beyond the ISO C standard are marked.



41410 **NAME**
 41411 `mbtowc` — convert a character to a wide-character code

41412 **SYNOPSIS**

41413 `#include <stdlib.h>`
 41414 `int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);`

41415 **DESCRIPTION**

41416 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41417 conflict between the requirements described here and the ISO C standard is unintentional. This
 41418 volume of POSIX.1-200x defers to the ISO C standard.

41419 If *s* is not a null pointer, *mbtowc()* shall determine the number of bytes that constitute the
 41420 character pointed to by *s*. It shall then determine the wide-character code for the value of type
 41421 `wchar_t` that corresponds to that character. (The value of the wide-character code corresponding
 41422 to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc()* shall store
 41423 the wide-character code in the object pointed to by *pwc*.

41424 The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 41425 state-dependent encoding, this function is placed into its initial state by a call for which its
 41426 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 41427 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 41428 null pointer shall cause this function to return a non-zero value if encodings have state
 41429 dependency, and 0 otherwise. If the implementation employs special bytes to change the shift
 41430 state, these bytes shall not produce separate wide-character codes, but shall be grouped with an
 41431 adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be
 41432 unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

41433 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 41434 *mbtowc()*.

41435 **RETURN VALUE**

41436 If *s* is a null pointer, *mbtowc()* shall return a non-zero or 0 value, if character encodings,
 41437 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc()*
 41438 shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the
 41439 CX converted character (if the next *n* or fewer bytes form a valid character), or return -1 and may
 41440 set *errno* to indicate the error (if they do not form a valid character).

41441 In no case shall the value returned be greater than *n* or the value of the `{MB_CUR_MAX}` macro.

41442 **ERRORS**

41443 The *mbtowc()* function may fail if:

41444 XSI **[EILSEQ]** Invalid character sequence is detected.

41445 **EXAMPLES**

41446 None.

41447 **APPLICATION USAGE**

41448 None.

41449 **RATIONALE**

41450 None.

mbtowc()

41451

FUTURE DIRECTIONS

41452

None.

41453

SEE ALSO

41454

mblen(), *mbstowcs()*, *wctomb()*, *wcstombs()*

41455

XBD <stdlib.h>

41456

CHANGE HISTORY

41457

First released in Issue 4. Aligned with the ISO C standard.

41458

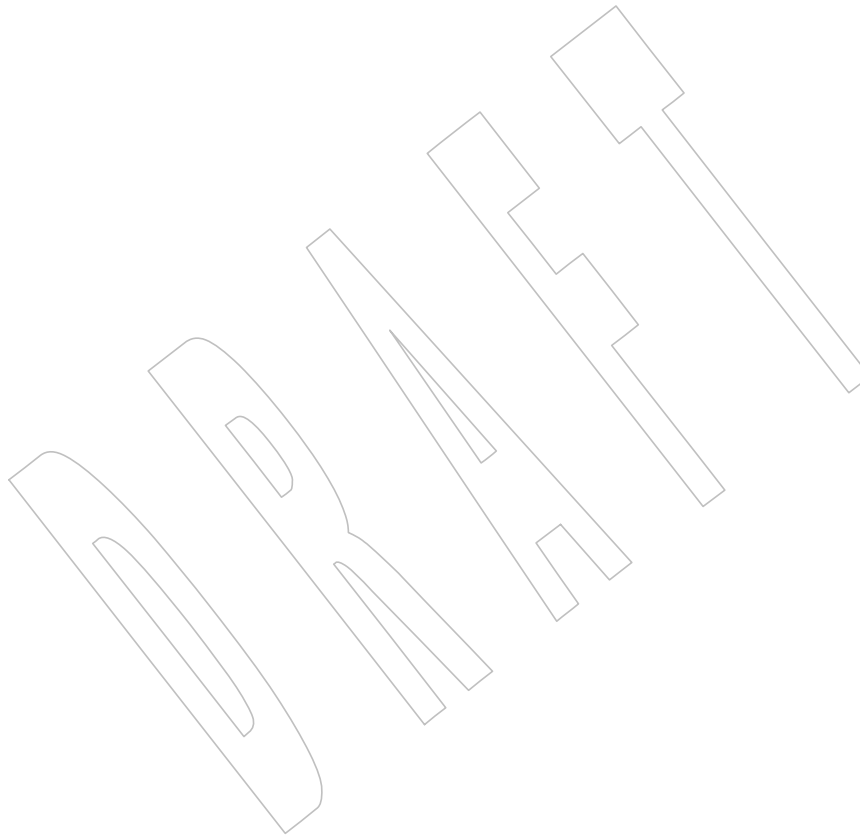
Issue 6

41459

The *mbtowc()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41460

Extensions beyond the ISO C standard are marked.



41461 **NAME**

41462 memccpy — copy bytes in memory

41463 **SYNOPSIS**

```
41464 XSI #include <string.h>
41465 void *memccpy(void *restrict s1, const void *restrict s2,
41466 int c, size_t n);
```

41467 **DESCRIPTION**

41468 The *memccpy()* function shall copy bytes from memory area *s2* into *s1*, stopping after the first
 41469 occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied,
 41470 whichever comes first. If copying takes place between objects that overlap, the behavior is
 41471 undefined.

41472 **RETURN VALUE**

41473 The *memccpy()* function shall return a pointer to the byte after the copy of *c* in *s1*, or a null
 41474 pointer if *c* was not found in the first *n* bytes of *s2*.

41475 **ERRORS**

41476 No errors are defined.

41477 **EXAMPLES**

41478 None.

41479 **APPLICATION USAGE**

41480 The *memccpy()* function does not check for the overflow of the receiving memory area.

41481 **RATIONALE**

41482 None.

41483 **FUTURE DIRECTIONS**

41484 None.

41485 **SEE ALSO**

41486 XBD [<string.h>](#)

41487 **CHANGE HISTORY**

41488 First released in Issue 1. Derived from Issue 1 of the SVID.

41489 **Issue 6**

41490 The **restrict** keyword is added to the *memccpy()* prototype for alignment with the
 41491 ISO/IEC 9899:1999 standard.

41492 **NAME**

41493 memchr — find byte in memory

41494 **SYNOPSIS**

41495 #include <string.h>

41496 void *memchr(const void *s, int c, size_t n);

41497 **DESCRIPTION**

41498 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41499 conflict between the requirements described here and the ISO C standard is unintentional. This
 41500 volume of POSIX.1-200x defers to the ISO C standard.

41501 The *memchr()* function shall locate the first occurrence of *c* (converted to an **unsigned char**) in
 41502 the initial *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s*.

41503 **RETURN VALUE**

41504 The *memchr()* function shall return a pointer to the located byte, or a null pointer if the byte does
 41505 not occur in the object.

41506 **ERRORS**

41507 No errors are defined.

41508 **EXAMPLES**

41509 None.

41510 **APPLICATION USAGE**

41511 None.

41512 **RATIONALE**

41513 None.

41514 **FUTURE DIRECTIONS**

41515 None.

41516 **SEE ALSO**

41517 XBD <string.h>

41518 **CHANGE HISTORY**

41519 First released in Issue 1. Derived from Issue 1 of the SVID.

41520 **NAME**

41521 memcmp — compare bytes in memory

41522 **SYNOPSIS**

41523 #include <string.h>

41524 int memcmp(const void *s1, const void *s2, size_t n);

41525 **DESCRIPTION**

41526 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41527 conflict between the requirements described here and the ISO C standard is unintentional. This
 41528 volume of POSIX.1-200x defers to the ISO C standard.

41529 The *memcmp()* function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the
 41530 object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

41531 The sign of a non-zero return value shall be determined by the sign of the difference between the
 41532 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects
 41533 being compared.

41534 **RETURN VALUE**

41535 The *memcmp()* function shall return an integer greater than, equal to, or less than 0, if the object
 41536 pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

41537 **ERRORS**

41538 No errors are defined.

41539 **EXAMPLES**

41540 None.

41541 **APPLICATION USAGE**

41542 None.

41543 **RATIONALE**

41544 None.

41545 **FUTURE DIRECTIONS**

41546 None.

41547 **SEE ALSO**

41548 XBD <string.h>

41549 **CHANGE HISTORY**

41550 First released in Issue 1. Derived from Issue 1 of the SVID.

41551 **NAME**41552 `memcpy` — copy bytes in memory41553 **SYNOPSIS**41554 `#include <string.h>`41555 `void *memcpy(void *restrict s1, const void *restrict s2, size_t n);`41556 **DESCRIPTION**

41557 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41558 conflict between the requirements described here and the ISO C standard is unintentional. This
 41559 volume of POSIX.1-200x defers to the ISO C standard.

41560 The `memcpy()` function shall copy *n* bytes from the object pointed to by *s2* into the object pointed
 41561 to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

41562 **RETURN VALUE**41563 The `memcpy()` function shall return *s1*; no return value is reserved to indicate an error.41564 **ERRORS**

41565 No errors are defined.

41566 **EXAMPLES**

41567 None.

41568 **APPLICATION USAGE**41569 The `memcpy()` function does not check for the overflow of the receiving memory area.41570 **RATIONALE**

41571 None.

41572 **FUTURE DIRECTIONS**

41573 None.

41574 **SEE ALSO**41575 XBD `<string.h>`41576 **CHANGE HISTORY**

41577 First released in Issue 1. Derived from Issue 1 of the SVID.

41578 **Issue 6**41579 The `memcpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

41580 **NAME**

41581 memmove — copy bytes in memory with overlapping areas

41582 **SYNOPSIS**

41583 #include <string.h>

41584 void *memmove(void *s1, const void *s2, size_t n);

41585 **DESCRIPTION**

41586 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41587 conflict between the requirements described here and the ISO C standard is unintentional. This
 41588 volume of POSIX.1-200x defers to the ISO C standard.

41589 The *memmove()* function shall copy *n* bytes from the object pointed to by *s2* into the object
 41590 pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first
 41591 copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and
 41592 *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

41593 **RETURN VALUE**41594 The *memmove()* function shall return *s1*; no return value is reserved to indicate an error.41595 **ERRORS**

41596 No errors are defined.

41597 **EXAMPLES**

41598 None.

41599 **APPLICATION USAGE**

41600 None.

41601 **RATIONALE**

41602 None.

41603 **FUTURE DIRECTIONS**

41604 None.

41605 **SEE ALSO**

41606 XBD <string.h>

41607 **CHANGE HISTORY**

41608 First released in Issue 4. Derived from the ANSI C standard.

41609 **NAME**41610 `memset` — set bytes in memory41611 **SYNOPSIS**41612 `#include <string.h>`41613 `void *memset(void *s, int c, size_t n);`41614 **DESCRIPTION**

41615 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 41616 conflict between the requirements described here and the ISO C standard is unintentional. This
 41617 volume of POSIX.1-200x defers to the ISO C standard.

41618 The `memset()` function shall copy `c` (converted to an **unsigned char**) into each of the first `n` bytes
 41619 of the object pointed to by `s`.

41620 **RETURN VALUE**41621 The `memset()` function shall return `s`; no return value is reserved to indicate an error.41622 **ERRORS**

41623 No errors are defined.

41624 **EXAMPLES**

41625 None.

41626 **APPLICATION USAGE**

41627 None.

41628 **RATIONALE**

41629 None.

41630 **FUTURE DIRECTIONS**

41631 None.

41632 **SEE ALSO**41633 XBD [<string.h>](#)41634 **CHANGE HISTORY**

41635 First released in Issue 1. Derived from Issue 1 of the SVID.

41636 **NAME**

41637 mkdir, mkdirat — make a directory relative to directory file descriptor

41638 **SYNOPSIS**

41639 #include <sys/stat.h>

41640 int mkdir(const char *path, mode_t mode);

41641 int mkdirat(int fd, const char *path, mode_t mode);

41642 **DESCRIPTION**

41643 The *mkdir()* function shall create a new directory with name *path*. The file permission bits of the
 41644 new directory shall be initialized from *mode*. These file permission bits of the *mode* argument
 41645 shall be modified by the process' file creation mask.

41646 When bits in *mode* other than the file permission bits are set, the meaning of these additional bits
 41647 is implementation-defined.

41648 The directory's user ID shall be set to the process' effective user ID. The directory's group ID
 41649 shall be set to the group ID of the parent directory or to the effective group ID of the process.
 41650 Implementations shall provide a way to initialize the directory's group ID to the group ID of the
 41651 parent directory. Implementations may, but need not, provide an implementation-defined way
 41652 to initialize the directory's group ID to the effective group ID of the calling process.

41653 The newly created directory shall be an empty directory.

41654 If *path* names a symbolic link, *mkdir()* shall fail and set *errno* to [EEXIST].

41655 Upon successful completion, *mkdir()* shall mark for update the last data access, last data
 41656 modification, and last file status change timestamps of the directory. Also, the last data
 41657 modification and last file status change timestamps of the directory that contains the new entry
 41658 shall be marked for update.

41659 The *mkdirat()* function shall be equivalent to the *mkdir()* function except in the case where *path*
 41660 specifies a relative path. In this case the newly created directory is created relative to the
 41661 directory associated with the file descriptor *fd* instead of the current working directory. It is
 41662 unspecified whether directory searches are permitted based on whether the file was opened
 41663 with search permission or on the current permissions of the directory underlying the file
 41664 descriptor.

41665 If *mkdirat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
 41666 directory is used and the behavior shall be identical to a call to *mkdir()*.

41667 **RETURN VALUE**

41668 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 41669 return -1 and set *errno* to indicate the error. If -1 is returned, no directory shall be created.

41670 **ERRORS**

41671 These functions shall fail if:

41672 [EACCES] Search permission is denied on a component of the path prefix, or write
 41673 permission is denied on the parent directory of the directory to be created.

41674 [EEXIST] The named file exists.

41675 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 41676 argument.

41677 [EMLINK] The link count of the parent directory would exceed {LINK_MAX}.

- 41678 [ENAMETOOLONG]
 41679 The length of the *path* argument exceeds {PATH_MAX} or a pathname
 41680 component is longer than {NAME_MAX}.
- 41681 [ENOENT] A component of the path prefix specified by *path* does not name an existing
 41682 directory or *path* is an empty string.
- 41683 [ENOSPC] The file system does not contain enough space to hold the contents of the new
 41684 directory or to extend the parent directory of the new directory.
- 41685 [ENOTDIR] A component of the path prefix is not a directory.
- 41686 [EROFS] The parent directory resides on a read-only file system.
- 41687 In addition, the *mkdirat*() function shall fail if:
- 41688 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 41689 neither AT_FDCWD nor a valid file descriptor open for reading.

41690 These functions may fail if:

- 41691 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 41692 resolution of the *path* argument.
- 41693 [ENAMETOOLONG]
 41694 As a result of encountering a symbolic link in resolution of the *path* argument,
 41695 the length of the substituted pathname string exceeded {PATH_MAX}.
- 41696 The *mkdirat*() function may fail if:
- 41697 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
 41698 file descriptor associated with a directory.

41699 EXAMPLES

41700 Creating a Directory

41701 The following example shows how to create a directory named **/home/cnd/mod1**, with
 41702 read/write/search permissions for owner and group, and with read/search permissions for
 41703 others.

```
41704 #include <sys/types.h>
41705 #include <sys/stat.h>
41706
41707 int status;
41708 ...
41709 status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

41709 APPLICATION USAGE

41710 None.

41711 RATIONALE

41712 The *mkdir*() function originated in 4.2 BSD and was added to System V in Release 3.0.

41713 4.3 BSD detects [ENAMETOOLONG].

41714 The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the
 41715 group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2
 41716 required that implementations provide a way to have the group ID be set to the group ID of the
 41717 containing directory, but did not prohibit implementations also supporting a way to set the
 41718 group ID to the effective group ID of the creating process. Conforming applications should not
 41719 assume which group ID will be used. If it matters, an application can use *chown*() to set the
 41720 group ID after the directory is created, or determine under what conditions the implementation
 41721 will set the desired group ID.

41722 The purpose of the *mkdirat()* function is to create a directory in directories other than the current
 41723 working directory without exposure to race conditions. Any part of the path of a file could be
 41724 changed in parallel to the call to *mkdir()*, resulting in unspecified behavior. By opening a file
 41725 descriptor for the target directory and using the *mkdirat()* function it can be guaranteed that the
 41726 newly created directory is located relative to the desired directory.

FUTURE DIRECTIONS

41727 None.
 41728

SEE ALSO

41729 *chmod*, *mkdtemp()*, *mknod()*, *umask*

41731 XBD [<sys/stat.h>](#), [<sys/types.h>](#) |

CHANGE HISTORY

41732 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.
 41733

Issue 6

41734 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.
 41735

41736 The following new requirements on POSIX implementations derive from alignment with the
 41737 Single UNIX Specification:

- 41738 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 41739 required for conforming implementations of previous POSIX specifications, it was not
 41740 required for UNIX applications.
- 41741 • The [ELOOP] mandatory error condition is added.
- 41742 • A second [ENAMETOOLONG] is added as an optional error condition.

41743 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41744 • The [ELOOP] optional error condition is added.

Issue 7

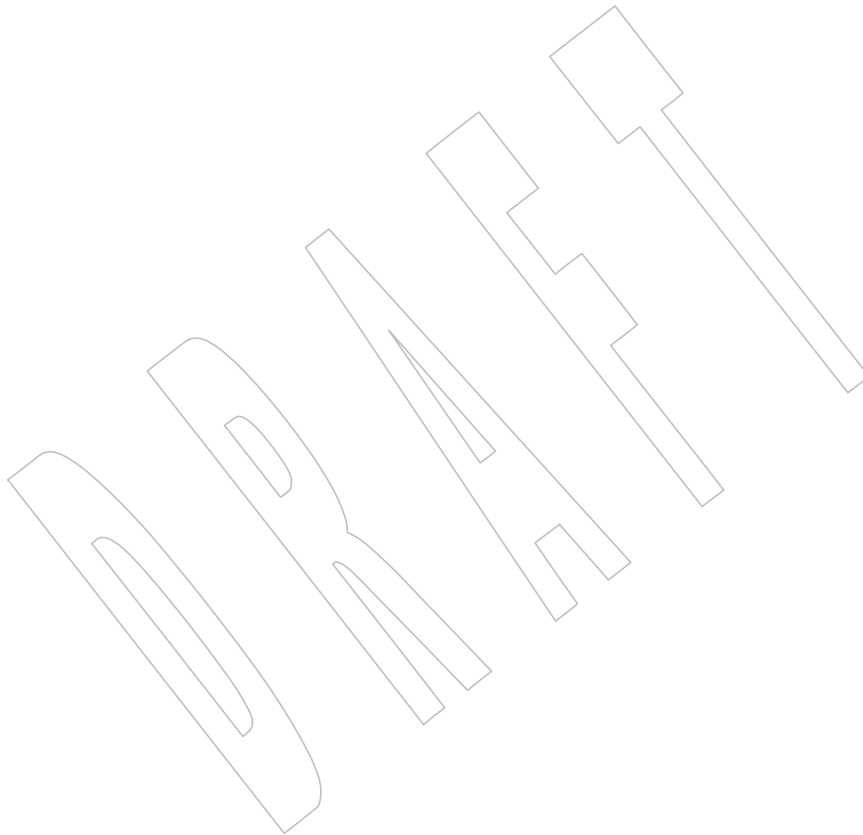
41745 The *mkdirat()* function is added from The Open Group Technical Standard, 2006, Extended API
 41746 Set Part 2.

41747 Changes are made related to support for finegrained timestamps.
 41748 +

41749 **NAME**
41750 mkdirat — make a directory relative to directory file descriptor

41751 **SYNOPSIS**
41752 #include <sys/stat.h>
41753 int mkdirat(int *fd*, const char **path*, mode_t *mode*);

41754 **DESCRIPTION**
41755 Refer to *mkdir*.



41756 **NAME**
 41757 mkdtemp, mkstemp — create a unique directory or file

41758 **SYNOPSIS**

```
41759 CX #include <stdlib.h>
41760 char *mkdtemp(char *template);
41761 int mkstemp(char *template);
```

41762 **DESCRIPTION**

41763 The *mkdtemp()* function uses the contents of *template* to construct a unique directory name. The
 41764 string provided in *template* shall be a filename ending with six trailing 'X's. The *mkdtemp()*
 41765 function shall replace each 'X' with a character from the portable filename character set. The
 41766 characters are chosen such that the resulting name does not duplicate the name of an existing file
 41767 at the time of a call to *mkdtemp()*. The unique directory name is used to attempt to create the
 41768 directory using mode 0700 as modified by the file creation mask.

41769 The *mkstemp()* function shall replace the contents of the string pointed to by *template* by a unique
 41770 filename, and return a file descriptor for the file open for reading and writing. The *mkstemp()*
 41771 function shall create the file, and obtain a file descriptor for it, as if by a call to:

```
41772 open(filename, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)
```

41773 The function thus prevents any possible race condition between testing whether the file exists
 41774 and opening it for use. The string in *template* should look like a filename with six trailing 'X's;
 41775 *mkstemp()* replaces each 'X' with a character from the portable filename character set. The
 41776 characters are chosen such that the resulting name does not duplicate the name of an existing file
 41777 at the time of a call to *mkstemp()*.

41778 **RETURN VALUE**

41779 Upon successful completion, the *mkdtemp()* function shall return a pointer to the string
 41780 containing the directory name if it was created. Otherwise, it shall return a null pointer and shall
 41781 set *errno* to indicate the error.

41782 Upon successful completion, the *mkstemp()* function shall return an open file descriptor. |
 41783 Otherwise, it shall return -1 if no suitable file could be created.

41784 **ERRORS**

41785 The *mkdtemp()* function shall fail if: |

41786 [EACCES] Search permission is denied on a component of the path prefix, or write
 41787 permission is denied on the parent directory of the directory to be created.

41788 [EINVAL] The string pointed to by *template* does not end in "XXXXXX".

41789 [ELOOP] A loop exists in symbolic links encountered during resolution of the path of
 41790 the directory to be created.

41791 [EMLINK] The link count of the parent directory would exceed {LINK_MAX}.

41792 [ENAMETOOLONG]

41793 The length of the *template* argument exceeds {PATH_MAX} or a pathname
 41794 component is longer than {NAME_MAX}.

41795 [ENOENT] A component of the path prefix specified by the *template* argument does not
 41796 name an existing directory or path is an empty string.

- 41797 [ENOSPC] The file system does not contain enough space to hold the contents of the new
41798 directory or to extend the parent directory of the new directory.
- 41799 [ENOTDIR] A component of the path prefix is not a directory.
- 41800 [EROFS] The parent directory resides on a read-only file system.
- 41801 The *mkdtemp()* function may fail if:
- 41802 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
41803 resolution of the path of the directory to be created.
- 41804 [ENAMETOOLONG]
41805 As a result of encountering a symbolic link in resolution of the path of the
41806 directory to be created, the length of the substituted pathname string exceeded
41807 {PATH_MAX}.
- 41808 The error conditions for the *mkstemp()* function are defined in *open()*. +

EXAMPLES**Generating a Filename**

The following example creates a file with a 10-character name beginning with the characters "file" and opens the file for reading and writing. The value returned as the value of *fd* is a file descriptor that identifies the file.

```
41814 #include <stdlib.h>
41815 ...
41816 char template[] = "/tmp/fileXXXXXX";
41817 int fd;
41818
41819 fd = mkstemp(template);
```

APPLICATION USAGE

It is possible to run out of letters.

The *mkdtemp()* and *mkstemp()* functions need not check to determine whether the filename part of *template* exceeds the maximum allowable filename length.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid(), *mkdir*, *open()*, *tmpfile()*, *tmpnam()*

XBD <stdlib.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 7

The *mkstemp()* function is moved from the XSI option to the Base.

The *mkdtemp()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

SD5-XSH-ERN-168 is applied, clarifying file permissions upon creation.

41839 **NAME**

41840 mkfifo, mkfifoat — make a FIFO special file relative to directory file descriptor

41841 **SYNOPSIS**

41842 #include <sys/stat.h>

41843 int mkfifo(const char *path, mode_t mode);

41844 int mkfifoat(int fd, const char *path, mode_t mode);

41845 **DESCRIPTION**41846 The *mkfifo()* function shall create a new FIFO special file named by the pathname pointed to by
41847 *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission
41848 bits of the *mode* argument shall be modified by the process' file creation mask.41849 When bits in *mode* other than the file permission bits are set, the effect is implementation-
41850 defined.41851 If *path* names a symbolic link, *mkfifo()* shall fail and set *errno* to [EEXIST].41852 The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set
41853 to the group ID of the parent directory or to the effective group ID of the process.
41854 Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the
41855 parent directory. Implementations may, but need not, provide an implementation-defined way
41856 to initialize the FIFO's group ID to the effective group ID of the calling process.41857 Upon successful completion, *mkfifo()* shall mark for update the last data access, last data |
41858 modification, and last file status change timestamps of the file. Also, the last data modification |
41859 and last file status change timestamps of the directory that contains the new entry shall be |
41860 marked for update.41861 The *mkfifoat()* function shall be equivalent to the *mkfifo()* function except in the case where *path*
41862 specifies a relative path. In this case the newly created FIFO is created relative to the directory
41863 associated with the file descriptor *fd* instead of the current working directory. It is unspecified
41864 whether directory searches are permitted based on whether the file was opened with search
41865 permission or on the current permissions of the directory underlying the file descriptor.41866 If *mkfifoat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
41867 directory is used and the behavior shall be identical to a call to *mkfifo()*.41868 **RETURN VALUE**41869 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
41870 return -1 and set *errno* to indicate the error. If -1 is returned, no FIFO shall be created.41871 **ERRORS**

41872 These functions shall fail if:

41873 [EACCES] A component of the path prefix denies search permission, or write permission
41874 is denied on the parent directory of the FIFO to be created.

41875 [EEXIST] The named file already exists.

41876 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
41877 argument.

41878 [ENAMETOOLONG]

41879 The length of the *path* argument exceeds {PATH_MAX} or a pathname
41880 component is longer than {NAME_MAX}.

- 41881 [ENOENT] A component of the path prefix specified by *path* does not name an existing
41882 directory or *path* is an empty string.
- 41883 [ENOSPC] The directory that would contain the new file cannot be extended or the file
41884 system is out of file-allocation resources.
- 41885 [ENOTDIR] A component of the path prefix is not a directory.
- 41886 [EROFS] The named file resides on a read-only file system.
- 41887 The *mkfifoat()* function shall fail if:
- 41888 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
41889 neither AT_FDCWD nor a valid file descriptor open for reading.
- 41890 These functions may fail if:
- 41891 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
41892 resolution of the *path* argument.
- 41893 [ENAMETOOLONG]
41894 As a result of encountering a symbolic link in resolution of the *path* argument,
41895 the length of the substituted pathname string exceeded {PATH_MAX}.
- 41896 The *mkfifoat()* function may fail if:
- 41897 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
41898 file descriptor associated with a directory.

EXAMPLES**Creating a FIFO File**

The following example shows how to create a FIFO file named `/home/cnd/mod_done`, with read/write permissions for owner, and with read permissions for group and others.

```
41903 #include <sys/types.h>
41904 #include <sys/stat.h>
41905 int status;
41906 ...
41907 status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
41908               S_IRGRP | S_IROTH);
```

APPLICATION USAGE

None.

RATIONALE

The syntax of this function is intended to maintain compatibility with historical implementations of *mknod()*. The latter function was included in the 1984 /usr/group standard but only for use in creating FIFO special files. The *mknod()* function was originally excluded from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir()* and *mkfifo()*. The *mknod()* function is now included for alignment with the Single UNIX Specification.

The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the FIFO is created, or determine under what conditions the implementation will set the desired group ID.

41926 The purpose of the *mkfifoat()* function is to create a FIFO special file in directories other than the
 41927 current working directory without exposure to race conditions. Any part of the path of a file
 41928 could be changed in parallel to a call to *mkfifo()*, resulting in unspecified behavior. By opening a
 41929 file descriptor for the target directory and using the *mkfifoat()* function it can be guaranteed that
 41930 the newly created FIFO is located relative to the desired directory.

41931 FUTURE DIRECTIONS

41932 None.

41933 SEE ALSO

41934 *chmod*, *mknod()*, *umask*

41935 XBD [<sys/stat.h>](#), [<sys/types.h>](#) |

41936 CHANGE HISTORY

41937 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

41938 Issue 6

41939 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

41940 The following new requirements on POSIX implementations derive from alignment with the
 41941 Single UNIX Specification:

- 41942 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
 41943 required for conforming implementations of previous POSIX specifications, it was not
 41944 required for UNIX applications.
- 41945 • The [ELOOP] mandatory error condition is added.
- 41946 • A second [ENAMETOOLONG] is added as an optional error condition.

41947 The following changes were made to align with the IEEE P1003.1a draft standard:

- 41948 • The [ELOOP] optional error condition is added.

41949 Issue 7

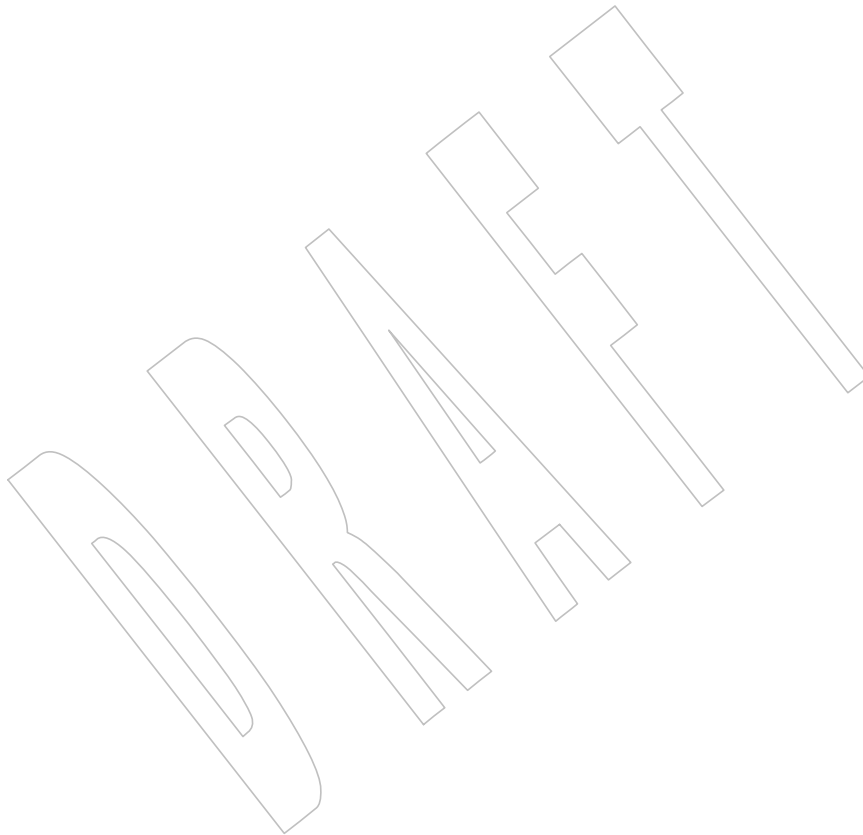
41950 The *mkfifoat()* function is added from The Open Group Technical Standard, 2006, Extended API
 41951 Set Part 2.

41952 Changes are made related to support for finegrained timestamps. +

41953 **NAME**
41954 `mkfifoat` — make a FIFO special file relative to directory file descriptor

41955 **SYNOPSIS**
41956 `#include <sys/stat.h>`
41957 `int mkfifoat(int fd, const char *path, mode_t mode);`

41958 **DESCRIPTION**
41959 Refer to *mkfifo*.



41960 **NAME**

41961 mknod, mknodat — make directory, special file, or regular file

41962 **SYNOPSIS**

```
41963 XSI #include <sys/stat.h>
41964 int mknod(const char *path, mode_t mode, dev_t dev);
41965 int mknodat(int fd, const char *path, mode_t mode, dev_t dev);
```

41966 **DESCRIPTION**41967 The *mknod()* function shall create a new file named by the pathname to which the argument *path*
41968 points.41969 The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the
41970 following symbolic constants:

Name	Description
S_IFIFO	FIFO-special
S_IFCHR	Character-special (non-portable)
S_IFDIR	Directory (non-portable)
S_IFBLK	Block-special (non-portable)
S_IFREG	Regular (non-portable)

41977 The only portable use of *mknod()* is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is
41978 not 0, the behavior of *mknod()* is unspecified.41979 The permissions for the new file are OR'ed into the *mode* argument, and may be selected from
41980 any combination of the following symbolic constants:

Name	Description
S_ISUID	Set user ID on execution.
S_ISGID	Set group ID on execution.
S_IRWXU	Read, write, or execute (search) by owner.
S_IRUSR	Read by owner.
S_IWUSR	Write by owner.
S_IXUSR	Execute (search) by owner.
S_IRWXG	Read, write, or execute (search) by group.
S_IRGRP	Read by group.
S_IWGRP	Write by group.
S_IXGRP	Execute (search) by group.
S_IRWXO	Read, write, or execute (search) by others.
S_IROTH	Read by others.
S_IWOTH	Write by others.
S_IXOTH	Execute (search) by others.
S_ISVTX	On directories, restricted deletion flag.

41997 The user ID of the file shall be initialized to the effective user ID of the process. The group ID of
41998 the file shall be initialized to either the effective group ID of the process or the group ID of the
41999 parent directory. Implementations shall provide a way to initialize the file's group ID to the
42000 group ID of the parent directory. Implementations may, but need not, provide an
42001 implementation-defined way to initialize the file's group ID to the effective group ID of the
42002 calling process. The owner, group, and other permission bits of *mode* shall be modified by the file
42003 mode creation mask of the process. The *mknod()* function shall clear each bit whose
42004 corresponding bit in the file mode creation mask of the process is set.

42005 If *path* names a symbolic link, *mknod()* shall fail and set *errno* to [EEXIST].

42006 Upon successful completion, *mknod()* shall mark for update the last data access, last data
42007 modification, and last file status change timestamps of the file. Also, the last data modification
42008 and last file status change timestamps of the directory that contains the new entry shall be
42009 marked for update.

42010 Only a process with appropriate privileges may invoke *mknod()* for file types other than FIFO-
42011 special.

42012 The *mknodat()* function shall be equivalent to the *mknod()* function except in the case where *path*
42013 specifies a relative path. In this case the newly created directory, special file, or regular file is
42014 located relative to the directory associated with the file descriptor *fd* instead of the current
42015 working directory. It is unspecified whether directory searches are permitted based on whether
42016 the file was opened with search permission or on the current permissions of the directory
42017 underlying the file descriptor.

42018 If *mknodat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
42019 directory is used and the behavior shall be identical to a call to *mknod()*.

RETURN VALUE

42020 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
42021 return -1 and set *errno* to indicate the error. If -1 is returned, the new file shall not be created.
42022

ERRORS

42023 These functions shall fail if:

42024 [EACCES] A component of the path prefix denies search permission, or write permission
42025 is denied on the parent directory.
42026

42027 [EEXIST] The named file exists.

42028 [EINVAL] An invalid argument exists.

42029 [EIO] An I/O error occurred while accessing the file system.

42030 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
42031 argument.

42032 [ENAMETOOLONG]

42033 The length of a pathname exceeds {PATH_MAX} or a pathname component is
42034 longer than {NAME_MAX}.

42035 [ENOENT] A component of the path prefix specified by *path* does not name an existing
42036 directory or *path* is an empty string.

42037 [ENOSPC] The directory that would contain the new file cannot be extended or the file
42038 system is out of file allocation resources.

42039 [ENOTDIR] A component of the path prefix is not a directory.

42040 [EPERM] The invoking process does not have appropriate privileges and the file type is
42041 not FIFO-special.

42042 [EROFS] The directory in which the file is to be created is located on a read-only file
42043 system.

42044 The *mknodat()* function shall fail if:

42045 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
42046 neither AT_FDCWD nor a valid file descriptor open for reading.

42047 These functions may fail if:

- 42048 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
42049 resolution of the *path* argument.
- 42050 [ENAMETOOLONG]
42051 Pathname resolution of a symbolic link produced an intermediate result
42052 whose length exceeds {PATH_MAX}.
- 42053 The *mknodat()* function may fail if:
- 42054 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
42055 file descriptor associated with a directory.

EXAMPLES**Creating a FIFO Special File**

The following example shows how to create a FIFO special file named */home/cnd/mod_done*, with read/write permissions for owner, and with read permissions for group and others.

```
42060 #include <sys/types.h>
42061 #include <sys/stat.h>
42062 dev_t dev;
42063 int status;
42064 ...
42065 status = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
42066 S_IRUSR | S_IRGRP | S_IROTH, dev);
```

APPLICATION USAGE

The *mkfifo()* function is preferred over this function for making FIFO special files.

RATIONALE

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown()* to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mknodat()* function is to create directories, special files, or regular files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mknod()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mknodat()* function it can be guaranteed that the newly created directory, special file, or regular file is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *creat()*, *exec*, *fstatat()*, *mkdir*, *mkfifo*, *open()*, *umask*

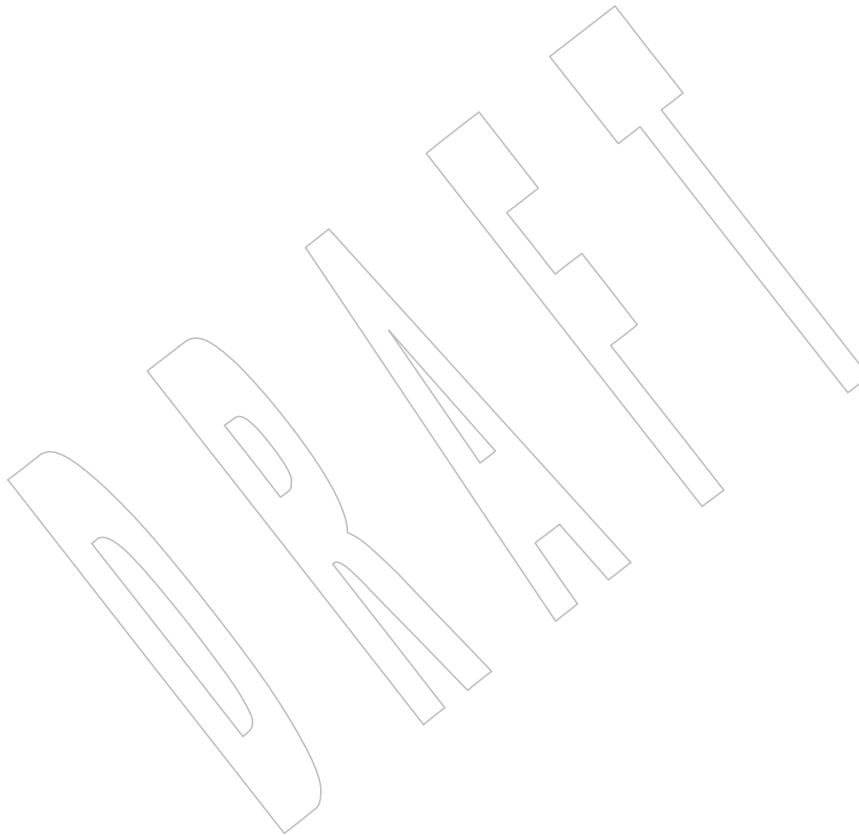
XBD [<sys/stat.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

mknod()

42091	Issue 5	
42092		Moved from X/OPEN UNIX extension to BASE.
42093	Issue 6	
42094		The normative text is updated to avoid use of the term “must” for application requirements.
42095		The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.
42096		
42097	Issue 7	
42098		The <i>mknodat()</i> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
42099		
42100		Changes are made related to support for finegrained timestamps. +



42101 **NAME**
42102 mknodat — make directory, special file, or regular file

42103 **SYNOPSIS**

42104 XSI `#include <sys/stat.h>`
42105 `int mknodat(int fd, const char *path, mode_t mode, dev_t dev);`

42106 **DESCRIPTION**

42107 Refer to *mknod()*.

mkstemp()

42108 **NAME**
42109 mkstemp — create a unique directory

SYNOPSIS

42111 CX `#include <stdlib.h>`
42112 `int mkstemp(char *template);`

DESCRIPTION

42113 Refer to *mkdtemp()*.
42114

42115 **NAME**

42116 mktime — convert broken-down time into time since the Epoch

42117 **SYNOPSIS**

42118 #include <time.h>

42119 time_t mktime(struct tm *timeptr);

42120 **DESCRIPTION**

42121 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 42122 conflict between the requirements described here and the ISO C standard is unintentional. This
 42123 volume of POSIX.1-200x defers to the ISO C standard.

42124 The *mktime()* function shall convert the broken-down time, expressed as local time, in the
 42125 structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that
 42126 of the values returned by *time()*. The original values of the *tm_wday* and *tm_yday* components of
 42127 the structure are ignored, and the original values of the other components are not restricted to
 42128 the ranges described in <time.h>.

42129 CX A positive or 0 value for *tm_isdst* shall cause *mktime()* to presume initially that Daylight Savings
 42130 Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall
 42131 cause *mktime()* to attempt to determine whether Daylight Savings Time is in effect for the
 42132 specified time.

42133 Local timezone information shall be set as though *mktime()* called *tzset()*.

42134 The relationship between the **tm** structure (defined in the <time.h> header) and the time in
 42135 seconds since the Epoch is that the result shall be as specified in the expression given in the
 42136 definition of seconds since the Epoch (see XBD Section 4.15, on page 100) corrected for timezone
 42137 and any seasonal time adjustments, where the names in the structure and in the expression
 42138 correspond.

42139 Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure
 42140 shall be set appropriately, and the other components are set to represent the specified time since
 42141 the Epoch, but with their values forced to the ranges indicated in the <time.h> entry; the final
 42142 value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

42143 **RETURN VALUE**

42144 The *mktime()* function shall return the specified time since the Epoch encoded as a value of type
 42145 **time_t**. If the time since the Epoch cannot be represented, the function shall return the value
 42146 CX **(time_t)-1** and may set *errno* to indicate the error.

42147 **ERRORS**42148 The *mktime()* function may fail if:

42149 CX [EOVERFLOW] The result cannot be represented.

42150 **EXAMPLES**

42151 What day of the week is July 4, 2001?

42152 #include <stdio.h>

42153 #include <time.h>

42154 struct tm time_str;

42155 char daybuf[20];

42156 int main(void)

42157 {

42158 time_str.tm_year = 2001 - 1900;

```

42159         time_str.tm_mon = 7 - 1;
42160         time_str.tm_mday = 4;
42161         time_str.tm_hour = 0;
42162         time_str.tm_min = 0;
42163         time_str.tm_sec = 1;
42164         time_str.tm_isdst = -1;
42165         if (mktime(&time_str) == -1)
42166             (void)puts("-unknown-");
42167         else {
42168             (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
42169             (void)puts(daybuf);
42170         }
42171         return 0;
42172     }

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), clock(), ctime(), difftime(), gmtime(), localtime(), strftime(), strptime(), time, tzset(), utime()

XBD [Section 4.15](#) (on page 100), [<time.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ANSI C standard.

Issue 6

Extensions beyond the ISO C standard are marked.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/58 is applied, updating the RETURN VALUE and ERRORS sections to add the optional [Eoverflow] error as a CX extension.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/59 is applied, adding the *tzset()* function to the SEE ALSO section.

42192 **NAME**42193 mlock, munlock — lock or unlock a range of process address space (**REALTIME**)42194 **SYNOPSIS**

```
42195 MLR #include <sys/mman.h>
42196 int mlock(const void *addr, size_t len);
42197 int munlock(const void *addr, size_t len);
```

42198 **DESCRIPTION**

42199 The *mlock()* function shall cause those whole pages containing any part of the address space of
 42200 the process starting at address *addr* and continuing for *len* bytes to be memory-resident until
 42201 unlocked or until the process exits or *execs* another process image. The implementation may
 42202 require that *addr* be a multiple of {PAGESIZE}.

42203 The *munlock()* function shall unlock those whole pages containing any part of the address space
 42204 of the process starting at address *addr* and continuing for *len* bytes, regardless of how many
 42205 times *mlock()* has been called by the process for any of the pages in the specified range. The
 42206 implementation may require that *addr* be a multiple of {PAGESIZE}.

42207 If any of the pages in the range specified to a call to *munlock()* are also mapped into the address
 42208 spaces of other processes, any locks established on those pages by another process are
 42209 unaffected by the call of this process to *munlock()*. If any of the pages in the range specified by a
 42210 call to *munlock()* are also mapped into other portions of the address space of the calling process
 42211 outside the range specified, any locks established on those pages via the other mappings are also
 42212 unaffected by this call.

42213 Upon successful return from *mlock()*, pages in the specified range shall be locked and memory-
 42214 resident. Upon successful return from *munlock()*, pages in the specified range shall be unlocked
 42215 with respect to the address space of the process. Memory residency of unlocked pages is
 42216 unspecified.

42217 The appropriate privilege is required to lock process memory with *mlock()*.

42218 **RETURN VALUE**

42219 Upon successful completion, the *mlock()* and *munlock()* functions shall return a value of zero.
 42220 Otherwise, no change is made to any locks in the address space of the process, and the function
 42221 shall return a value of -1 and set *errno* to indicate the error.

42222 **ERRORS**

42223 The *mlock()* and *munlock()* functions shall fail if:

42224 [ENOMEM] Some or all of the address range specified by the *addr* and *len* arguments does
 42225 not correspond to valid mapped pages in the address space of the process.

42226 The *mlock()* function shall fail if:

42227 [EAGAIN] Some or all of the memory identified by the operation could not be locked
 42228 when the call was made.

42229 The *mlock()* and *munlock()* functions may fail if:

42230 [EINVAL] The *addr* argument is not a multiple of {PAGESIZE}.

42231 The *mlock()* function may fail if:

42232 [ENOMEM] Locking the pages mapped by the specified range would exceed an
 42233 implementation-defined limit on the amount of memory that the process may
 42234 lock.

mlock()

42235 [EPERM] The calling process does not have the appropriate privilege to perform the
 42236 requested operation.

EXAMPLES

42237 None.
 42238

APPLICATION USAGE

42239 None.
 42240

RATIONALE

42241 None.
 42242

FUTURE DIRECTIONS

42243 None.
 42244

SEE ALSO

42245 *exec*, *exit()*, *fork()*, *mlockall()*, *munmap()*
 42246

42247 XBD <[sys/mman.h](#)>

CHANGE HISTORY

42248 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
 42249

Issue 6

42250 The *mlock()* and *munlock()* functions are marked as part of the Range Memory Locking option.
 42251

42252 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 42253 implementation does not support the Range Memory Locking option.

42254 **NAME**
 42255 `mlockall`, `munlockall` — lock/unlock the address space of a process (**REALTIME**)

42256 **SYNOPSIS**

```
42257 ML #include <sys/mman.h>
42258 int mlockall(int flags);
42259 int munlockall(void);
```

42260 **DESCRIPTION**

42261 The `mlockall()` function shall cause all of the pages mapped by the address space of a process to
 42262 be memory-resident until unlocked or until the process exits or *execs* another process image. The
 42263 *flags* argument determines whether the pages to be locked are those currently mapped by the
 42264 address space of the process, those that are mapped in the future, or both. The *flags* argument is
 42265 constructed from the bitwise-inclusive OR of one or more of the following symbolic constants,
 42266 defined in `<sys/mman.h>`:

42267 `MCL_CURRENT` Lock all of the pages currently mapped into the address space of the process.

42268 `MCL_FUTURE` Lock all of the pages that become mapped into the address space of the
 42269 process in the future, when those mappings are established.

42270 If `MCL_FUTURE` is specified, and the automatic locking of future mappings eventually causes
 42271 the amount of locked memory to exceed the amount of available physical memory or any other
 42272 implementation-defined limit, the behavior is implementation-defined. The manner in which the
 42273 implementation informs the application of these situations is also implementation-defined.

42274 The `munlockall()` function shall unlock all currently mapped pages of the address space of the
 42275 process. Any pages that become mapped into the address space of the process after a call to
 42276 `munlockall()` shall not be locked, unless there is an intervening call to `mlockall()` specifying
 42277 `MCL_FUTURE` or a subsequent call to `mlockall()` specifying `MCL_CURRENT`. If pages mapped
 42278 into the address space of the process are also mapped into the address spaces of other processes
 42279 and are locked by those processes, the locks established by the other processes shall be
 42280 unaffected by a call by this process to `munlockall()`.

42281 Upon successful return from the `mlockall()` function that specifies `MCL_CURRENT`, all currently
 42282 mapped pages of the address space of the process shall be memory-resident and locked. Upon
 42283 return from the `munlockall()` function, all currently mapped pages of the address space of the
 42284 process shall be unlocked with respect to the address space of the process. The memory
 42285 residency of unlocked pages is unspecified.

42286 The appropriate privilege is required to lock process memory with `mlockall()`.

42287 **RETURN VALUE**

42288 Upon successful completion, the `mlockall()` function shall return a value of zero. Otherwise, no
 42289 additional memory shall be locked, and the function shall return a value of `-1` and set *errno* to
 42290 indicate the error. The effect of failure of `mlockall()` on previously existing locks in the address
 42291 space is unspecified.

42292 If it is supported by the implementation, the `munlockall()` function shall always return a value of
 42293 zero. Otherwise, the function shall return a value of `-1` and set *errno* to indicate the error.

42294 **ERRORS**

42295 The `mlockall()` function shall fail if:

mlockall()

42296	[EAGAIN]	Some or all of the memory identified by the operation could not be locked
42297		when the call was made.
42298	[EINVAL]	The <i>flags</i> argument is zero, or includes unimplemented flags.
42299		The <i>mlockall()</i> function may fail if:
42300	[ENOMEM]	Locking all of the pages currently mapped into the address space of the
42301		process would exceed an implementation-defined limit on the amount of
42302		memory that the process may lock.
42303	[EPERM]	The calling process does not have the appropriate privilege to perform the
42304		requested operation.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*exec*, *exit()*, *fork()*, *mlock()*, *munmap()*XBD <[sys/mman.h](#)>**CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6The *mlockall()* and *munlockall()* functions are marked as part of the Process Memory Locking option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Process Memory Locking option.

42323 **NAME**

42324 mmap — map pages of memory

42325 **SYNOPSIS**

42326 #include <sys/mman.h>

42327 void *mmap(void *addr, size_t len, int prot, int flags,
42328 int fildes, off_t off);42329 **DESCRIPTION**42330 The *mmap()* function shall establish a mapping between an address space of a process and a
42331 memory object.42332 The *mmap()* function shall be supported for the following memory objects:

- 42333 • Regular files
- 42334 SHM • Shared memory objects
- 42335 TYM • Typed memory objects

42336 Support for any other type of file is unspecified.

42337 The format of the call is as follows:

42338 *pa*=mmap(*addr*, *len*, *prot*, *flags*, *fildes*, *off*);

42339 The *mmap()* function shall establish a mapping between the address space of the process at an
42340 address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off*
42341 for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and
42342 the values of *flags*, further described below. A successful *mmap()* call shall return *pa* as its result.
42343 The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible
42344 (not necessarily current) address space of the process. The range of bytes starting at *off* and
42345 continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the
42346 memory object represented by *fildes*.

42347 TYM If *fildes* represents a typed memory object opened with either the
42348 POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG
42349 flag, the memory object to be mapped shall be that portion of the typed memory object allocated
42350 by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap()*
42351 is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling
42352 process, *mmap()* shall fail.

42353 The mapping established by *mmap()* shall replace any previous mappings for those whole pages
42354 containing any part of the address space of the process starting at *pa* and continuing for *len*
42355 bytes.

42356 If the size of the mapped file changes after the call to *mmap()* as a result of some other operation
42357 on the mapped file, the effect of references to portions of the mapped region that correspond to
42358 added or removed portions of the file is unspecified.

42359 If *len* is zero, *mmap()* shall fail and no mapping shall be established.

42360 The parameter *prot* determines whether read, write, execute, or some combination of accesses
42361 are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-
42362 inclusive OR of one or more of the other flags in the following table, defined in the
42363 <sys/mman.h> header.

42364
42365
42366
42367
42368

Symbolic Constant	Description
PROT_READ	Data can be read.
PROT_WRITE	Data can be written.
PROT_EXEC	Data can be executed.
PROT_NONE	Data cannot be accessed.

42369
42370

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap()* shall fail.

42371
42372
42373
42374
42375
42376
42377
42378
42379

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT_WRITE has not been set and shall not permit any access where PROT_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. The file descriptor *fdes* shall have been opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fdes* with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

42380
42381

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in `<sys/mman.h>`:

42382
42383
42384
42385

Symbolic Constant	Description
MAP_SHARED	Changes are shared.
MAP_PRIVATE	Changes are private.
MAP_FIXED	Interpret <i>addr</i> exactly.

42386
42387

XSI It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be supported on XSI-conformant systems.

42388
42389
42390
42391
42392
42393
42394

MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory object. If MAP_SHARED is specified, write references shall change the underlying object. If MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP_PRIVATE mapping is established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork()*.

42395
42396
42397

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

42398
42399
42400
42401
42402
42403
42404
42405
42406
42407
42408
42409
42410
42411
42412

TYM When *fdes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, *mmap()* shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fdes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fdes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fdes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two processes map an area of typed memory using the same *off* and *len* values and using file descriptors that refer to the same memory pool (either from the same port or from a different

- 42413 port), both processes shall map the same region of storage.
- 42414 When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of
42415 *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap()* may return MAP_FAILED and set *errno* to
42416 [EINVAL]. If a MAP_FIXED request is successful, the mapping established by *mmap()* replaces
42417 any previous mappings for the pages in the range [*pa,pa+len*) of the process.
- 42418 When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined
42419 manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the
42420 implementation deems suitable for a mapping of *len* bytes to the file. All implementations
42421 interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*,
42422 subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a
42423 process address near which the mapping should be placed. When the implementation selects a
42424 value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.
- 42425 If MAP_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off*
42426 parameter, modulo the page size as returned by *sysconf()* when passed _SC_PAGESIZE or
42427 _SC_PAGE_SIZE. The implementation may require that *off* is a multiple of the page size. If
42428 MAP_FIXED is specified, the implementation may require that *addr* is a multiple of the page
42429 size. The system performs mapping operations over whole pages. Thus, while the parameter *len*
42430 need not meet a size or alignment constraint, the system shall include, in any mapping
42431 operation, any partial page specified by the address range starting at *pa* and continuing for *len*
42432 bytes.
- 42433 The system shall always zero-fill any partial page at the end of an object. Further, the system
42434 shall never write out any modified portions of the last page of an object which are beyond its
42435 end. References within the address range starting at *pa* and continuing for *len* bytes to whole
42436 pages following the end of an object shall result in delivery of a SIGBUS signal.
- 42437 An implementation may generate SIGBUS signals when a reference would cause an error in the
42438 mapped object, such as out-of-space condition.
- 42439 The *mmap()* function shall add an extra reference to the file associated with the file descriptor
42440 *fildev* which is not removed by a subsequent *close()* on that file descriptor. This reference shall be
42441 removed when there are no more mappings to the file.
- 42442 The last data access timestamp of the mapped file may be marked for update at any time |
42443 between the *mmap()* call and the corresponding *munmap()* call. The initial read or write |
42444 reference to a mapped region shall cause the file's last data access timestamp to be marked for |
42445 update if it has not already been marked for update.
- 42446 The last data modification and last file status change timestamps of a file that is mapped with |
42447 MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval |
42448 between a write reference to the mapped region and the next call to *msync()* with MS_ASYNC or |
42449 MS_SYNC for that portion of the file by any process. If there is no such call and if the |
42450 underlying file is modified as a result of a write reference, then these timestamps shall be |
42451 marked for update at some time after the write reference.
- 42452 There may be implementation-defined limits on the number of memory regions that can be
42453 mapped (per process or per system).
- 42454 XSI If such a limit is imposed, whether the number of memory regions that can be mapped by a
42455 process is decreased by the use of *shmat()* is implementation-defined.
- 42456 If *mmap()* fails for reasons other than [EBADF], [EINVAL], or [ENOTSUP], some of the
42457 mappings in the address range starting at *addr* and continuing for *len* bytes may have been
42458 unmapped.

mmap()

42459

RETURN VALUE

42460

Upon successful completion, the *mmap()* function shall return the address at which the mapping was placed (*pa*); otherwise, it shall return a value of `MAP_FAILED` and set *errno* to indicate the error. The symbol `MAP_FAILED` is defined in the `<sys/mman.h>` header. No successful return from *mmap()* shall return the value `MAP_FAILED`.

42461

42462

42463

42464

ERRORS

42465

The *mmap()* function shall fail if:

42466

[EACCES] The *fildev* argument is not open for read, regardless of the protection specified, or *fildev* is not open for write and `PROT_WRITE` was specified for a `MAP_SHARED` type mapping.

42467

42468

42469 ML

[EAGAIN] The mapping could not be locked in memory, if required by *mlockall()*, due to a lack of resources.

42470

42471

[EBADF] The *fildev* argument is not a valid open file descriptor.

42472

[EINVAL] The value of *len* is zero.

42473

42474

[EINVAL] The value of *flags* is invalid (neither `MAP_PRIVATE` nor `MAP_SHARED` is set).

42475

42476

[EMFILE] The number of mapped regions would exceed an implementation-defined limit (per process or per system).

42477

[ENODEV] The *fildev* argument refers to a file whose type is not supported by *mmap()*.

42478

42479

42480

[ENOMEM] `MAP_FIXED` was specified, and the range [*addr*,*addr+len*) exceeds that allowed for the address space of a process; or, if `MAP_FIXED` was not specified and there is insufficient room in the address space to effect the mapping.

42481 ML

[ENOMEM] The mapping could not be locked in memory, if required by *mlockall()*, because it would require more space than the system is able to supply.

42482

42483 TYM

[ENOMEM] Not enough unallocated memory resources remain in the typed memory object designated by *fildev* to allocate *len* bytes.

42484

42485

42486

[ENOTSUP] `MAP_FIXED` or `MAP_PRIVATE` was specified in the *flags* argument and the implementation does not support this functionality.

42487

42488

The implementation does not support the combination of accesses requested in the *prot* argument.

42489

[ENXIO] Addresses in the range [*off*,*off+len*) are invalid for the object specified by *fildev*.

42490

42491

[ENXIO] `MAP_FIXED` was specified in *flags* and the combination of *addr*, *len*, and *off* is invalid for the object specified by *fildev*.

42492 TYM

[ENXIO] The *fildev* argument refers to a typed memory object that is not accessible from the calling process.

42493

42494

42495

[EOVERFLOW] The file is a regular file and the value of *off* plus *len* exceeds the offset maximum established in the open file description associated with *fildev*.

42496

The *mmap()* function may fail if:

42497

42498

42499

[EINVAL] The *addr* argument (if `MAP_FIXED` was specified) or *off* is not a multiple of the page size as returned by *sysconf()*, or is considered invalid by the implementation.

EXAMPLES

None.

APPLICATION USAGE

Use of *mmap()* may reduce the amount of memory available to other memory allocation functions.

Use of *MAP_FIXED* may result in unspecified behavior in further use of *malloc()* and *shmat()*. The use of *MAP_FIXED* is discouraged, as it may prevent an implementation from making the most effective use of resources. Most implementations require that *off* and *addr* are multiples of the page size as returned by *sysconf()*.

The application must ensure correct synchronization when using *mmap()* in conjunction with any other file access method, such as *read()* and *write()*, standard input/output, and *shmat()*.

The *mmap()* function allows access to resources via address space manipulations, instead of *read()/write()*. Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use *mmap()*, the following:

```
fildes = open(...)
lseek(fildes, some_offset)
read(fildes, buf, len)
/* Use data in buf. */
```

becomes:

```
fildes = open(...)
address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
/* Use data at address. */
```

RATIONALE

After considering several other alternatives, it was decided to adopt the *mmap()* definition found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is minimal, in that it describes only what has been built, and what appears to be necessary for a general and portable mapping facility.

Note that while *mmap()* was first designed for mapping files, it is actually a general-purpose mapping facility. It can be used to map any appropriate object, such as memory, files, devices, and so on, into the address space of a process.

When a mapping is established, it is possible that the implementation may need to map more than is requested into the address space of the process because of hardware requirements. An application, however, cannot count on this behavior. Implementations that do not use a paged architecture may simply allocate a common memory region and return the address of it; such implementations probably do not allocate any more than is necessary. References past the end of the requested area are unspecified.

If an application requests a mapping that would overlay existing mappings in the process, it might be desirable that an implementation detect this and inform the application. However, the default, portable (not *MAP_FIXED*) operation does not overlay existing mappings. On the other hand, if the program specifies a fixed address mapping (which requires some implementation knowledge to determine a suitable address, if the function is supported at all), then the program is presumed to be successfully managing its own address space and should be trusted when it asks to map over existing data structures. Furthermore, it is also desirable to make as few system calls as possible, and it might be considered onerous to require an *munmap()* before an *mmap()* to the same address range. This volume of POSIX.1-200x specifies that the new mappings replace any existing mappings, following existing practice in this regard.

It is not expected that all hardware implementations are able to support all combinations of

42548 permissions at all addresses. Implementations are required to disallow write access to mappings
 42549 without write permission and to disallow access to mappings without any access permission.
 42550 Other than these restrictions, implementations may allow access types other than those
 42551 requested by the application. For example, if the application requests only PROT_WRITE, the
 42552 implementation may also allow read access. A call to *mmap()* fails if the implementation cannot
 42553 support allowing all the access requested by the application. For example, some
 42554 implementations cannot support a request for both write access and execute access
 42555 simultaneously. All implementations must support requests for no access, read access, write
 42556 access, and both read and write access. Strictly conforming code must only rely on the required
 42557 checks. These restrictions allow for portability across a wide range of hardware.

42558 The MAP_FIXED address treatment is likely to fail for non-page-aligned values and for certain
 42559 architecture-dependent address ranges. Conforming implementations cannot count on being
 42560 able to choose address values for MAP_FIXED without utilizing non-portable, implementation-
 42561 defined knowledge. Nonetheless, MAP_FIXED is provided as a standard interface conforming
 42562 to existing practice for utilizing such knowledge when it is available.

42563 Similarly, in order to allow implementations that do not support virtual addresses, support for
 42564 directly specifying any mapping addresses via MAP_FIXED is not required and thus a
 42565 conforming application may not count on it.

42566 The MAP_PRIVATE function can be implemented efficiently when memory protection hardware
 42567 is available. When such hardware is not available, implementations can implement such
 42568 “mappings” by simply making a real copy of the relevant data into process private memory,
 42569 though this tends to behave similarly to *read()*.

42570 The function has been defined to allow for many different models of using shared memory.
 42571 However, all uses are not equally portable across all machine architectures. In particular, the
 42572 *mmap()* function allows the system as well as the application to specify the address at which to
 42573 map a specific region of a memory object. The most portable way to use the function is always to
 42574 let the system choose the address, specifying NULL as the value for the argument *addr* and not
 42575 to specify MAP_FIXED.

42576 If it is intended that a particular region of a memory object be mapped at the same address in a
 42577 group of processes (on machines where this is even possible), then MAP_FIXED can be used to
 42578 pass in the desired mapping address. The system can still be used to choose the desired address
 42579 if the first such mapping is made without specifying MAP_FIXED, and then the resulting
 42580 mapping address can be passed to subsequent processes for them to pass in via MAP_FIXED.
 42581 The availability of a specific address range cannot be guaranteed, in general.

42582 The *mmap()* function can be used to map a region of memory that is larger than the current size
 42583 of the object. Memory access within the mapping but beyond the current end of the underlying
 42584 objects may result in SIGBUS signals being sent to the process. The reason for this is that the size
 42585 of the object can be manipulated by other processes and can change at any moment. The
 42586 implementation should tell the application that a memory reference is outside the object where
 42587 this can be detected; otherwise, written data may be lost and read data may not reflect actual
 42588 data in the object.

42589 Note that references beyond the end of the object do not extend the object as the new end cannot
 42590 be determined precisely by most virtual memory hardware. Instead, the size can be directly
 42591 manipulated by *ftruncate()*.

42592 Process memory locking does apply to shared memory regions, and the MEMLOCK_FUTURE
 42593 argument to *mlockall()* can be relied upon to cause new shared memory regions to be
 42594 automatically locked.

42595 Existing implementations of *mmap()* return the value `-1` when unsuccessful. Since the casting of
 42596 this value to type `void *` cannot be guaranteed by the ISO C standard to be distinct from a
 42597 successful value, this volume of POSIX.1-200x defines the symbol MAP_FAILED, which a

42598 conforming implementation does not return as the result of a successful call.

42599 **FUTURE DIRECTIONS**

42600 None.

42601 **SEE ALSO**

42602 *exec, fcntl(), fork(), lockf(), msync(), munmap(), mprotect(), posix_typed_mem_open(), shmat(),*
42603 *sysconf()*

42604 XBD <sys/mman.h>

42605 **CHANGE HISTORY**

42606 First released in Issue 4, Version 2.

42607 **Issue 5**

42608 Moved from X/OPEN UNIX extension to BASE.

42609 Aligned with *mmap()* in the POSIX Realtime Extension as follows:

- 42610 • The DESCRIPTION is extensively reworded.
- 42611 • The [EAGAIN] and [ENOTSUP] mandatory error conditions are added.
- 42612 • New cases of [ENOMEM] and [ENXIO] are added as mandatory error conditions.
- 42613 • The value returned on failure is the value of the constant MAP_FAILED; this was
42614 previously defined as -1.

42615 Large File Summit extensions are added.

42616 **Issue 6**

42617 The *mmap()* function is marked as part of the Memory Mapped Files option.

42618 The Open Group Corrigendum U028/6 is applied, changing (void *)-1 to MAP_FAILED.

42619 The following new requirements on POSIX implementations derive from alignment with the
42620 Single UNIX Specification:

- 42621 • The DESCRIPTION is updated to describe the use of MAP_FIXED.
- 42622 • The DESCRIPTION is updated to describe the addition of an extra reference to the file
42623 associated with the file descriptor passed to *mmap()*.
- 42624 • The DESCRIPTION is updated to state that there may be implementation-defined limits on
42625 the number of memory regions that can be mapped.
- 42626 • The DESCRIPTION is updated to describe constraints on the alignment and size of the *off*
42627 argument.
- 42628 • The [EINVAL] and [EMFILE] error conditions are added.
- 42629 • The [EOVERFLOW] error condition is added. This change is to support large files.

42630 The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- 42631 • The DESCRIPTION is updated to describe the cases when MAP_PRIVATE and
42632 MAP_FIXED need not be supported.

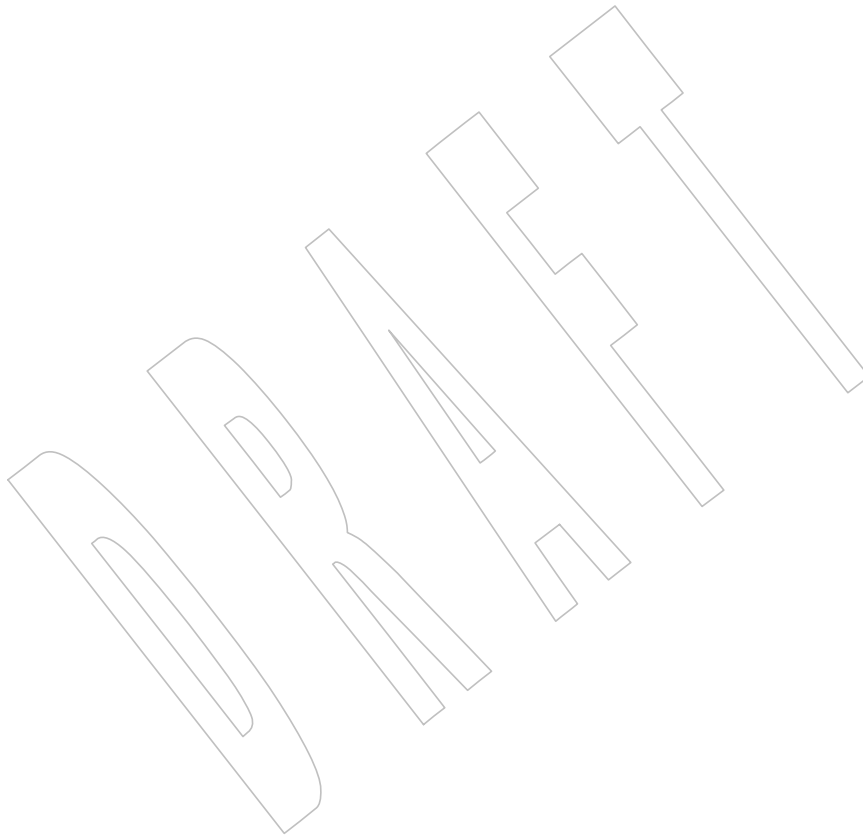
42633 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 42634 • Semantics for typed memory objects are added to the DESCRIPTION.
- 42635 • New [ENOMEM] and [ENXIO] errors are added to the ERRORS section.
- 42636 • The *posix_typed_mem_open()* function is added to the SEE ALSO section.

42637 The normative text is updated to avoid use of the term “must” for application requirements.

mmap()

- 42638 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/34 is applied, changing the margin code
42639 in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).
- 42640 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/60 is applied, updating the
42641 DESCRIPTION and ERRORS sections to add the [EINVAL] error when *len* is zero.
- 42642 **Issue 7**
- 42643 Austin Group Interpretations 1003.1-2001 #078 and #079 are applied, clarifying page alignment
42644 requirements and adding a note about the state of synchronization objects becoming undefined
42645 when a shared region is unmapped.
- 42646 Functionality relating to the Memory Protection and Memory Mapped Files options is moved to
42647 the Base.
- 42648 Changes are made related to support for finegrained timestamps. +



42649 **NAME**

42650 modf, modff, modfl — decompose a floating-point number

42651 **SYNOPSIS**

42652 #include <math.h>

42653 double modf(double *x*, double **iptr*);42654 float modff(float *value*, float **iptr*);42655 long double modfl(long double *value*, long double **iptr*);42656 **DESCRIPTION**42657 CX The functionality described on this reference page is aligned with the ISO C standard. Any
42658 conflict between the requirements described here and the ISO C standard is unintentional. This
42659 volume of POSIX.1-200x defers to the ISO C standard.42660 These functions shall break the argument *x* into integral and fractional parts, each of which has
42661 the same sign as the argument. It stores the integral part as a **double** (for the *modf()* function), a
42662 **float** (for the *modff()* function), or a **long double** (for the *modfl()* function), in the object pointed
42663 to by *iptr*.42664 **RETURN VALUE**42665 Upon successful completion, these functions shall return the signed fractional part of *x*.42666 MX If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.42667 If *x* is ±Inf, ±0 shall be returned, and **iptr* shall be set to ±Inf.42668 **ERRORS**

42669 No errors are defined.

42670 **EXAMPLES**

42671 None.

42672 **APPLICATION USAGE**42673 The *modf()* function computes the function result and **iptr* such that:

42674 a = modf(x, iptr) ;

42675 x == a+*iptr ;

42676 allowing for the usual floating-point inaccuracies.

42677 **RATIONALE**

42678 None.

42679 **FUTURE DIRECTIONS**

42680 None.

42681 **SEE ALSO**42682 *frexp()*, *isnan()*, *ldexp()*

42683 XBD <math.h>

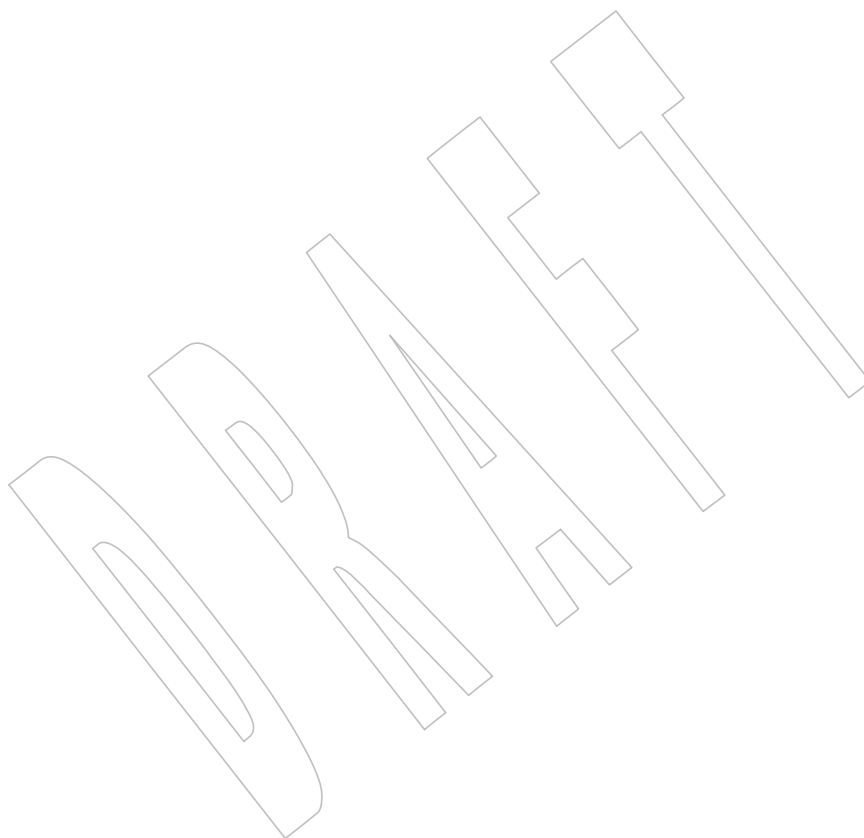
42684 **CHANGE HISTORY**

42685 First released in Issue 1. Derived from Issue 1 of the SVID.

42686 **Issue 5**42687 The DESCRIPTION is updated to indicate how an application should check for an error. This
42688 text was previously published in the APPLICATION USAGE section.

Issue 6

- 42689 The *modff()* and *modfl()* functions are added for alignment with the ISO/IEC 9899:1999
42690 standard.
42691
- 42692 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
42693 revised to align with the ISO/IEC 9899:1999 standard.
- 42694 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
42695 marked.
- 42696 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/35 is applied, correcting the code example
42697 in the APPLICATION USAGE section.



42698 **NAME**
 42699 `mprotect` — set protection of memory mapping

42700 **SYNOPSIS**
 42701 `#include <sys/mman.h>`

42702 `int mprotect(void *addr, size_t len, int prot);`

42703 **DESCRIPTION**

42704 The `mprotect()` function shall change the access protections to be that specified by `prot` for those
 42705 whole pages containing any part of the address space of the process starting at address `addr` and
 42706 continuing for `len` bytes. The parameter `prot` determines whether read, write, execute, or some
 42707 combination of accesses are permitted to the data being mapped. The `prot` argument should be
 42708 either `PROT_NONE` or the bitwise-inclusive OR of one or more of `PROT_READ`, `PROT_WRITE`,
 42709 and `PROT_EXEC`.

42710 If an implementation cannot support the combination of access types specified by `prot`, the call to
 42711 `mprotect()` shall fail.

42712 An implementation may permit accesses other than those specified by `prot`; however, no
 42713 implementation shall permit a write to succeed where `PROT_WRITE` has not been set or shall
 42714 permit any access where `PROT_NONE` alone has been set. Implementations shall support at
 42715 least the following values of `prot`: `PROT_NONE`, `PROT_READ`, `PROT_WRITE`, and the bitwise-
 42716 inclusive OR of `PROT_READ` and `PROT_WRITE`. If `PROT_WRITE` is specified, the application
 42717 shall ensure that it has opened the mapped objects in the specified address range with write
 42718 permission, unless `MAP_PRIVATE` was specified in the original mapping, regardless of whether
 42719 the file descriptors used to map the objects have since been closed.

42720 The implementation may require that `addr` be a multiple of the page size as returned by
 42721 `sysconf()`.

42722 The behavior of this function is unspecified if the mapping was not established by a call to
 42723 `mmap()`.

42724 When `mprotect()` fails for reasons other than `[EINVAL]`, the protections on some of the pages in
 42725 the range `[addr,addr+len)` may have been changed.

42726 **RETURN VALUE**

42727 Upon successful completion, `mprotect()` shall return 0; otherwise, it shall return `-1` and set `errno`
 42728 to indicate the error.

42729 **ERRORS**

42730 The `mprotect()` function shall fail if:

42731 `[EACCES]` The `prot` argument specifies a protection that violates the access permission the
 42732 process has to the underlying memory object.

42733 `[EAGAIN]` The `prot` argument specifies `PROT_WRITE` over a `MAP_PRIVATE` mapping
 42734 and there are insufficient memory resources to reserve for locking the private
 42735 page.

42736 `[ENOMEM]` Addresses in the range `[addr,addr+len)` are invalid for the address space of a
 42737 process, or specify one or more pages which are not mapped.

42738 `[ENOMEM]` The `prot` argument specifies `PROT_WRITE` on a `MAP_PRIVATE` mapping, and
 42739 it would require more space than the system is able to supply for locking the
 42740 private pages, if required.

mprotect()

42741 [ENOTSUP] The implementation does not support the combination of accesses requested
42742 in the *prot* argument.

42743 The *mprotect()* function may fail if:

42744 [EINVAL] The *addr* argument is not a multiple of the page size as returned by *sysconf()*.

EXAMPLES

42745 None.
42746

APPLICATION USAGE

42747 Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.
42748

RATIONALE

42749 None.
42750

FUTURE DIRECTIONS

42751 None.
42752

SEE ALSO

42753 *mmap()*, *sysconf()*
42754

42755 XBD <[sys/mman.h](#)>

CHANGE HISTORY

42756 First released in Issue 4, Version 2.
42757

Issue 5

42758 Moved from X/OPEN UNIX extension to BASE.
42759

42760 Aligned with *mprotect()* in the POSIX Realtime Extension as follows:

- 42761 • The DESCRIPTION is largely reworded.
- 42762 • [ENOTSUP] and a second form of [ENOMEM] are added as mandatory error conditions.
- 42763 • [EAGAIN] is moved from the optional to the mandatory error conditions.

Issue 6

42764 The *mprotect()* function is marked as part of the Memory Protection option.
42765

42766 The following new requirements on POSIX implementations derive from alignment with the
42767 Single UNIX Specification:

- 42768 • The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
42769 of the page size as returned by *sysconf()*.
- 42770 • The [EINVAL] error condition is added.

42771 The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

42772 SD5-XSH-ERN-22 is applied, deleting erroneous APPLICATION USAGE.
42773

42774 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment
42775 requirements.

42776 The *mprotect()* function is moved from the Memory Protection option to the Base.

42777 **NAME**
 42778 `mq_close` — close a message queue (**REALTIME**)

42779 **SYNOPSIS**

42780 MSG `#include <mqqueue.h>`
 42781 `int mq_close(mqd_t mqdes);`

42782 **DESCRIPTION**

42783 The `mq_close()` function shall remove the association between the message queue descriptor,
 42784 `mqdes`, and its message queue. The results of using this message queue descriptor after successful
 42785 return from this `mq_close()`, and until the return of this message queue descriptor from a
 42786 subsequent `mq_open()`, are undefined.

42787 If the process has successfully attached a notification request to the message queue via this
 42788 `mqdes`, this attachment shall be removed, and the message queue is available for another process
 42789 to attach for notification.

42790 **RETURN VALUE**

42791 Upon successful completion, the `mq_close()` function shall return a value of zero; otherwise, the
 42792 function shall return a value of `-1` and set `errno` to indicate the error.

42793 **ERRORS**

42794 The `mq_close()` function shall fail if:

42795 [EBADF] The `mqdes` argument is not a valid message queue descriptor.

42796 **EXAMPLES**

42797 None.

42798 **APPLICATION USAGE**

42799 None.

42800 **RATIONALE**

42801 None.

42802 **FUTURE DIRECTIONS**

42803 None.

42804 **SEE ALSO**

42805 [mq_open\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

42806 XBD [<mqqueue.h>](#)

42807 **CHANGE HISTORY**

42808 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

42809 **Issue 6**

42810 The `mq_close()` function is marked as part of the Message Passing option.

42811 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 42812 implementation does not support the Message Passing option.

42813 **NAME**42814 `mq_getattr` — get message queue attributes (**REALTIME**)42815 **SYNOPSIS**

```
42816 MSG #include <mqueue.h>
42817 int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

42818 **DESCRIPTION**42819 The `mq_getattr()` function shall obtain status information and attributes of the message queue
42820 and the open message queue description associated with the message queue descriptor.42821 The `mqdes` argument specifies a message queue descriptor.42822 The results shall be returned in the **mq_attr** structure referenced by the `mqstat` argument.42823 Upon return, the following members shall have the values associated with the open message
42824 queue description as set when the message queue was opened and as modified by subsequent
42825 `mq_setattr()` calls: `mq_flags`.42826 The following attributes of the message queue shall be returned as set at message queue
42827 creation: `mq_maxmsg`, `mq_msgsize`.42828 Upon return, the following members within the **mq_attr** structure referenced by the `mqstat`
42829 argument shall be set to the current state of the message queue:42830 `mq_curmsgs` The number of messages currently on the queue.42831 **RETURN VALUE**42832 Upon successful completion, the `mq_getattr()` function shall return zero. Otherwise, the function
42833 shall return `-1` and set `errno` to indicate the error.42834 **ERRORS**42835 The `mq_getattr()` function may fail if:42836 [EBADF] The `mqdes` argument is not a valid message queue descriptor.42837 **EXAMPLES**

42838 None.

42839 **APPLICATION USAGE**

42840 None.

42841 **RATIONALE**

42842 None.

42843 **FUTURE DIRECTIONS**

42844 None.

42845 **SEE ALSO**42846 `mq_open()`, `mq_send()`, `mq_setattr()`, `msgctl()`, `msgget()`, `msgrcv()`, `msgsnd()` -42847 XBD `<mqueue.h>` |42848 **CHANGE HISTORY**

42849 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

42850

Issue 6

42851

The `mq_getattr()` function is marked as part of the Message Passing option.

42852

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

42853

42854

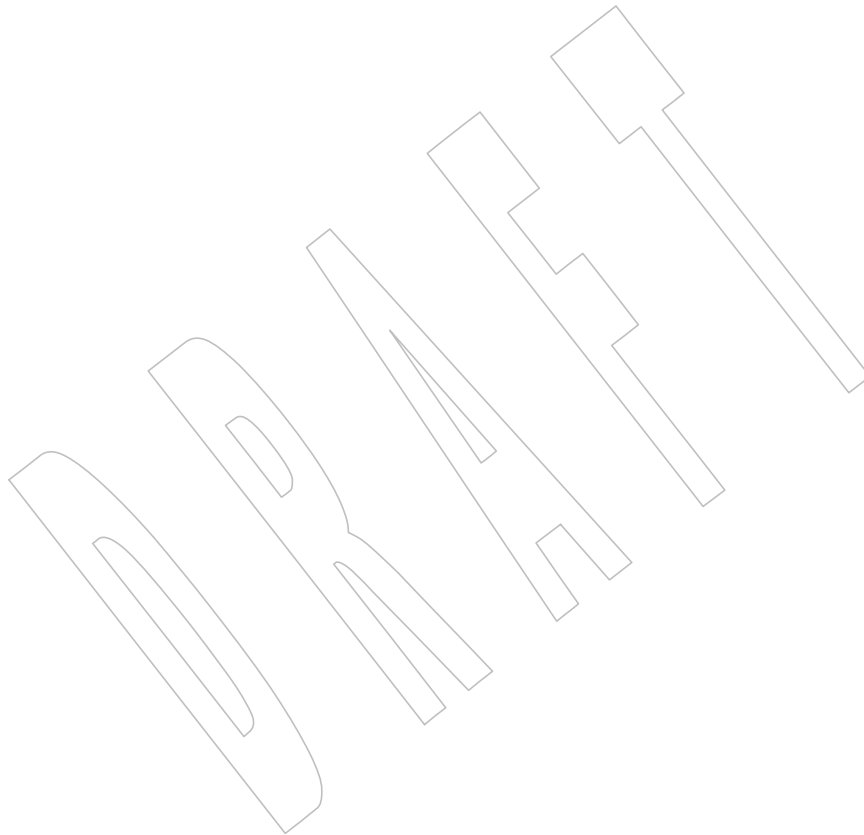
The `mq_timedsend()` function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

42855

42856

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/61 is applied, updating the ERRORS section to change the [EBADF] error from mandatory to optional.

42857



42858 **NAME**42859 `mq_notify` — notify process that a message is available (**REALTIME**)42860 **SYNOPSIS**

```
42861 MSG #include <mqueue.h>
42862 int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

42863 **DESCRIPTION**

42864 If the argument *notification* is not NULL, this function shall register the calling process to be
 42865 notified of message arrival at an empty message queue associated with the specified message
 42866 queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to
 42867 the process when the message queue transitions from empty to non-empty. At any time, only
 42868 one process may be registered for notification by a message queue. If the calling process or any
 42869 other process has already registered for notification of message arrival at the specified message
 42870 queue, subsequent attempts to register for that message queue shall fail.

42871 If *notification* is NULL and the process is currently registered for notification by the specified
 42872 message queue, the existing registration shall be removed.

42873 When the notification is sent to the registered process, its registration shall be removed. The
 42874 message queue shall then be available for registration.

42875 If a process has registered for notification of message arrival at a message queue and some
 42876 thread is blocked in *mq_receive()* or *mq_timedreceive()* waiting to receive a message when a
 42877 message arrives at the queue, the arriving message shall satisfy the appropriate *mq_receive()* or
 42878 *mq_timedreceive()*, respectively. The resulting behavior is as if the message queue remains empty,
 42879 and no notification shall be sent.

42880 **RETURN VALUE**

42881 Upon successful completion, the *mq_notify()* function shall return a value of zero; otherwise, the
 42882 function shall return a value of -1 and set *errno* to indicate the error.

42883 **ERRORS**

42884 The *mq_notify()* function shall fail if:

42885 [EBADF] The *mqdes* argument is not a valid message queue descriptor.

42886 [EBUSY] A process is already registered for notification by the message queue.

42887 The *mq_notify()* function may fail if:

42888 [EINVAL] The *notification* argument is NULL and the process is currently not registered.

42889 **EXAMPLES**

42890 None.

42891 **APPLICATION USAGE**

42892 None.

42893 **RATIONALE**

42894 None.

42895 **FUTURE DIRECTIONS**

42896 None.

42897
42898
42899
42900
42901
42902
42903
42904
42905
42906
42907
42908
42909
42910

SEE ALSO

mq_open(), *mq_send()*, *mq_receive()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

XBD <[mqqueue.h](#)>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *mq_notify()* function is marked as part of the Message Passing option.

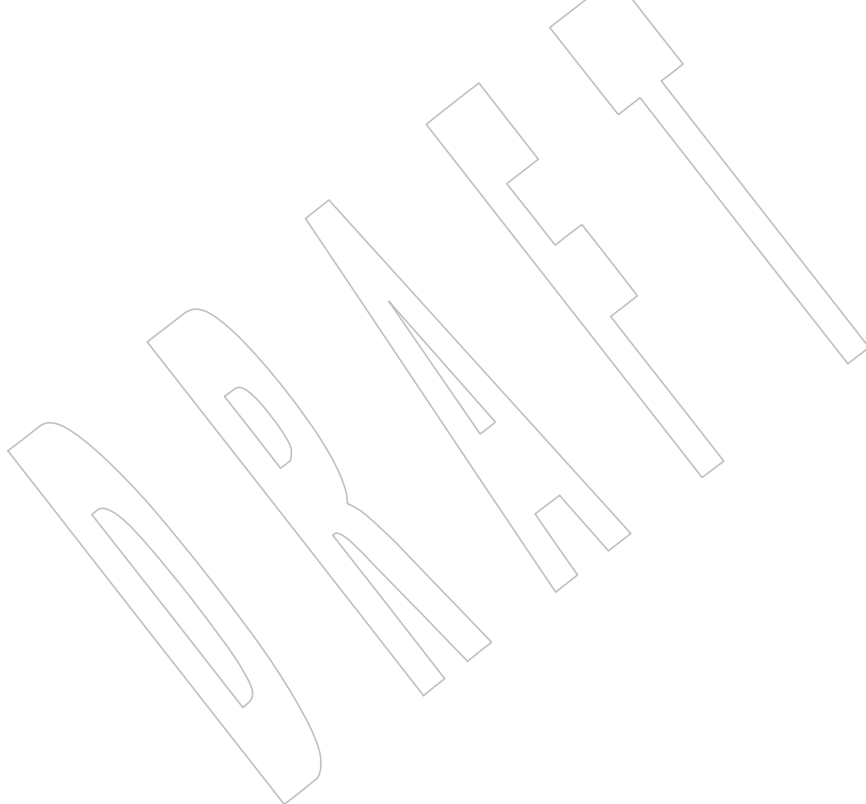
The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

Issue 7

SD5-XSH-ERN-38 is applied, adding the *mq_timedreceive()* function to the DESCRIPTION.

Austin Group Interpretation 1003.1-2001 #032 is applied, adding the [EINVAL] error.



42911 **NAME**42912 mq_open — open a message queue (**REALTIME**)42913 **SYNOPSIS**

```
42914 MSG #include <mqueue.h>
42915 mqd_t mq_open(const char *name, int oflag, ...);
```

42916 **DESCRIPTION**

42917 The *mq_open()* function shall establish the connection between a process and a message queue
 42918 with a message queue descriptor. It shall create an open message queue description that refers to
 42919 the message queue, and a message queue descriptor that refers to that open message queue
 42920 description. The message queue descriptor is used by other functions to refer to that message
 42921 queue. The *name* argument points to a string naming a message queue. It is unspecified whether
 42922 the name appears in the file system and is visible to other functions that take pathnames as
 42923 arguments. The *name* argument conforms to the construction rules for a pathname, except that
 42924 the interpretation of slash characters other than the leading slash character in *name* is
 42925 implementation-defined, and that the length limits for the *name* argument are implementation-
 42926 defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If
 42927 *name* begins with the slash character, then processes calling *mq_open()* with the same value of
 42928 *name* shall refer to the same message queue object, as long as that name has not been removed. If
 42929 *name* does not begin with the slash character, the effect is implementation-defined. If the *name*
 42930 argument is not the name of an existing message queue and creation is not requested, *mq_open()*
 42931 shall fail and return an error.

42932 A message queue descriptor may be implemented using a file descriptor, in which case
 42933 applications can open up to at least {OPEN_MAX} file and message queues.

42934 The *oflag* argument requests the desired receive and/or send access to the message queue. The
 42935 requested access permission to receive messages or send messages shall be granted if the calling
 42936 process would be granted read or write access, respectively, to an equivalently protected file.

42937 The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications
 42938 shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

42939 **O_RDONLY** Open the message queue for receiving messages. The process can use the
 42940 returned message queue descriptor with *mq_receive()*, but not *mq_send()*. A
 42941 message queue may be open multiple times in the same or different processes
 42942 for receiving messages.

42943 **O_WRONLY** Open the queue for sending messages. The process can use the returned
 42944 message queue descriptor with *mq_send()* but not *mq_receive()*. A message
 42945 queue may be open multiple times in the same or different processes for
 42946 sending messages.

42947 **O_RDWR** Open the queue for both receiving and sending messages. The process can use
 42948 any of the functions allowed for **O_RDONLY** and **O_WRONLY**. A message
 42949 queue may be open multiple times in the same or different processes for
 42950 sending messages.

42951 Any combination of the remaining flags may be specified in the value of *oflag*:

42952 **O_CREAT** Create a message queue. It requires two additional arguments: *mode*, which
 42953 shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr**
 42954 structure. If the pathname *name* has already been used to create a message
 42955 queue that still exists, then this flag shall have no effect, except as noted under

42956 O_EXCL. Otherwise, a message queue shall be created without any messages
 42957 in it. The user ID of the message queue shall be set to the effective user ID of
 42958 the process. The group ID of the message queue shall be set to the effective
 42959 group ID of the process; however, if the *name* argument is visible in the file
 42960 system, the group ID may be set to the group ID of the containing directory.
 42961 When bits in *mode* other than the file permission bits are specified, the effect is
 42962 unspecified. If *attr* is NULL, the message queue shall be created with
 42963 implementation-defined default message queue attributes. If *attr* is non-NULL
 42964 and the calling process has the appropriate privilege on *name*, the message
 42965 queue *mq_maxmsg* and *mq_msgsize* attributes shall be set to the values of the
 42966 corresponding members in the **mq_attr** structure referred to by *attr*. If *attr* is
 42967 non-NULL, but the calling process does not have the appropriate privilege on
 42968 *name*, the *mq_open()* function shall fail and return an error without creating
 42969 the message queue.

42970 O_EXCL If O_EXCL and O_CREAT are set, *mq_open()* shall fail if the message queue
 42971 *name* exists. The check for the existence of the message queue and the creation
 42972 of the message queue if it does not exist shall be atomic with respect to other
 42973 threads executing *mq_open()* naming the same *name* with O_EXCL and
 42974 O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is
 42975 undefined.

42976 O_NONBLOCK Determines whether an *mq_send()* or *mq_receive()* waits for resources or
 42977 messages that are not currently available, or fails with *errno* set to [EAGAIN];
 42978 see *mq_send()* and *mq_receive()* for details.

42979 The *mq_open()* function does not add or remove messages from the queue.

42980 RETURN VALUE

42981 Upon successful completion, the function shall return a message queue descriptor; otherwise,
 42982 the function shall return (**mqd_t**)-1 and set *errno* to indicate the error.

42983 ERRORS

42984 The *mq_open()* function shall fail if:

42985 [EACCES] The message queue exists and the permissions specified by *oflag* are denied, or
 42986 the message queue does not exist and permission to create the message queue
 42987 is denied.

42988 [EEXIST] O_CREAT and O_EXCL are set and the named message queue already exists.

42989 [EINTR] The *mq_open()* function was interrupted by a signal.

42990 [EINVAL] The *mq_open()* function is not supported for the given name.

42991 [EINVAL] O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either
 42992 *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

42993 [EMFILE] Too many message queue descriptors or file descriptors are currently in use by
 42994 this process.

42995 [ENFILE] Too many message queues are currently open in the system.

42996 [ENOENT] O_CREAT is not set and the named message queue does not exist.

42997 [ENOSPC] There is insufficient space for the creation of the new message queue.

42998 If any of the following conditions occur, the *mq_open()* function may return (**mqd_t**)-1 and set
 42999 *errno* to the corresponding value.

mq_open()*System Interfaces*

43000 [ENAMETOOLONG]
 43001 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 43002 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 43003 systems, or has a pathname component that is longer than
 43004 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 43005 longer than {_XOPEN_NAME_MAX} on XSI systems.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_close(), *mq_getattr()*, *mq_receive()*, *mq_send()*, *mq_setattr()*, *mq_unlink()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

XBD <[mqqueue.h](#)>**CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6The *mq_open()* function is marked as part of the Message Passing option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

The *mq_timedreceive()* and *mq_timedsend()* functions are added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION of O_EXCL is updated in response to IEEE PASC Interpretation 1003.1c #48.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/62 is applied, updating the description of the permission bits in the DESCRIPTION. The change is made for consistency with the *shm_open()* and *sem_open()* functions.**Issue 7**Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the message queue.

43035 **NAME**43036 mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)43037 **SYNOPSIS**

```
43038 MSG #include <mqqueue.h>
43039
43039 ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
43040 unsigned *msg_prio);
43041
43041 #include <mqqueue.h>
43042 #include <time.h>
43043
43043 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
43044 size_t msg_len, unsigned *restrict msg_prio,
43045 const struct timespec *restrict abstime);
```

43046 **DESCRIPTION**

43047 The *mq_receive()* function shall receive the oldest of the highest priority message(s) from the
 43048 message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len*
 43049 argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and
 43050 return an error. Otherwise, the selected message shall be removed from the queue and copied to
 43051 the buffer pointed to by the *msg_ptr* argument.

43052 If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

43053 If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the
 43054 location referenced by *msg_prio*.

43055 If the specified message queue is empty and O_NONBLOCK is not set in the message queue
 43056 description associated with *mqdes*, *mq_receive()* shall block until a message is enqueued on the
 43057 message queue or until *mq_receive()* is interrupted by a signal. If more than one thread is waiting
 43058 to receive a message when a message arrives at an empty queue and the Priority Scheduling
 43059 option is supported, then the thread of highest priority that has been waiting the longest shall be
 43060 selected to receive the message. Otherwise, it is unspecified which waiting thread receives the
 43061 message. If the specified message queue is empty and O_NONBLOCK is set in the message
 43062 queue description associated with *mqdes*, no message shall be removed from the queue, and
 43063 *mq_receive()* shall return an error.

43064 The *mq_timedreceive()* function shall receive the oldest of the highest priority messages from the
 43065 message queue specified by *mqdes* as described for the *mq_receive()* function. However, if
 43066 O_NONBLOCK was not specified when the message queue was opened via the *mq_open()*
 43067 function, and no message exists on the queue to satisfy the receive, the wait for such a message
 43068 shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is
 43069 equivalent to *mq_receive()*.

43070 The timeout expires when the absolute time specified by *abstime* passes, as measured by the
 43071 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 43072 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
 43073 call.

43074 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
 43075 be the resolution of the clock on which it is based. The *timespec* argument is defined in the
 43076 **<time.h>** header.

43077 Under no circumstance shall the operation fail with a timeout if a message can be removed from
 43078 the message queue immediately. The validity of the *abstime* parameter need not be checked if a
 43079 message can be removed from the message queue immediately.

mq_receive()

System Interfaces

43080 **RETURN VALUE**

43081 Upon successful completion, the *mq_receive()* and *mq_timedreceive()* functions shall return the
 43082 length of the selected message in bytes and the message shall be removed from the queue.
 43083 Otherwise, no message shall be removed from the queue, the functions shall return a value of -1 ,
 43084 and set *errno* to indicate the error.

43085 **ERRORS**

43086 The *mq_receive()* and *mq_timedreceive()* functions shall fail if:

- 43087 [EAGAIN] O_NONBLOCK was set in the message description associated with *mqdes*, and
 43088 the specified message queue is empty.
- 43089 [EBADF] The *mqdes* argument is not a valid message queue descriptor open for reading.
- 43090 [EMSGSIZE] The specified message buffer size, *msg_len*, is less than the message size
 43091 attribute of the message queue.
- 43092 [EINTR] The *mq_receive()* or *mq_timedreceive()* operation was interrupted by a signal.
- 43093 [EINVAL] The process or thread would have blocked, and the *abstime* parameter
 43094 specified a nanoseconds field value less than zero or greater than or equal to
 43095 1 000 million.
- 43096 [ETIMEDOUT] The O_NONBLOCK flag was not set when the message queue was opened,
 43097 but no message arrived on the queue before the specified timeout expired.

43098 The *mq_receive()* and *mq_timedreceive()* functions may fail if:

- 43099 [EBADMSG] The implementation has detected a data corruption problem with the
 43100 message.

43101 **EXAMPLES**

43102 None.

43103 **APPLICATION USAGE**

43104 None.

43105 **RATIONALE**

43106 None.

43107 **FUTURE DIRECTIONS**

43108 None.

43109 **SEE ALSO**

43110 *mq_open()*, *mq_send()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*, *time*

43111 XBD [<mqqueue.h>](#), [<time.h>](#)

43112 **CHANGE HISTORY**

43113 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43114 **Issue 6**

43115 The *mq_receive()* function is marked as part of the Message Passing option.

43116 The Open Group Corrigendum U021/4 is applied. The DESCRIPTION is changed to refer to
 43117 *msg_len* rather than *maxsize*.

43118 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 43119 implementation does not support the Message Passing option.

43120 The following new requirements on POSIX implementations derive from alignment with the
 43121 Single UNIX Specification:

- 43122
- 43123
- 43124
- In this function it is possible for the return value to exceed the range of the type `ssize_t` (since `size_t` has a larger range of positive values than `ssize_t`). A sentence restricting the size of the `size_t` object is added to the description to resolve this conflict.

43125 The `mq_timedreceive()` function is added for alignment with IEEE Std 1003.1d-1999.

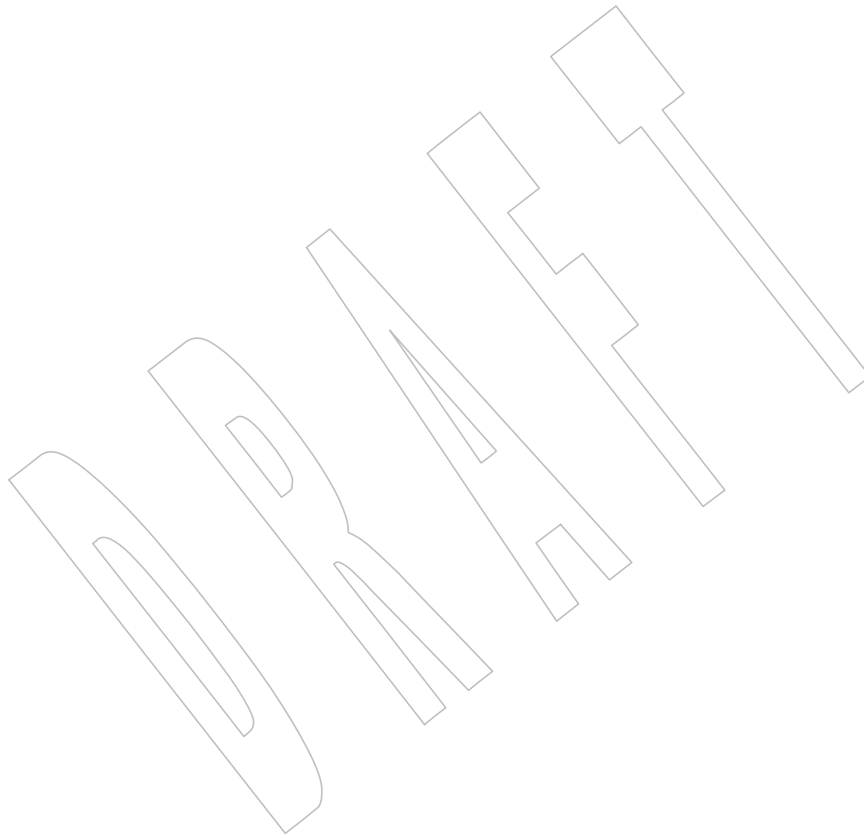
43126 The `restrict` keyword is added to the `mq_timedreceive()` prototype for alignment with the
43127 ISO/IEC 9899:1999 standard.

43128 IEEE PASC Interpretation 1003.1 #109 is applied, correcting the return type for `mq_timedreceive()`
43129 from `int` to `ssize_t`.

Issue 7

43130 The `mq_timedreceive()` function is moved from the Timeouts option to the Base.

43132 Functionality relating to the Timers option is moved to the Base.



43133 **NAME**43134 `mq_send`, `mq_timedsend` — send a message to a message queue (**REALTIME**)43135 **SYNOPSIS**

```

43136 MSG #include <mqqueue.h>
43137
43137 int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43138             unsigned msg_prio);
43139
43139 #include <mqqueue.h>
43140 #include <time.h>
43141
43141 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
43142                 unsigned msg_prio, const struct timespec *abstime);

```

43143 **DESCRIPTION**

43144 The `mq_send()` function shall add the message pointed to by the argument `msg_ptr` to the
 43145 message queue specified by `mqdes`. The `msg_len` argument specifies the length of the message, in
 43146 bytes, pointed to by `msg_ptr`. The value of `msg_len` shall be less than or equal to the `mq_msgsize`
 43147 attribute of the message queue, or `mq_send()` shall fail.

43148 If the specified message queue is not full, `mq_send()` shall behave as if the message is inserted
 43149 into the message queue at the position indicated by the `msg_prio` argument. A message with a
 43150 larger numeric value of `msg_prio` shall be inserted before messages with lower values of
 43151 `msg_prio`. A message shall be inserted after other messages in the queue, if any, with equal
 43152 `msg_prio`. The value of `msg_prio` shall be less than {MQ_PRIO_MAX}.

43153 If the specified message queue is full and `O_NONBLOCK` is not set in the message queue
 43154 description associated with `mqdes`, `mq_send()` shall block until space becomes available to
 43155 enqueue the message, or until `mq_send()` is interrupted by a signal. If more than one thread is
 43156 waiting to send when space becomes available in the message queue and the Priority Scheduling
 43157 option is supported, then the thread of the highest priority that has been waiting the longest
 43158 shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is
 43159 unblocked. If the specified message queue is full and `O_NONBLOCK` is set in the message
 43160 queue description associated with `mqdes`, the message shall not be queued and `mq_send()` shall
 43161 return an error.

43162 The `mq_timedsend()` function shall add a message to the message queue specified by `mqdes` in the
 43163 manner defined for the `mq_send()` function. However, if the specified message queue is full and
 43164 `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, the wait for
 43165 sufficient room in the queue shall be terminated when the specified timeout expires. If
 43166 `O_NONBLOCK` is set in the message queue description, this function shall be equivalent to
 43167 `mq_send()`.

43168 The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the
 43169 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 43170 `abstime`), or if the absolute time specified by `abstime` has already been passed at the time of the
 43171 call.

43172 The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall
 43173 be the resolution of the clock on which it is based. The `timespec` argument is defined in the
 43174 `<time.h>` header.

43175 Under no circumstance shall the operation fail with a timeout if there is sufficient room in the
 43176 queue to add the message immediately. The validity of the `abstime` parameter need not be
 43177 checked when there is sufficient room in the queue.

43178 **RETURN VALUE**

43179 Upon successful completion, the `mq_send()` and `mq_timedsend()` functions shall return a value of
 43180 zero. Otherwise, no message shall be enqueued, the functions shall return `-1`, and `errno` shall be
 43181 set to indicate the error.

43182 **ERRORS**

43183 The `mq_send()` and `mq_timedsend()` functions shall fail if:

- 43184 [EAGAIN] The `O_NONBLOCK` flag is set in the message queue description associated
 43185 with `mqdes`, and the specified message queue is full.
- 43186 [EBADF] The `mqdes` argument is not a valid message queue descriptor open for writing.
- 43187 [EINTR] A signal interrupted the call to `mq_send()` or `mq_timedsend()`.
- 43188 [EINVAL] The value of `msg_prio` was outside the valid range.
- 43189 [EINVAL] The process or thread would have blocked, and the `abstime` parameter
 43190 specified a nanoseconds field value less than zero or greater than or equal to
 43191 1 000 million.
- 43192 [EMSGSIZE] The specified message length, `msg_len`, exceeds the message size attribute of
 43193 the message queue.
- 43194 [ETIMEDOUT] The `O_NONBLOCK` flag was not set when the message queue was opened,
 43195 but the timeout expired before the message could be added to the queue.

43196 **EXAMPLES**

43197 None.

43198 **APPLICATION USAGE**

43199 The value of the symbol `{MQ_PRIO_MAX}` limits the number of priority levels supported by the
 43200 application. Message priorities range from 0 to `{MQ_PRIO_MAX}-1`.

43201 **RATIONALE**

43202 None.

43203 **FUTURE DIRECTIONS**

43204 None.

43205 **SEE ALSO**

43206 [*mq_open\(\)*](#), [*mq_receive\(\)*](#), [*mq_setattr\(\)*](#), [*time*](#)

43207 XBD [*<mqqueue.h>*](#), [*<time.h>*](#)

43208 **CHANGE HISTORY**

43209 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43210 **Issue 6**

43211 The `mq_send()` function is marked as part of the Message Passing option.

43212 The `[ENOSYS]` error condition has been removed as stubs need not be provided if an
 43213 implementation does not support the Message Passing option.

43214 The `mq_timedsend()` function is added for alignment with IEEE Std 1003.1d-1999.

43215 **Issue 7**

43216 The `mq_timedsend()` function is moved from the Timeouts option to the Base.

43217 Functionality relating to the Timers option is moved to the Base.

43218 **NAME**43219 `mq_setattr` — set message queue attributes (**REALTIME**)43220 **SYNOPSIS**

```
43221 MSG #include <mqqueue.h>
43222 int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
43223               struct mq_attr *restrict omqstat);
```

43224 **DESCRIPTION**43225 The `mq_setattr()` function shall set attributes associated with the open message queue
43226 description referenced by the message queue descriptor specified by `mqdes`.43227 The message queue attributes corresponding to the following members defined in the **mq_attr**
43228 structure shall be set to the specified values upon successful completion of `mq_setattr()`:43229 `mq_flags` The value of this member is the bitwise-logical OR of zero or more of
43230 `O_NONBLOCK` and any implementation-defined flags.43231 The values of the `mq_maxmsg`, `mq_msgsize`, and `mq_curmsgs` members of the **mq_attr** structure
43232 shall be ignored by `mq_setattr()`.43233 If `omqstat` is non-NULL, the `mq_setattr()` function shall store, in the location referenced by
43234 `omqstat`, the previous message queue attributes and the current queue status. These values shall
43235 be the same as would be returned by a call to `mq_getattr()` at that point.43236 **RETURN VALUE**43237 Upon successful completion, the function shall return a value of zero and the attributes of the
43238 message queue shall have been changed as specified.43239 Otherwise, the message queue attributes shall be unchanged, and the function shall return a
43240 value of `-1` and set `errno` to indicate the error.43241 **ERRORS**43242 The `mq_setattr()` function shall fail if:43243 [EBADF] The `mqdes` argument is not a valid message queue descriptor.43244 **EXAMPLES**

43245 None.

43246 **APPLICATION USAGE**

43247 None.

43248 **RATIONALE**

43249 None.

43250 **FUTURE DIRECTIONS**

43251 None.

43252 **SEE ALSO**43253 [*mq_open\(\)*](#), [*mq_send\(\)*](#), [*msgctl\(\)*](#), [*msgget\(\)*](#), [*msgrcv\(\)*](#), [*msgsnd\(\)*](#)43254 XBD [**<mqqueue.h>**](#)43255 **CHANGE HISTORY**

43256 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43257

Issue 6

43258

The *mq_setattr()* function is marked as part of the Message Passing option.

43259

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

43260

43261

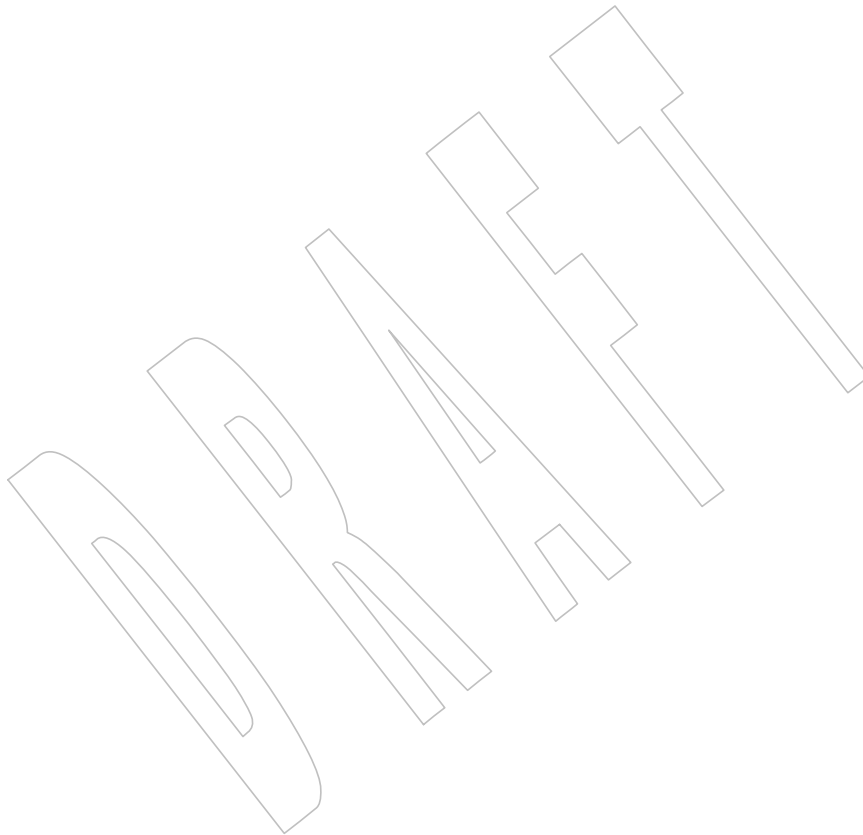
The *mq_timedsend()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

43262

43263

The **restrict** keyword is added to the *mq_setattr()* prototype for alignment with the ISO/IEC 9899:1999 standard.

43264



43265 **NAME**43266 mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)43267 **SYNOPSIS**

```
43268 MSG #include <mqueue.h>  
43269 #include <time.h>  
  
43270 ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,  
43271 size_t msg_len, unsigned *restrict msg_prio,  
43272 const struct timespec *restrict abstime);
```

43273 **DESCRIPTION**43274 Refer to *mq_receive()*.

43275 **NAME**43276 mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)43277 **SYNOPSIS**

```
43278 MSG #include <mqueue.h>  
43279 #include <time.h>  
  
43280 int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,  
43281 unsigned msg_prio, const struct timespec *abstime);
```

43282 **DESCRIPTION**43283 Refer to [mq_send\(\)](#).

43284 **NAME**
 43285 `mq_unlink` — remove a message queue (**REALTIME**)

43286 **SYNOPSIS**

43287 MSG `#include <mqeue.h>`
 43288 `int mq_unlink(const char *name);`

43289 **DESCRIPTION**

43290 The `mq_unlink()` function shall remove the message queue named by the pathname *name*. After
 43291 a successful call to `mq_unlink()` with *name*, a call to `mq_open()` with *name* shall fail if the flag
 43292 `O_CREAT` is not set in *flags*. If one or more processes have the message queue open when
 43293 `mq_unlink()` is called, destruction of the message queue shall be postponed until all references to
 43294 the message queue have been closed.

43295 Calls to `mq_open()` to recreate the message queue may fail until the message queue is actually
 43296 removed. However, the `mq_unlink()` call need not block until all references have been closed; it
 43297 may return immediately.

43298 **RETURN VALUE**

43299 Upon successful completion, the function shall return a value of zero. Otherwise, the named
 43300 message queue shall be unchanged by this function call, and the function shall return a value of
 43301 `-1` and set *errno* to indicate the error.

43302 **ERRORS**

43303 The `mq_unlink()` function shall fail if:

43304 `[EACCES]` Permission is denied to unlink the named message queue.

43305 `[ENOENT]` The named message queue does not exist.

43306 The `mq_unlink()` function may fail if:

43307 `[ENAMETOOLONG]`

43308 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
 43309 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 43310 systems, or has a pathname component that is longer than
 43311 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or
 43312 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to `mq_unlink()`
 43313 with a *name* argument that contains the same message queue name as was
 43314 previously used in a successful `mq_open()` call shall not give an
 43315 `[ENAMETOOLONG]` error.

43316 **EXAMPLES**

43317 None.

43318 **APPLICATION USAGE**

43319 None.

43320 **RATIONALE**

43321 None.

43322 **FUTURE DIRECTIONS**

43323 None.

43324

SEE ALSO

43325

mq_close(), *mq_open()*, *msgctl()*, *msgget()*, *msgrcv()*, *msgsnd()*

43326

XBD <mqqueue.h>

43327

CHANGE HISTORY

43328

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

43329

Issue 6

43330

The *mq_unlink()* function is marked as part of the Message Passing option.

43331

The Open Group Corrigendum U021/5 is applied, clarifying that upon unsuccessful completion, the named message queue is unchanged by this function.

43332

43333

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Message Passing option.

43334

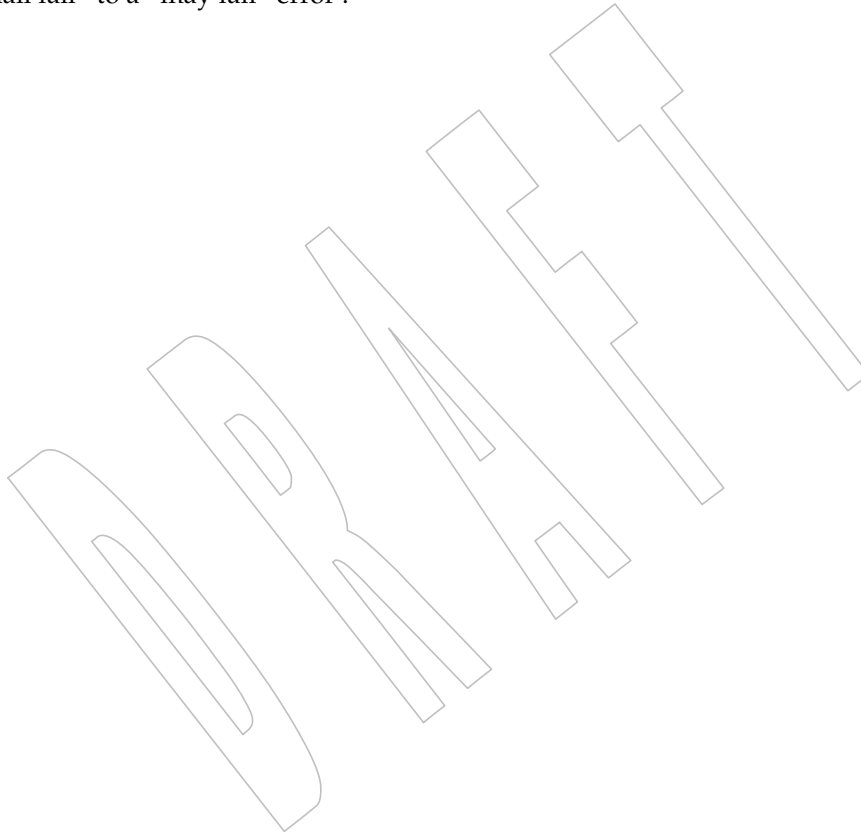
43335

Issue 7

43336

Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error .

43337



rand48()

43338 **NAME**
43339 `rand48` — generate uniformly distributed pseudo-random signed long integers

SYNOPSIS

43340 XSI `#include <stdlib.h>`
43341
43342 `long rand48(void);`

DESCRIPTION

43343 Refer to *drand48()*.
43344

43345 **NAME**
 43346 msgctl — XSI message control operations

43347 SYNOPSIS

```
43348 XSI #include <sys/msg.h>
43349 int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

43350 DESCRIPTION

43351 The *msgctl()* function operates on XSI message queues (see XBD [Section 3.223](#), on page 64). It is
 43352 unspecified whether this function interoperates with the realtime interprocess communication
 43353 facilities defined in [Section 2.8](#) (on page 476).

43354 The *msgctl()* function shall provide message control operations as specified by *cmd*. The
 43355 following values for *cmd*, and the message control operations they specify, are:

43356 **IPC_STAT** Place the current value of each member of the **msqid_ds** data structure
 43357 associated with *msqid* into the structure pointed to by *buf*. The contents of this
 43358 structure are defined in **<sys/msg.h>**.

43359 **IPC_SET** Set the value of the following members of the **msqid_ds** data structure
 43360 associated with *msqid* to the corresponding value found in the structure
 43361 pointed to by *buf*:

```
43362 msg_perm.uid
43363 msg_perm.gid
43364 msg_perm.mode
43365 msg_qbytes
```

43366 **IPC_SET** can only be executed by a process with appropriate privileges or that
 43367 has an effective user ID equal to the value of **msg_perm.cuid** or
 43368 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a
 43369 process with appropriate privileges can raise the value of **msg_qbytes**.

43370 **IPC_RMID** Remove the message queue identifier specified by *msqid* from the system and
 43371 destroy the message queue and **msqid_ds** data structure associated with it.
 43372 **IPC_RMID** can only be executed by a process with appropriate privileges or
 43373 one that has an effective user ID equal to the value of **msg_perm.cuid** or
 43374 **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

43375 RETURN VALUE

43376 Upon successful completion, *msgctl()* shall return 0; otherwise, it shall return -1 and set *errno* to
 43377 indicate the error.

43378 ERRORS

43379 The *msgctl()* function shall fail if:

43380 **[EACCES]** The argument *cmd* is **IPC_STAT** and the calling process does not have read
 43381 permission; see [Section 2.7](#) (on page 474).

43382 **[EINVAL]** The value of *msqid* is not a valid message queue identifier; or the value of *cmd*
 43383 is not a valid command.

43384 **[EPERM]** The argument *cmd* is **IPC_RMID** or **IPC_SET** and the effective user ID of the
 43385 calling process is not equal to that of a process with appropriate privileges and
 43386 it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data
 43387 structure associated with *msqid*.

43388 [EPERM] The argument *cmd* is IPC_SET, an attempt is being made to increase to the
 43389 value of **msg_qbytes**, and the effective user ID of the calling process does not
 43390 have appropriate privileges.

43391 **EXAMPLES**

43392 None.

43393 **APPLICATION USAGE**

43394 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
 43395 (IPC). Application developers who need to use IPC should design their applications so that
 43396 modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to
 43397 use the alternative interfaces.

43398 **RATIONALE**

43399 None.

43400 **FUTURE DIRECTIONS**

43401 None.

43402 **SEE ALSO**

43403 [Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#),
 43404 [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

43405 XBD [Section 3.223](#) (on page 64), [<sys/msg.h>](#)

43406 **CHANGE HISTORY**

43407 First released in Issue 2. Derived from Issue 2 of the SVID.

43408 **Issue 5**

43409 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
 43410 DIRECTIONS to a new APPLICATION USAGE section.

43411 **NAME**

43412 msgget — get the XSI message queue identifier

43413 **SYNOPSIS**

```
43414 XSI #include <sys/msg.h>
43415 int msgget(key_t key, int msgflg);
```

43416 **DESCRIPTION**

43417 The `msgget()` function operates on XSI message queues (see XBD Section 3.223, on page 64). It is
 43418 unspecified whether this function interoperates with the realtime interprocess communication
 43419 facilities defined in Section 2.8 (on page 476).

43420 The `msgget()` function shall return the message queue identifier associated with the argument
 43421 `key`.

43422 A message queue identifier, associated message queue, and data structure (see `<sys/msg.h>`),
 43423 shall be created for the argument `key` if one of the following is true:

- 43424 • The argument `key` is equal to `IPC_PRIVATE`.
- 43425 • The argument `key` does not already have a message queue identifier associated with it, and
 43426 (`msgflg & IPC_CREAT`) is non-zero.

43427 Upon creation, the data structure associated with the new message queue identifier shall be
 43428 initialized as follows:

- 43429 • `msg_perm.cuid`, `msg_perm.uid`, `msg_perm.cgid`, and `msg_perm.gid` shall be set equal to
 43430 the effective user ID and effective group ID, respectively, of the calling process.
- 43431 • The low-order 9 bits of `msg_perm.mode` shall be set equal to the low-order 9 bits of `msgflg`.
- 43432 • `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` shall be set equal to 0.
- 43433 • `msg_ctime` shall be set equal to the current time.
- 43434 • `msg_qbytes` shall be set equal to the system limit.

43435 **RETURN VALUE**

43436 Upon successful completion, `msgget()` shall return a non-negative integer, namely a message
 43437 queue identifier. Otherwise, it shall return `-1` and set `errno` to indicate the error.

43438 **ERRORS**

43439 The `msgget()` function shall fail if:

- | | | |
|-------|----------|---|
| 43440 | [EACCES] | A message queue identifier exists for the argument <code>key</code> , but operation permission as specified by the low-order 9 bits of <code>msgflg</code> would not be granted; see Section 2.7 (on page 474). |
| 43443 | [EEXIST] | A message queue identifier exists for the argument <code>key</code> but $((msgflg \& IPC_CREAT) \&\& (msgflg \& IPC_EXCL))$ is non-zero. |
| 43445 | [ENOENT] | A message queue identifier does not exist for the argument <code>key</code> and $(msgflg \& IPC_CREAT)$ is 0. |
| 43447 | [ENOSPC] | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded. |

msgget()

43450

EXAMPLES

43451

None.

43452

APPLICATION USAGE

43453

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

43454

43455

43456

43457

RATIONALE

43458

None.

43459

FUTURE DIRECTIONS

43460

None.

43461

SEE ALSO

43462

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#), [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgrcv\(\)](#), [msgsnd\(\)](#)

43463

43464

XBD [Section 3.223](#) (on page 64), [<sys/msg.h>](#)

43465

CHANGE HISTORY

43466

First released in Issue 2. Derived from Issue 2 of the SVID.

43467

Issue 5

43468

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

43469

NAME

msgrcv — XSI message receive operation

SYNOPSIS

```
XSI #include <sys/msg.h>
    ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
        int msgflg);
```

DESCRIPTION

The *msgrcv()* function operates on XSI message queues (see XBD Section 3.223, on page 64). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in Section 2.8 (on page 476).

The *msgrcv()* function shall read a message from the queue associated with the message queue identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long    mtype;    /* Message type. */
    char    mtext[1]; /* Message text. */
}
```

The structure member *mtype* is the received message's type as specified by the sending process.

The structure member *mtext* is the text of the message.

The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The truncated part of the message shall be lost and no indication of the truncation shall be given to the calling process.

If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

The argument *msgtyp* specifies the type of message requested as follows:

- If *msgtyp* is 0, the first message on the queue shall be received.
- If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
- If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* shall be received.

The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

- If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a return value of -1 and *errno* set to [ENOMSG].
- If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:

— A message of the desired type is placed on the queue.

- 43509 — The message queue identifier *msqid* is removed from the system; when this occurs,
43510 *errno* shall be set equal to [EIDRM] and -1 shall be returned.
- 43511 — The calling thread receives a signal that is to be caught; in this case a message is not
43512 received and the calling thread resumes execution in the manner prescribed in
43513 *sigaction()*.

43514 Upon successful completion, the following actions are taken with respect to the data structure
43515 associated with *msqid*:

- 43516 • **msg_qnum** shall be decremented by 1.
- 43517 • **msg_lrpid** shall be set equal to the process ID of the calling process.
- 43518 • **msg_rtime** shall be set equal to the current time.

43519 RETURN VALUE

43520 Upon successful completion, *msgrcv()* shall return a value equal to the number of bytes actually
43521 placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv()* shall return
43522 (**ssize_t**)-1, and *errno* shall be set to indicate the error.

43523 ERRORS

43524 The *msgrcv()* function shall fail if:

- | | | |
|-------|----------|--|
| 43525 | [E2BIG] | The value of <i>mtext</i> is greater than <i>msgsz</i> and (<i>msgflg</i> & MSG_NOERROR) is 0. |
| 43526 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page
43527 474). |
| 43528 | [EIDRM] | The message queue identifier <i>msqid</i> is removed from the system. |
| 43529 | [EINTR] | The <i>msgrcv()</i> function was interrupted by a signal. |
| 43530 | [EINVAL] | <i>msqid</i> is not a valid message queue identifier. |
| 43531 | [ENOMSG] | The queue does not contain a message of the desired type and (<i>msgflg</i> &
43532 IPC_NOWAIT) is non-zero. |

43533 EXAMPLES

43534 Receiving a Message

43535 The following example receives the first message on the queue (based on the value of the *msgtyp*
43536 argument, 0). The queue is identified by the *msqid* argument (assuming that the value has
43537 previously been set). This call specifies that an error should be reported if no message is
43538 available, but not if the message is too large. The message size is calculated directly using the
43539 *sizeof* operator.

```
43540 #include <sys/msg.h>
43541 ...
43542 int result;
43543 int msqid;
43544 struct message {
43545     long type;
43546     char text[20];
43547 } msg;
43548 long msgtyp = 0;
43549 ...
43550 result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
43551               msgtyp, MSG_NOERROR | IPC_NOWAIT);
```


43552
43553
43554
43555
43556

43557
43558

43559
43560

43561
43562
43563
43564

43565

43566
43567

43568
43569
43570

43571
43572

43573
43574

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#), [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgsnd\(\)](#), [sigaction\(\)](#)

XBD [Section 3.223](#) (on page 64), [<sys/msg.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The type of the return value is changed from **int** to **ssize_t**, and a warning is added to the DESCRIPTION about values of *msgsz* larger than {SSIZE_MAX}.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

43575 **NAME**
 43576 msgsnd — XSI message send operation

43577 **SYNOPSIS**

```
43578 XSI #include <sys/msg.h>
43579 int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

43580 **DESCRIPTION**

43581 The *msgsnd()* function operates on XSI message queues (see XBD Section 3.223, on page 64). It is
 43582 unspecified whether this function interoperates with the realtime interprocess communication
 43583 facilities defined in Section 2.8 (on page 476).

43584 The *msgsnd()* function shall send a message to the queue associated with the message queue
 43585 identifier specified by *msqid*.

43586 The application shall ensure that the argument *msgp* points to a user-defined buffer that contains
 43587 first a field of type **long** specifying the type of the message, and then a data portion that holds
 43588 the data bytes of the message. The structure below is an example of what this user-defined
 43589 buffer might look like:

```
43590 struct mymsg {
43591     long   mtype;           /* Message type. */
43592     char   mtext[1];       /* Message text. */
43593 }
```

43594 The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving
 43595 process for message selection.

43596 The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range
 43597 from 0 to a system-imposed maximum.

43598 The argument *msgflg* specifies the action to be taken if one or more of the following is true:

- 43599 • The number of bytes already on the queue is equal to **msg_qbytes**; see `<sys/msg.h>`.
- 43600 • The total number of messages on all queues system-wide is equal to the system-imposed
 43601 limit.

43602 These actions are as follows:

- 43603 • If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling
 43604 thread shall return immediately.
- 43605 • If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the
 43606 following occurs:
 - 43607 — The condition responsible for the suspension no longer exists, in which case the
 43608 message is sent.
 - 43609 — The message queue identifier *msqid* is removed from the system; when this occurs,
 43610 *errno* shall be set equal to [EIDRM] and `-1` shall be returned.
 - 43611 — The calling thread receives a signal that is to be caught; in this case the message is not
 43612 sent and the calling thread resumes execution in the manner prescribed in *sigaction()*.

43613 Upon successful completion, the following actions are taken with respect to the data structure
43614 associated with *msqid*; see `<sys/msg.h>`:

- 43615 • `msg_qnum` shall be incremented by 1.
- 43616 • `msg_lspid` shall be set equal to the process ID of the calling process.
- 43617 • `msg_stime` shall be set equal to the current time.

43618 RETURN VALUE

43619 Upon successful completion, *msgsnd()* shall return 0; otherwise, no message shall be sent,
43620 *msgsnd()* shall return `-1`, and *errno* shall be set to indicate the error.

43621 ERRORS

43622 The *msgsnd()* function shall fail if:

- | | | |
|-------|----------|---|
| 43623 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 474). |
| 43625 | [EAGAIN] | The message cannot be sent for one of the reasons cited above and (<i>msgflg</i> & <code>IPC_NOWAIT</code>) is non-zero. |
| 43627 | [EIDRM] | The message queue identifier <i>msqid</i> is removed from the system. |
| 43628 | [EINTR] | The <i>msgsnd()</i> function was interrupted by a signal. |
| 43629 | [EINVAL] | The value of <i>msqid</i> is not a valid message queue identifier, or the value of <i>mtype</i> is less than 1; or the value of <i>msgsz</i> is less than 0 or greater than the system-imposed limit. |

43632 EXAMPLES

43633 Sending a Message

43634 The following example sends a message to the queue identified by the *msqid* argument
43635 (assuming that value has previously been set). This call specifies that an error should be
43636 reported if no message is available. The message size is calculated directly using the *sizeof*
43637 operator.

```
43638 #include <sys/msg.h>
43639 ...
43640 int result;
43641 int msqid;
43642 struct message {
43643     long type;
43644     char text[20];
43645 } msg;
43646
43647 msg.type = 1;
43648 strcpy(msg.text, "This is message 1");
43649 ...
43650 result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

43650 APPLICATION USAGE

43651 The POSIX Realtime Extension defines alternative interfaces for interprocess communication
43652 (IPC). Application developers who need to use IPC should design their applications so that
43653 modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to
43654 use the alternative interfaces.

43655

RATIONALE

43656

None.

43657

FUTURE DIRECTIONS

43658

None.

43659

SEE ALSO

43660

Section 2.7 (on page 474), Section 2.8 (on page 476), [mq_close\(\)](#), [mq_getattr\(\)](#), [mq_notify\(\)](#), [mq_open\(\)](#), [mq_receive\(\)](#), [mq_send\(\)](#), [mq_setattr\(\)](#), [mq_unlink\(\)](#), [msgctl\(\)](#), [msgget\(\)](#), [msgrcv\(\)](#), [sigaction\(\)](#)

43661

43662

43663

XBD Section 3.223 (on page 64), [<sys/msg.h>](#)

43664

CHANGE HISTORY

43665

First released in Issue 2. Derived from Issue 2 of the SVID.

43666

Issue 5

43667

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

43668

43669

Issue 6

43670

The normative text is updated to avoid use of the term “must” for application requirements.

43671 **NAME**
 43672 `msync` — synchronize memory with physical storage

43673 **SYNOPSIS**

```
43674 XSI|SIO #include <sys/mman.h>
43675 int msync(void *addr, size_t len, int flags);
```

43676 **DESCRIPTION**

43677 The `msync()` function shall write all modified data to permanent storage locations, if any, in
 43678 those whole pages containing any part of the address space of the process starting at address
 43679 `addr` and continuing for `len` bytes. If no such storage exists, `msync()` need not have any effect. If
 43680 requested, the `msync()` function shall then invalidate cached copies of data.

43681 The implementation may require that `addr` be a multiple of the page size as returned by
 43682 `sysconf()`.

43683 For mappings to files, the `msync()` function shall ensure that all write operations are completed
 43684 as defined for synchronized I/O data integrity completion. It is unspecified whether the
 43685 implementation also writes out other file attributes. When the `msync()` function is called on
 43686 MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and
 43687 shall not cause such data to be made visible to other processes. It is unspecified whether data in
 43688 SHM|TYM MAP_PRIVATE mappings has any permanent storage locations. The effect of `msync()` on a
 43689 shared memory object or a typed memory object is unspecified. The behavior of this function is
 43690 unspecified if the mapping was not established by a call to `mmap()`.

43691 The `flags` argument is constructed from the bitwise-inclusive OR of one or more of the following
 43692 flags defined in the `<sys/mman.h>` header:

Symbolic Constant	Description
MS_ASYNC	Perform asynchronous writes.
MS_SYNC	Perform synchronous writes.
MS_INVALIDATE	Invalidate cached data.

43697 When MS_ASYNC is specified, `msync()` shall return immediately once all the write operations
 43698 are initiated or queued for servicing; when MS_SYNC is specified, `msync()` shall not return until
 43699 all write operations are completed as defined for synchronized I/O data integrity completion.
 43700 Either MS_ASYNC or MS_SYNC shall be specified, but not both.

43701 When MS_INVALIDATE is specified, `msync()` shall invalidate all cached copies of mapped data
 43702 that are inconsistent with the permanent storage locations such that subsequent references shall
 43703 obtain data that was consistent with the permanent storage locations sometime between the call
 43704 to `msync()` and the first subsequent memory reference to the data.

43705 If `msync()` causes any write to a file, the file's last data modification and last file status change
 43706 timestamps shall be marked for update.

43707 **RETURN VALUE**

43708 Upon successful completion, `msync()` shall return 0; otherwise, it shall return -1 and set `errno` to
 43709 indicate the error.

43710 **ERRORS**

43711 The `msync()` function shall fail if:

43712 [EBUSY] Some or all of the addresses in the range starting at `addr` and continuing for `len`
 43713 bytes are locked, and MS_INVALIDATE is specified.

- 43714 [EINVAL] The value of *flags* is invalid.
- 43715 [ENOMEM] The addresses in the range starting at *addr* and continuing for *len* bytes are
- 43716 outside the range allowed for the address space of a process or specify one or
- 43717 more pages that are not mapped.

43718 The *msync()* function may fail if:

- 43719 [EINVAL] The value of *addr* is not a multiple of the page size as returned by *sysconf()*.

43720 EXAMPLES

43721 None.

43722 APPLICATION USAGE

43723 The *msync()* function is only supported if the Synchronized Input and Output option is

43724 supported, and thus need not be available on all implementations.

43725 The *msync()* function should be used by programs that require a memory object to be in a

43726 known state; for example, in building transaction facilities.

43727 Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees

43728 that *msync()* is the only control over when pages are or are not written to disk.

43729 RATIONALE

43730 The *msync()* function writes out data in a mapped region to the permanent storage for the

43731 underlying object. The call to *msync()* ensures data integrity of the file.

43732 After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag

43733 was specified. This is useful on systems that do not support read/write consistency.

43734 FUTURE DIRECTIONS

43735 None.

43736 SEE ALSO

43737 [mmap\(\)](#), [sysconf\(\)](#)

43738 XBD [<sys/mman.h>](#)

43739 CHANGE HISTORY

43740 First released in Issue 4, Version 2.

43741 Issue 5

43742 Moved from X/OPEN UNIX extension to BASE.

43743 Aligned with *msync()* in the POSIX Realtime Extension as follows:

- 43744
- The DESCRIPTION is extensively reworded.
 - [EBUSY] and a new form of [EINVAL] are added as mandatory error conditions.
- 43745

43746 Issue 6

43747 The *msync()* function is marked as part of the Memory Mapped Files and Synchronized Input

43748 and Output options.

43749 The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- 43750
- The [EBUSY] mandatory error condition is added.

43751 The following new requirements on POSIX implementations derive from alignment with the

43752 Single UNIX Specification:

- 43753
- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple
- 43754 of the page size.

43755 • The second [EINVAL] error condition is made mandatory.

43756 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding reference to
43757 typed memory objects.

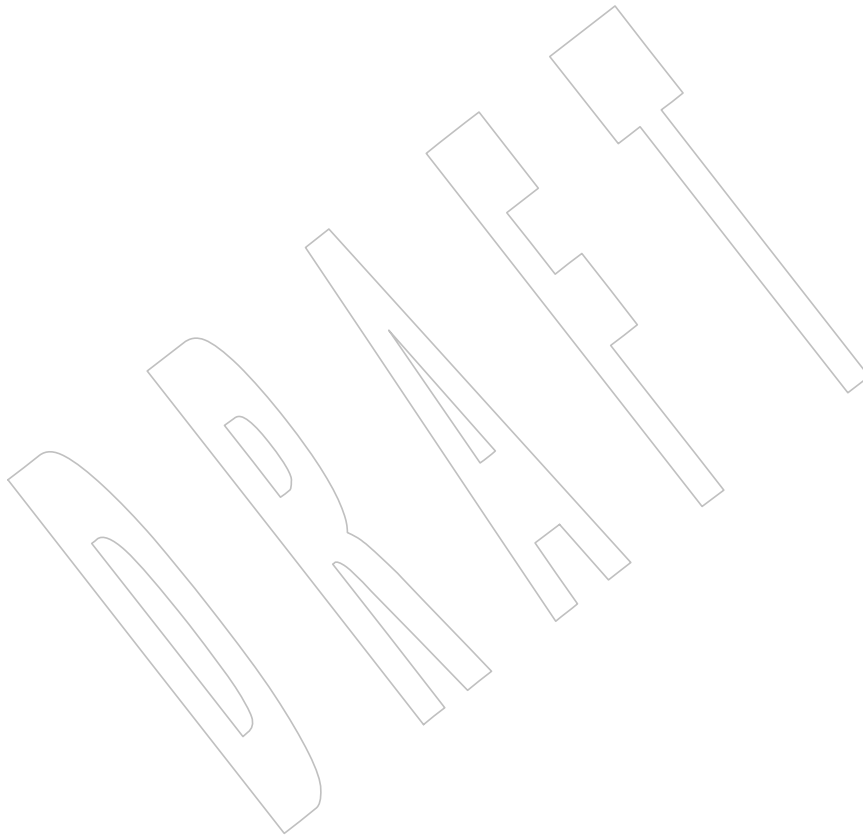
43758 **Issue 7**

43759 Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment -
43760 requirements.

43761 SD5-XSH-ERN-110 is applied. +

43762 The *msync()* function is marked as part of the Synchronized Input and Output option or XSI |
43763 option as the Memory Mapped Files is moved to the Base. |

43764 Changes are made related to support for finegrained timestamps.



munlock()

43765 **NAME**
43766 `munlock` — unlock a range of process address space

43767 **SYNOPSIS**

43768 MLR `#include <sys/mman.h>`
43769 `int munlock(const void *addr, size_t len);`

43770 **DESCRIPTION**

43771 Refer to *mlock()*.

43772 **NAME**
43773 `munlockall` — unlock the address space of a process

43774 **SYNOPSIS**

43775 ML `#include <sys/mman.h>`
43776 `int munlockall(void);`

43777 **DESCRIPTION**

43778 Refer to *mlockall()*.

43779 **NAME**43780 `munmap` — unmap pages of memory43781 **SYNOPSIS**43782 `#include <sys/mman.h>`43783 `int munmap(void *addr, size_t len);`43784 **DESCRIPTION**

43785 The `munmap()` function shall remove any mappings for those entire pages containing any part of
 43786 the address space of the process starting at `addr` and continuing for `len` bytes. Further references
 43787 to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no
 43788 mappings in the specified address range, then `munmap()` has no effect.

43789 The implementation may require that `addr` be a multiple of the page size as returned by
 43790 `sysconf()`.

43791 If a mapping to be removed was private, any modifications made in this address range shall be
 43792 discarded.

43793 MLIMLR Any memory locks (see `mlock()` and `mlockall()`) associated with this address range shall be
 43794 removed, as if by an appropriate call to `munlock()`.

43795 TYM If a mapping removed from a typed memory object causes the corresponding address range of
 43796 the memory pool to be inaccessible by any process in the system except through allocatable
 43797 mappings (that is, mappings of typed memory objects opened with the
 43798 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall
 43799 become deallocated and may become available to satisfy future typed memory allocation
 43800 requests.

43801 A mapping removed from a typed memory object opened with the
 43802 POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of
 43803 that typed memory for allocation.

43804 The behavior of this function is unspecified if the mapping was not established by a call to
 43805 `mmap()`.

43806 **RETURN VALUE**

43807 Upon successful completion, `munmap()` shall return 0; otherwise, it shall return -1 and set `errno`
 43808 to indicate the error.

43809 **ERRORS**

43810 The `munmap()` function shall fail if:

43811 [EINVAL] Addresses in the range [`addr,addr+len`) are outside the valid range for the
 43812 address space of a process.

43813 [EINVAL] The `len` argument is 0.

43814 The `munmap()` function may fail if:

43815 [EINVAL] The `addr` argument is not a multiple of the page size as returned by `sysconf()`.

43816
43817
43818
43819
43820
43821
43822
43823
43824
43825
43826
43827
43828
43829
43830
43831
43832
43833
43834
43835
43836
43837
43838
43839
43840
43841
43842
43843
43844
43845
43846
43847
43848
43849
43850
43851
43852
43853
43854

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *munmap()* function corresponds to SVR4, just as the *mmap()* function does.

It is possible that an application has applied process memory locking to a region that contains shared memory. If this has occurred, the *munmap()* call ignores those locks and, if necessary, causes those locks to be removed.

Most implementations require that *addr* is a multiple of the page size as returned by *sysconf()*.

FUTURE DIRECTIONS

None.

SEE ALSO

mlock(), *mlockall()*, *mmap()*, *posix_typed_mem_open()*, *sysconf()*

XBD <signal.h>, <sys/mman.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Aligned with *munmap()* in the POSIX Realtime Extension as follows:

- The DESCRIPTION is extensively reworded.
- The SIGBUS error is no longer permitted to be generated.

Issue 6

The *munmap()* function is marked as part of the Memory Mapped Files and Shared Memory Objects option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to state that implementations require *addr* to be a multiple of the page size.
- The [EINVAL] error conditions are added.

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- Semantics for typed memory objects are added to the DESCRIPTION.
- The *posix_typed_mem_open()* function is added to the SEE ALSO section.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/36 is applied, changing the margin code in the SYNOPSIS from MF|SHM to MC3 (notation for MF|SHM|TYM).

Issue 7

Austin Group Interpretation 1003.1-2001 #078 is applied, clarifying page alignment requirements.

The *munmap()* function is moved from the Memory Mapped Files option to the Base.

43855 **NAME**

43856 nan, nanf, nanl — return quiet NaN

43857 **SYNOPSIS**

```
43858 #include <math.h>
43859 double nan(const char *tagp);
43860 float nanf(const char *tagp);
43861 long double nanl(const char *tagp);
```

43862 **DESCRIPTION**

43863 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43864 conflict between the requirements described here and the ISO C standard is unintentional. This
 43865 volume of POSIX.1-200x defers to the ISO C standard.

43866 The function call *nan("n-char-sequence")* shall be equivalent to:

```
43867 strtod("NAN(n-char-sequence)", (char **) NULL);
```

43868 The function call *nan("")* shall be equivalent to:

```
43869 strtod("NAN()", (char **) NULL)
```

43870 If *tagp* does not point to an *n-char* sequence or an empty string, the function call shall be
 43871 equivalent to:

```
43872 strtod("NAN", (char **) NULL)
```

43873 Function calls to *nanf()* and *nanl()* are equivalent to the corresponding function calls to *strtof()*
 43874 and *strtold()*.

43875 **RETURN VALUE**

43876 These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

43877 If the implementation does not support quiet NaNs, these functions shall return zero.

43878 **ERRORS**

43879 No errors are defined.

43880 **EXAMPLES**

43881 None.

43882 **APPLICATION USAGE**

43883 None.

43884 **RATIONALE**

43885 None.

43886 **FUTURE DIRECTIONS**

43887 None.

43888 **SEE ALSO**

43889 [strtod\(\)](#)

43890 XBD [<math.h>](#)

43891 **CHANGE HISTORY**

43892 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.

43893 **NAME**

43894 nanosleep — high resolution sleep

43895 **SYNOPSIS**

```
43896 CX #include <time.h>
43897 int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
```

43898 **DESCRIPTION**

43899 The *nanosleep()* function shall cause the current thread to be suspended from execution until
 43900 either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the
 43901 calling thread, and its action is to invoke a signal-catching function or to terminate the process.
 43902 The suspension time may be longer than requested because the argument value is rounded up to
 43903 an integer multiple of the sleep resolution or because of the scheduling of other activity by the
 43904 system. But, except for the case of being interrupted by a signal, the suspension time shall not be
 43905 less than the time specified by *rqtp*, as measured by the system clock `CLOCK_REALTIME`.

43906 The use of the *nanosleep()* function has no effect on the action or blockage of any signal.

43907 **RETURN VALUE**

43908 If the *nanosleep()* function returns because the requested time has elapsed, its return value shall
 43909 be zero.

43910 If the *nanosleep()* function returns because it has been interrupted by a signal, it shall return a
 43911 value of `-1` and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the
 43912 **timespec** structure referenced by it is updated to contain the amount of time remaining in the
 43913 interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments may
 43914 point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

43915 If *nanosleep()* fails, it shall return a value of `-1` and set *errno* to indicate the error.

43916 **ERRORS**

43917 The *nanosleep()* function shall fail if:

- | | | |
|-------|----------|--|
| 43918 | [EINTR] | The <i>nanosleep()</i> function was interrupted by a signal. |
| 43919 | [EINVAL] | The <i>rqtp</i> argument specified a nanosecond value less than zero or greater than
43920 or equal to 1 000 million. |

43921 **EXAMPLES**

43922 None.

43923 **APPLICATION USAGE**

43924 None.

43925 **RATIONALE**

43926 It is common to suspend execution of a thread for an interval in order to poll the status of a non-
 43927 interrupting function. A large number of actual needs can be met with a simple extension to
 43928 *sleep()* that provides finer resolution.

43929 In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the
 43930 frequency of wakeup is limited by the resolution of the *alarm()* and *sleep()* functions. In 4.3 BSD,
 43931 it is possible to write such a routine using no static storage and reserving no system facilities.
 43932 Although it is possible to write a function with similar functionality to *sleep()* using the
 43933 remainder of the *timer_**() functions, such a function requires the use of signals and the
 43934 reservation of some signal number. This volume of POSIX.1-200x requires that *nanosleep()* be
 43935 non-intrusive of the signals function.

43936 The *nanosleep()* function shall return a value of 0 on success and -1 on failure or if interrupted.
43937 This latter case is different from *sleep()*. This was done because the remaining time is returned
43938 via an argument structure pointer, *rmtpt*, instead of as the return value.

FUTURE DIRECTIONS

43939 None.
43940

SEE ALSO

43941 *clock_nanosleep()*, *sleep*
43942

43943 XBD <[time.h](#)>

CHANGE HISTORY

43944 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
43945

Issue 6

43946 The *nanosleep()* function is marked as part of the Timers option.
43947

43948 The [ENOSYS] error condition has been removed as stubs need not be provided if an
43949 implementation does not support the Timers option.

43950 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/37 is applied, updating the SEE ALSO
43951 section to include the *clock_nanosleep()* function.

43952 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/63 is applied, correcting text in the
43953 RATIONALE section.

Issue 7

43954 SD5-XBD-ERN-33 is applied.
43955

43956 The *nanosleep()* function is moved from the Timers option to the Base.

43957 **NAME**
 43958 nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

43959 **SYNOPSIS**
 43960 #include <math.h>
 43961 double nearbyint(double x);
 43962 float nearbyintf(float x);
 43963 long double nearbyintl(long double x);

43964 **DESCRIPTION**
 43965 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 43966 conflict between the requirements described here and the ISO C standard is unintentional. This
 43967 volume of POSIX.1-200x defers to the ISO C standard.

43968 These functions shall round their argument to an integer value in floating-point format, using
 43969 the current rounding direction and without raising the inexact floating-point exception.

43970 An application wishing to check for error situations should set *errno* to zero and call
 43971 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 43972 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 43973 zero, an error has occurred.

43974 **RETURN VALUE**
 43975 Upon successful completion, these functions shall return the rounded integer value.

43976 MX If *x* is NaN, a NaN shall be returned.

43977 If *x* is ± 0 , ± 0 shall be returned.

43978 If *x* is $\pm\text{Inf}$, *x* shall be returned.

43979 XSI If the correct value would cause overflow, a range error shall occur and *nearbyint*(), *nearbyintf*(),
 43980 and *nearbyintl*() shall return the value of the macro $\pm\text{HUGE_VAL}$, $\pm\text{HUGE_VALF}$, and
 43981 $\pm\text{HUGE_VALL}$ (with the same sign as *x*), respectively.

43982 **ERRORS**
 43983 These functions shall fail if:

43984 XSI **Range Error** The result would cause an overflow.

43985 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 43986 then *errno* shall be set to [ERANGE]. If the integer expression
 43987 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 43988 floating-point exception shall be raised.

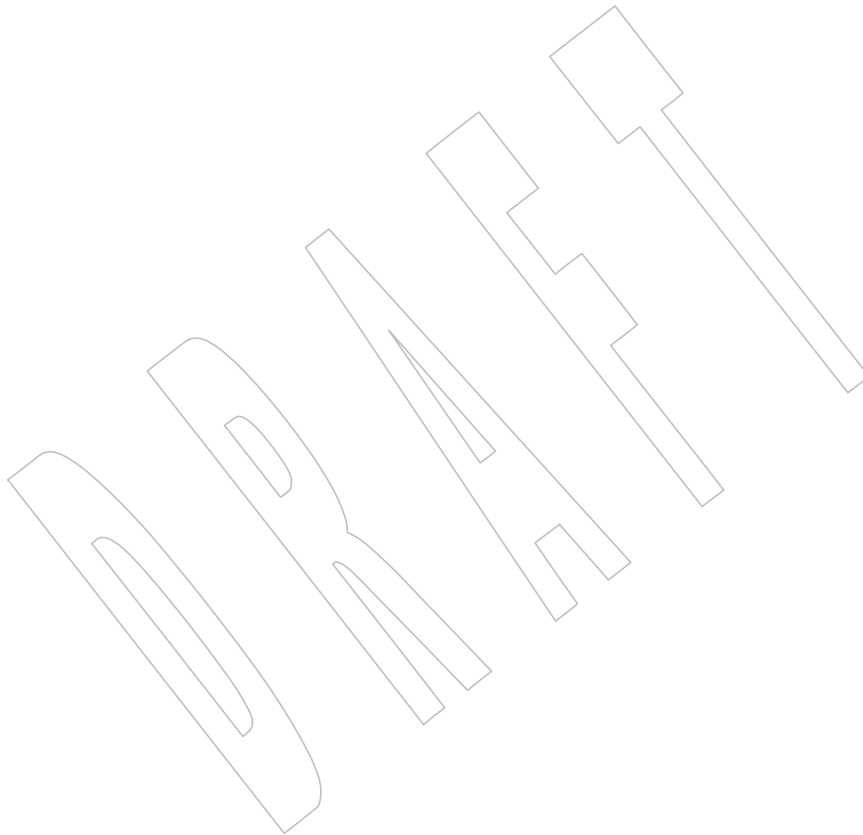
43989 **EXAMPLES**
 43990 None.

43991 **APPLICATION USAGE**
 43992 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 43993 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

43994 **RATIONALE**
 43995 None.

nearbyint()

- 43996 **FUTURE DIRECTIONS**
- 43997 None.
- 43998 **SEE ALSO**
- 43999 [fclearexcept\(\)](#), [fetestexcept\(\)](#)
- 44000 XBD [Section 4.19](#) (on page 104), [<math.h>](#)
- 44001 **CHANGE HISTORY**
- 44002 First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.



44003 **NAME**

44004 newlocale — create or modify a locale object

44005 **SYNOPSIS**

```
44006 CX #include <locale.h>
44007 locale_t newlocale(int category_mask, const char *locale,
44008 locale_t base);
```

44009 **DESCRIPTION**

44010 The *newlocale()* function shall create a new locale object or modify an existing one. If the *base*
 44011 argument is **(locale_t)0**, a new locale object shall be created. It is unspecified whether the locale
 44012 object pointed to by *base* shall be modified or freed and a new locale object created.

44013 The *category_mask* argument specifies the locale categories to be set or modified. Values for
 44014 *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants
 44015 *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*,
 44016 *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the other implementation-
 44017 defined *LC_*_MASK* values defined in **<locale.h>**.

44018 For each category with the corresponding bit set in *category_mask* the data from the locale named
 44019 by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale
 44020 named by *locale* shall replace the existing data within the locale object. If a completely new locale
 44021 object is created, the data for all sections not requested by *category_mask* shall be taken from the
 44022 default locale.

44023 The following preset values of *locale* are defined for all settings of *category_mask*:

44024	"POSIX"	Specifies the minimal environment for C-language translation called the 44025 POSIX locale.
44026	"C"	Equivalent to "POSIX".
44027	""	Specifies an implementation-defined native environment. This corresponds to 44028 the value of the associated environment variables, <i>LC_*</i> and <i>LANG</i> ; see XBD 44029 Chapter 7 (on page 121) and Chapter 8 (on page 159).

44030 If the *base* argument is not **(locale_t)0** and the *newlocale()* function call succeeds, the contents of
 44031 *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before
 44032 calling *newlocale()*. If the function call fails and the *base* argument is not **(locale_t)0**, the contents
 44033 of *base* shall remain valid and unchanged.

44034 The results are undefined if the *base* argument is the special locale object *LC_GLOBAL_LOCALE*.

44035 **RETURN VALUE**

44036 Upon successful completion, the *newlocale()* function shall return a handle which the caller may
 44037 use on subsequent calls to *duplocale()*, *freelocale()*, and other functions taking a **locale_t**
 44038 argument.

44039 Upon failure, the *newlocale()* function shall return **(locale_t)0** and set *errno* to indicate the error.

44040 **ERRORS**

44041 The *newlocale()* function shall fail if:

44042	[ENOMEM]	There is not enough memory available to create the locale object or load the 44043 locale data.
-------	----------	--

- 44044 [EINVAL] The *category_mask* contains a bit that does not correspond to a valid category.
- 44045 [ENOENT] For any of the categories in *category_mask*, the locale data is not available.
- 44046 The *newlocale()* function may fail if:
- 44047 [EINVAL] The *locale* argument is not a valid string pointer.

EXAMPLES

Constructing a Locale Object from Different Locales

The following example shows the construction of a locale where the *LC_CTYPE* category data comes from a locale *loc1* and the *LC_TIME* category data from a locale *tok2*:

```
44052 #include <locale.h>
44053 ...
44054 locale_t loc, new_loc;
44055 /* Get the "loc1" data. */
44056 loc = newlocale (LC_CTYPE_MASK, "loc1", NULL);
44057 if (loc == (locale_t) 0)
44058     abort ();
44059 /* Get the "loc2" data. */
44060 new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
44061 if (new_loc != (locale_t) 0)
44062     /* We don't abort if this fails. In this case this
44063     simply used to unchanged locale object. */
44064     loc = new_loc;
44065 ...
```

Freeing up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale()*:

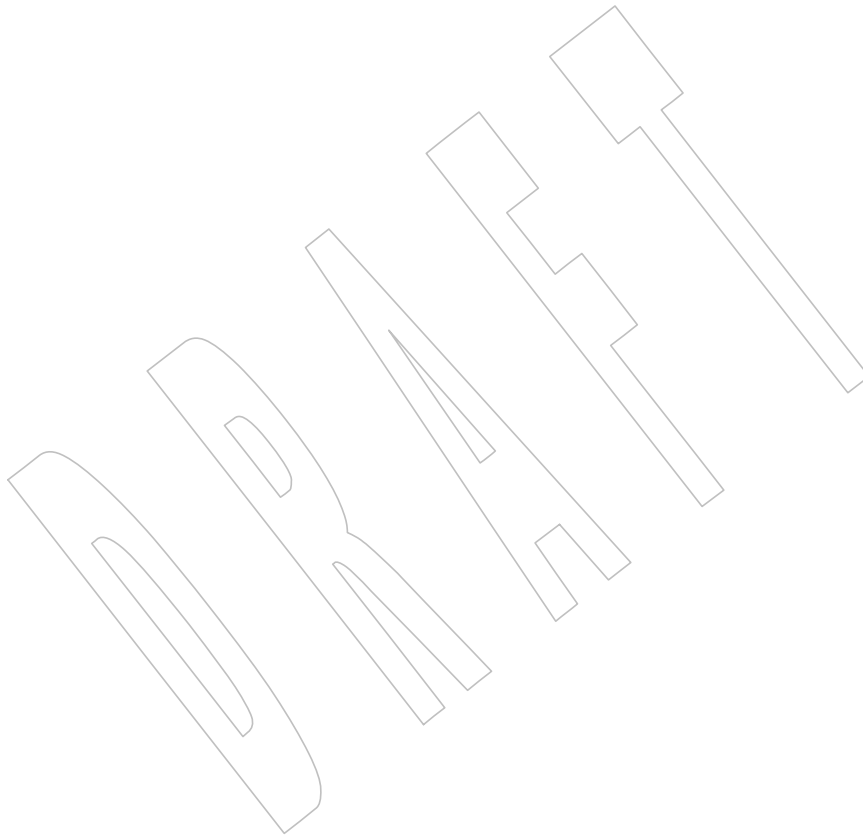
```
44068 #include <locale.h>
44069 ...
44070 /* Every locale object allocated with newlocale() should be
44071 * freed using freelocale():
44072 */
44073 locale_t loc;
44074 /* Get the locale. */
44075 loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);
44076 /* ... Use the locale object ... */
44077 ...
44078 /* Free the locale object resources. */
44079 freelocale (loc);
```

APPLICATION USAGE

Handles for locale objects created by the *newlocale()* function should be released by a corresponding call to *freelocale()*.

The special locale object *LC_GLOBAL_LOCALE* must not be passed for the *base* argument, even when returned by the *uselocale()* function.

44085	RATIONALE
44086	None.
44087	FUTURE DIRECTIONS
44088	None.
44089	SEE ALSO
44090	<i>duplocale()</i> , <i>freelocale()</i> , <i>uselocale()</i>
44091	XBD Chapter 7 (on page 121), Chapter 8 (on page 159), <locale.h>
44092	CHANGE HISTORY
44093	First released in Issue 7.



44094 NAME

44095 nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable
44096 floating-point number

44097 SYNOPSIS

```
44098 #include <math.h>

44099 double nextafter(double x, double y);
44100 float nextafterf(float x, float y);
44101 long double nextafterl(long double x, long double y);
44102 double nexttoward(double x, long double y);
44103 float nexttowardf(float x, long double y);
44104 long double nexttowardl(long double x, long double y);
```

44105 DESCRIPTION

44106 CX The functionality described on this reference page is aligned with the ISO C standard. Any
44107 conflict between the requirements described here and the ISO C standard is unintentional. This
44108 volume of POSIX.1-200x defers to the ISO C standard.

44109 The *nextafter()*, *nextafterf()*, and *nextafterl()* functions shall compute the next representable
44110 floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter()* shall
44111 return the largest representable floating-point number less than *x*. The *nextafter()*, *nextafterf()*,
44112 and *nextafterl()* functions shall return *y* if *x* equals *y*.

44113 The *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions shall be equivalent to the
44114 corresponding *nextafter()* functions, except that the second parameter shall have type **long**
44115 **double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

44116 An application wishing to check for error situations should set *errno* to zero and call
44117 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
44118 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
44119 zero, an error has occurred.

44120 RETURN VALUE

44121 Upon successful completion, these functions shall return the next representable floating-point
44122 value following *x* in the direction of *y*.

44123 If $x=y$, *y* (of the type *x*) shall be returned.

44124 If *x* is finite and the correct function value would overflow, a range error shall occur and
44125 \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL (with the same sign as *x*) shall be returned as
44126 appropriate for the return type of the function.

44127 MX If *x* or *y* is NaN, a NaN shall be returned.

44128 If $x \neq y$ and the correct function value is subnormal, zero, or underflows, a range error shall
44129 occur, and either the correct function value (if representable) or 0.0 shall be returned.

44130 ERRORS

44131 These functions shall fail if:

44132 Range Error The correct value overflows.

44133 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
44134 then *errno* shall be set to [ERANGE]. If the integer expression
44135 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
44136 floating-point exception shall be raised.

44137 MX **Range Error** The correct value is subnormal or underflows.
 44138 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 44139 then *errno* shall be set to [ERANGE]. If the integer expression
 44140 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 44141 floating-point exception shall be raised.

EXAMPLES

44142 None.
 44143

APPLICATION USAGE

44144 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 44145 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.
 44146

RATIONALE

44147 None.
 44148

FUTURE DIRECTIONS

44149 None.
 44150

SEE ALSO

44151 *feclearexcept()*, *fetestexcept()*
 44152

44153 XBD Section 4.19 (on page 104), <math.h>

CHANGE HISTORY

44154 First released in Issue 4, Version 2.
 44155

Issue 5

44156 Moved from X/OPEN UNIX extension to BASE.
 44157

Issue 6

44158 The *nextafter()* function is no longer marked as an extension.
 44159

44160 The *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, and *nexttowardl()* functions are added
 44161 for alignment with the ISO/IEC 9899:1999 standard.

44162 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
 44163 revised to align with the ISO/IEC 9899:1999 standard.

44164 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
 44165 marked.

44166 **NAME**

44167 nftw — walk a file tree

44168 **SYNOPSIS**

```
44169 XSI #include <ftw.h>
44170 int nftw(const char *path, int (*fn)(const char *,
44171     const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

44172 **DESCRIPTION**

44173 The *nftw()* function shall recursively descend the directory hierarchy rooted in *path*. The *nftw()*
 44174 function has a similar effect to *ftw()* except that it takes an additional argument *flags*, which is a
 44175 bitwise-inclusive OR of zero or more of the following flags:

44176 **FTW_CHDIR** If set, *nftw()* shall change the current working directory to each directory as it
 44177 reports files in that directory. If clear, *nftw()* shall not change the current
 44178 working directory.

44179 **FTW_DEPTH** If set, *nftw()* shall report all files in a directory before reporting the directory
 44180 itself. If clear, *nftw()* shall report any directory before reporting the files in that
 44181 directory.

44182 **FTW_MOUNT** If set, *nftw()* shall only report files in the same file system as *path*. If clear,
 44183 *nftw()* shall report all files encountered during the walk.

44184 **FTW_PHYS** If set, *nftw()* shall perform a physical walk and shall not follow symbolic links.

44185 If **FTW_PHYS** is clear and **FTW_DEPTH** is set, *nftw()* shall follow links instead of reporting
 44186 them, but shall not report any directory that would be a descendant of itself. If **FTW_PHYS** is
 44187 clear and **FTW_DEPTH** is clear, *nftw()* shall follow links instead of reporting them, but shall not
 44188 report the contents of any directory that would be a descendant of itself.

44189 At each file it encounters, *nftw()* shall call the user-supplied function *fn* with four arguments:

- 44190 • The first argument is the pathname of the object.
- 44191 • The second argument is a pointer to the **stat** buffer containing information on the object,
 44192 filled in as if *fstatat()*, *stat()*, or *lstat()* had been called to retrieve the information.
- 44193 • The third argument is an integer giving additional information. Its value is one of the
 44194 following:

44195 **FTW_F** The object is a file.

44196 **FTW_D** The object is a directory.

44197 **FTW_DP** The object is a directory and subdirectories have been visited. (This condition
 44198 shall only occur if the **FTW_DEPTH** flag is included in *flags*.)

44199 **FTW_SL** The object is a symbolic link. (This condition shall only occur if the
 44200 **FTW_PHYS** flag is included in *flags*.)

44201 **FTW_SLN** The object is a symbolic link that does not name an existing file. (This
 44202 condition shall only occur if the **FTW_PHYS** flag is not included in *flags*.)

44203 **FTW_DNR** The object is a directory that cannot be read. The *fn* function shall not be
 44204 called for any of its descendants.

44205 FTW_NS The *stat()* function failed on the object because of lack of appropriate
 44206 permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat()* for any
 44207 other reason is considered an error and *nftw()* shall return -1 .

44208 • The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the
 44209 object's filename in the pathname passed as the first argument to *fn*. The value of **level**
 44210 indicates depth relative to the root of the walk, where the root level is 0.

44211 The results are unspecified if the application-supplied *fn* function does not preserve the current
 44212 working directory.

44213 The argument *fd_limit* sets the maximum number of file descriptors that shall be used by *nftw()*
 44214 while traversing the file tree. At most one file descriptor shall be used for each directory level.

44215 The *nftw()* function need not be thread-safe. A function that is not required to be thread-safe is
 44216 not required to be reentrant.

44217 RETURN VALUE

44218 The *nftw()* function shall continue until the first of the following conditions occurs:

- 44219 • An invocation of *fn* shall return a non-zero value, in which case *nftw()* shall return that
 44220 value.
- 44221 • The *nftw()* function detects an error other than [EACCES] (see FTW_DNR and FTW_NS
 44222 above), in which case *nftw()* shall return -1 and set *errno* to indicate the error.
- 44223 • The tree is exhausted, in which case *nftw()* shall return 0.

44224 ERRORS

44225 The *nftw()* function shall fail if:

44226 [EACCES] Search permission is denied for any component of *path* or read permission is
 44227 denied for *path*, or *fn* returns -1 and does not reset *errno*.

44228 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
 44229 argument.

44230 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
 44231 component is longer than {NAME_MAX}.
 44232

44233 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

44234 [ENOTDIR] A component of *path* is not a directory.

44235 [EOVERFLOW] A field in the **stat** structure cannot be represented correctly in the current
 44236 programming environment for one or more files found in the file hierarchy.

44237 The *nftw()* function may fail if:

44238 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 44239 resolution of the *path* argument.

44240 [EMFILE] All file descriptors available to the process are currently open.

44241 [ENAMETOOLONG] Pathname resolution of a symbolic link produced an intermediate result
 44242 whose length exceeds {PATH_MAX}.
 44243

44244 [ENFILE] Too many files are currently open in the system.

44245 In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

EXAMPLES

The following example walks the `/tmp` directory and its subdirectories, calling the `nftw()` function for every directory entry, using a maximum of 5 file descriptors.

```

44246 #include <ftw.h>
44247 ...
44248 int nftwfunc(const char *, const struct stat *, int, struct FTW *);
44249
44250 int nftwfunc(const char *filename, const struct stat *statptr,
44251             int fileflags, struct FTW *pftw)
44252 {
44253     return 0;
44254 }
44255 ...
44256 char *startpath = "/tmp";
44257 int fd_limit = 5;
44258 int flags = FTW_CHDIR | FTW_DEPTH | FTW_MOUNT;
44259 int ret;
44260
44261 ret = nftw(startpath, nftwfunc, fd_limit, flags);

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*fdopendir\(\)*](#), [*fstatat\(\)*](#), [*readdir\(\)*](#)

XBD [`<ftw.h>`](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

In the DESCRIPTION, the definition of the *depth* argument is clarified.

Issue 6

The Open Group Base Resolution bwg97-003 is applied.

The ERRORS section is updated as follows:

- The wording of the mandatory [ELOOP] error condition is updated.
- A second optional [ELOOP] error condition is added.
- The [EOVERFLOW] mandatory error condition is added.

Text is added to the DESCRIPTION to say that the `nftw()` function need not be reentrant and that the results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/64 is applied, changing the argument *depth* to *fd_limit* throughout and changing “to a maximum of 5 levels deep” to “using a maximum of 5 file descriptors” in the EXAMPLES section.

44289

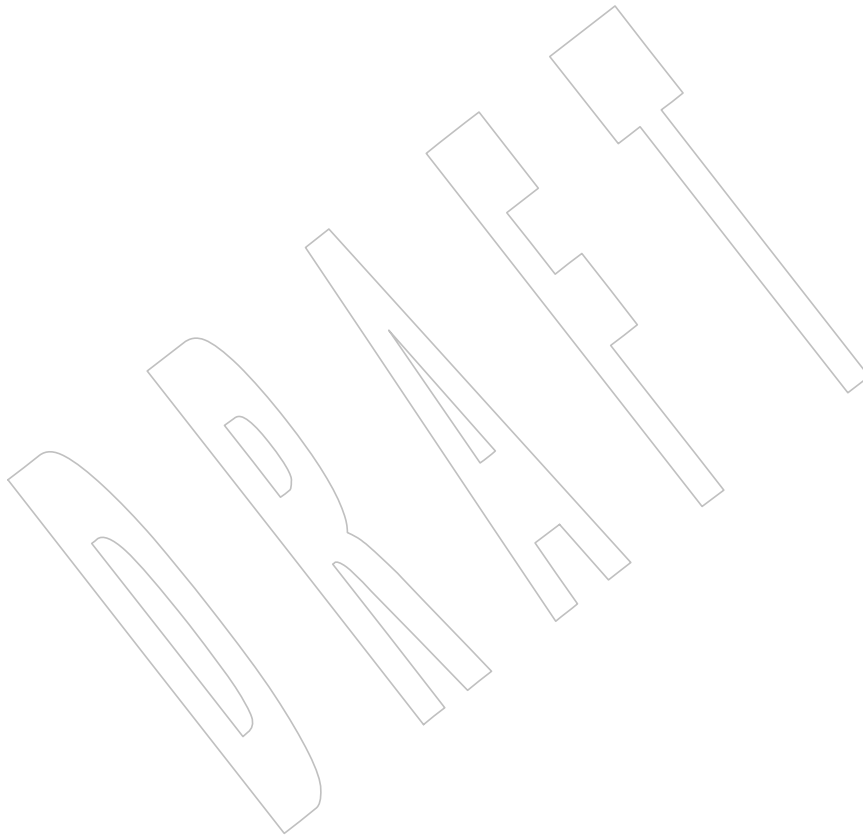
Issue 7

44290

SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

44291

SD5-XBD-ERN-61 is applied.



44292 **NAME**44293 `nice` — change the nice value of a process44294 **SYNOPSIS**44295 XSI

```
#include <unistd.h>
44296 int nice(int incr);
```

44297 **DESCRIPTION**44298 The `nice()` function shall add the value of `incr` to the nice value of the calling process. A nice
44299 value of a process is a non-negative number for which a more positive value shall result in less
44300 favorable scheduling.44301 A maximum nice value of $2*\{NZERO\}-1$ and a minimum nice value of 0 shall be imposed by the
44302 system. Requests for values above or below these limits shall result in the nice value being set to
44303 the corresponding limit. Only a process with appropriate privileges can lower the nice value.44304 PS|TPS Calling the `nice()` function has no effect on the priority of processes or threads with policy
44305 SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies
44306 is implementation-defined.44307 The nice value set with `nice()` shall be applied to the process. If the process is multi-threaded, the
44308 nice value shall affect all system scope threads in the process.44309 As -1 is a permissible return value in a successful situation, an application wishing to check for
44310 error situations should set `errno` to 0, then call `nice()`, and if it returns -1 , check to see whether
44311 `errno` is non-zero.44312 **RETURN VALUE**44313 Upon successful completion, `nice()` shall return the new nice value $-\{NZERO\}$. Otherwise, -1
44314 shall be returned, the nice value of the process shall not be changed, and `errno` shall be set to
44315 indicate the error.44316 **ERRORS**44317 The `nice()` function shall fail if:44318 [EPERM] The `incr` argument is negative and the calling process does not have
44319 appropriate privileges.44320 **EXAMPLES**44321 **Changing the Nice Value**44322 The following example adds the value of the `incr` argument, -20 , to the nice value of the calling
44323 process.44324

```
#include <unistd.h>
44325 ...
44326 int incr = -20;
44327 int ret;
44328 ret = nice(incr);
```

44329 **APPLICATION USAGE**

44330 None.

44331

RATIONALE

44332

None.

44333

FUTURE DIRECTIONS

44334

None.

44335

SEE ALSO

44336

exec, *getpriority()*

44337

XBD [<limits.h>](#), [<unistd.h>](#)

44338

CHANGE HISTORY

44339

First released in Issue 1. Derived from Issue 1 of the SVID.

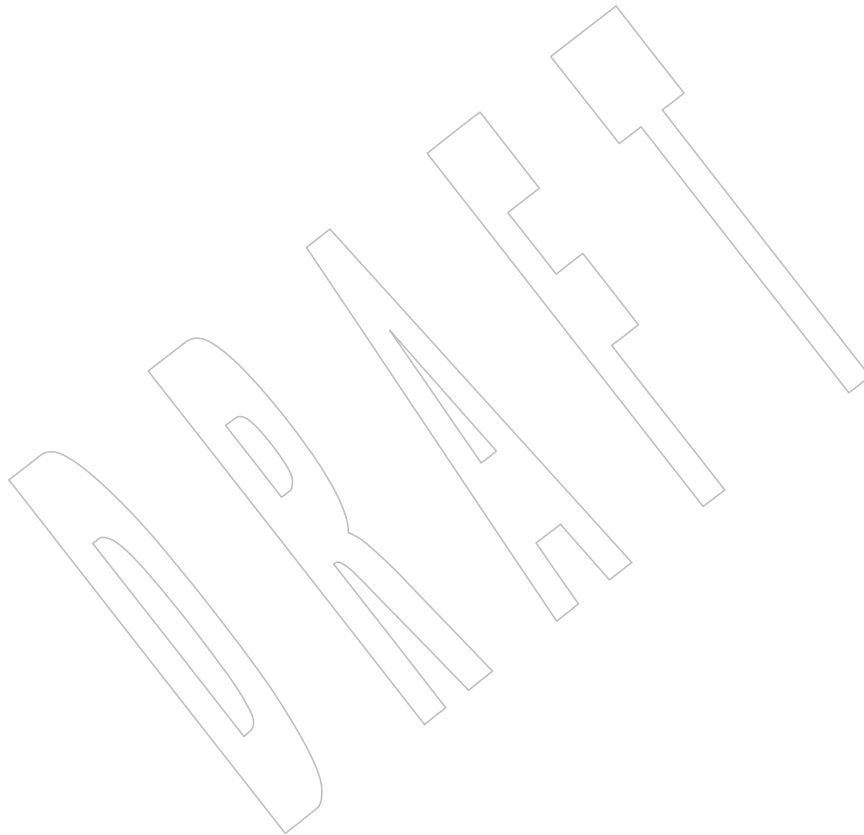
44340

Issue 5

44341

A statement is added to the description indicating the effects of this function on the different scheduling policies and multi-threaded processes.

44342



44343 **NAME**
 44344 nl_langinfo, nl_langinfo_l — language information

44345 **SYNOPSIS**
 44346 #include <langinfo.h>
 44347 char *nl_langinfo(nl_item item);
 44348 char *nl_langinfo_l(nl_item item, locale_t locale);

44349 **DESCRIPTION**
 44350 The *nl_langinfo()* and *nl_langinfo_l()* functions shall return a pointer to a string containing
 44351 information relevant to the particular language or cultural area defined in the locale of the
 44352 process, or in the locale represented by *locale*, respectively (see <langinfo.h>). The manifest
 44353 constant names and values of *item* are defined in <langinfo.h>. For example:

44354 nl_langinfo(ABDAY_1)
 44355 would return a pointer to the string "Dom" if the identified language was Portuguese, and
 44356 "Sun" if the identified language was English.

44357 nl_langinfo_l(ABDAY_1, loc)
 44358 would return a pointer to the string "Dom" if the identified language of the locale represented by
 44359 *loc* was Portuguese, and "Sun" if the identified language of the locale represented by *loc* was
 44360 English.

44361 Calls to *setlocale()* with a category corresponding to the category of *item* (see <langinfo.h>), or to
 44362 the category *LC_ALL*, may overwrite the array pointed to by the return value. Calls to *uselocale()*
 44363 which change the category corresponding to the category of *item* may overwrite the array
 44364 pointed to by the return value.

44365 The *nl_langinfo()* function need not be thread-safe. A function that is not required to be thread-
 44366 safe is not required to be reentrant.

44367 **RETURN VALUE**
 44368 In a locale where *langinfo* data is not defined, these functions shall return a pointer to the
 44369 corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to
 44370 an empty string if *item* contains an invalid setting.

44371 This pointer may point to static data that may be overwritten on the next call to either function.

44372 **ERRORS**
 44373 The *nl_langinfo_l()* function may fail if:
 44374 [EINVAL] *locale* is not a valid locale object handle.

44375 EXAMPLES

44376 Getting Date and Time Formatting Information

44377 The following example returns a pointer to a string containing date and time formatting
 44378 information, as defined in the *LC_TIME* category of the current locale.

```
44379 #include <time.h>
44380 #include <langinfo.h>
44381 ...
44382 strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
44383 ...
```

44384
44385
44386
44387
44388
44389
44390
44391
44392
44393
44394
44395
44396
44397
44398
44399
44400
44401
44402

APPLICATION USAGE

The array pointed to by the return value should not be modified by the program, but may be modified by further calls to these functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale(), *uselocale()*

XBD Chapter 7 (on page 121), [<langinfo.h>](#), [<locale.h>](#), [<nl_types.h>](#)

CHANGE HISTORY

First released in Issue 2.

Issue 5

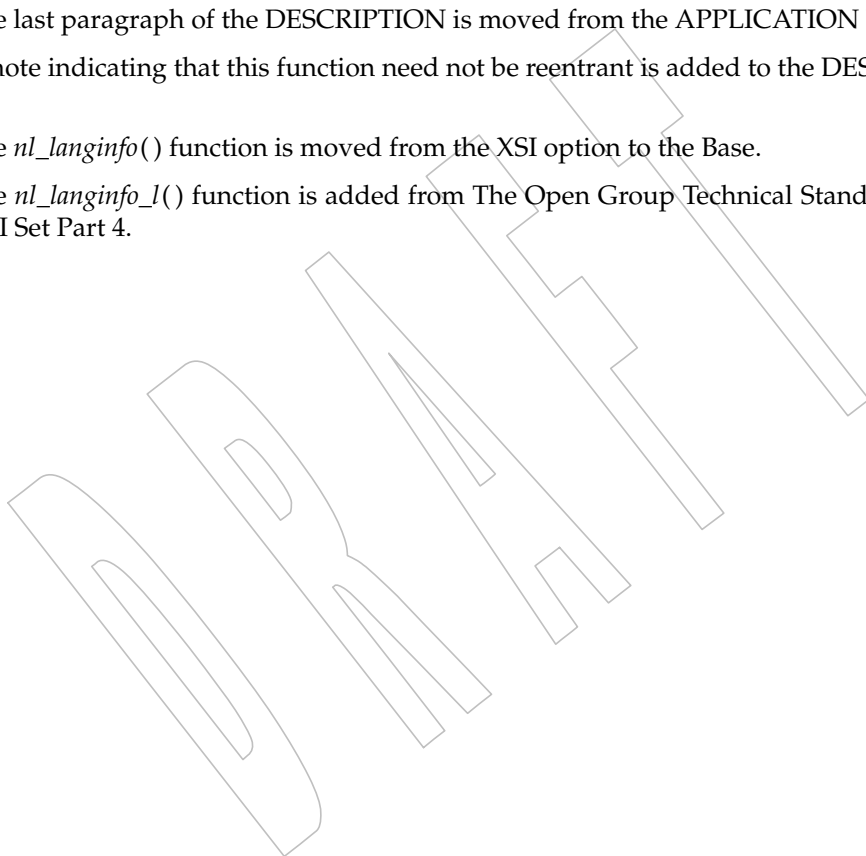
The last paragraph of the DESCRIPTION is moved from the APPLICATION USAGE section.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 7

The *nl_langinfo()* function is moved from the XSI option to the Base.

The *nl_langinfo_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



nrand48()*System Interfaces*

44403 **NAME**
44404 nrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

44405 XSI #include <stdlib.h>
44406
44407 long nrand48(unsigned short xsubi[3]);

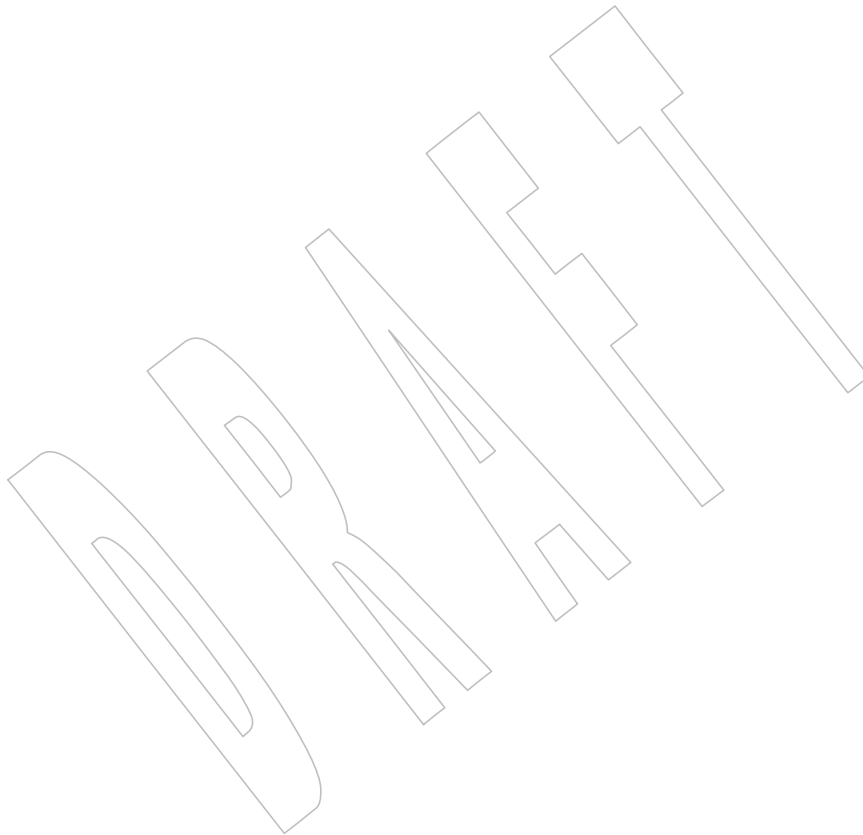
DESCRIPTION

44408 Refer to *drand48()*.
44409

44410 **NAME**
44411 ntohl, ntohs — convert values between host and network byte order

44412 **SYNOPSIS**
44413 #include <arpa/inet.h>
44414 uint32_t ntohl(uint32_t *netlong*);
44415 uint16_t ntohs(uint16_t *netshort*);

44416 **DESCRIPTION**
44417 Refer to *htonl()*.



44418 **NAME**
 44419 open, openat — open file relative to directory file descriptor

44420 **SYNOPSIS**

```
44421 OH #include <sys/stat.h>
44422 #include <fcntl.h>

44423 int open(const char *path, int oflag, ...);
44424 int openat(int fd, const char *path, int oflag, ...);
```

44425 **DESCRIPTION**

44426 The *open()* function shall establish the connection between a file and a file descriptor. It shall
 44427 create an open file description that refers to a file and a file descriptor that refers to that open file
 44428 description. The file descriptor is used by other I/O functions to refer to that file. The *path*
 44429 argument points to a pathname naming the file.

44430 The *open()* function shall return a file descriptor for the named file that is the lowest file
 44431 descriptor not currently open for that process. The open file description is new, and therefore the
 44432 file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file
 44433 descriptor flag associated with the new file descriptor shall be cleared.

44434 The file offset used to mark the current position within the file shall be set to the beginning of
 44435 the file.

44436 The file status flags and file access modes of the open file description shall be set according to
 44437 the value of *oflag*.

44438 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 44439 in **<fcntl.h>**. Applications shall specify exactly one of the first four values (file access modes)
 44440 below in the value of *oflag*:

44441 O_EXEC Open for execute only (non-directory files). Use of this flag on directories
 44442 is currently unspecified.

44443 O_RDONLY Open for reading only.

44444 O_WRONLY Open for writing only.

44445 O_RDWR Open for reading and writing. The result is undefined if this flag is
 44446 applied to a FIFO.

44447 Any combination of the following may be used:

44448 O_APPEND If set, the file offset shall be set to the end of the file prior to each write.

44449 O_CREAT If the file exists, this flag has no effect except as noted under O_EXCL
 44450 below. Otherwise, the file shall be created; the user ID of the file shall be
 44451 set to the effective user ID of the process; the group ID of the file shall be
 44452 set to the group ID of the file's parent directory or to the effective group
 44453 ID of the process; and the access permission bits (see **<sys/stat.h>**) of the
 44454 file mode shall be set to the value of the argument following the *oflag*
 44455 argument taken as type **mode_t** modified as follows: a bitwise AND is
 44456 performed on the file-mode bits and the corresponding bits in the
 44457 complement of the process' file mode creation mask. Thus, all bits in the
 44458 file mode whose corresponding bit in the file mode creation mask is set
 44459 are cleared. When bits other than the file permission bits are set, the effect
 44460 is unspecified. The argument following the *oflag* argument does not affect
 44461 whether the file is open for reading, writing, or for both. Implementations
 44462 shall provide a way to initialize the file's group ID to the group ID of the

44463			parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.
44464			
44465			
44466		O_DIRECTORY	If <i>path</i> does not name a directory, fail and set <i>errno</i> to [ENOTDIR].
44467	SIO	O_DSYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.
44468			
44469		O_EXCL	If O_CREAT and O_EXCL are set, <i>open()</i> shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing <i>open()</i> naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and <i>path</i> names a symbolic link, <i>open()</i> shall fail and set <i>errno</i> to [EEXIST], regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.
44470			
44471			
44472			
44473			
44474			
44475			
44476			
44477		O_NOCTTY	If set and <i>path</i> identifies a terminal device, <i>open()</i> shall not cause the terminal device to become the controlling terminal for the process.
44478			
44479		O_NOFOLLOW	If <i>path</i> names a symbolic link, fail and set <i>errno</i> to [ELOOP]. When opening a FIFO with O_RDONLY or O_WRONLY set:
44480			
44481			• If O_NONBLOCK is set, an <i>open()</i> for reading-only shall return without delay. An <i>open()</i> for writing-only shall return an error if no process currently has the file open for reading.
44482			
44483			
44484			• If O_NONBLOCK is clear, an <i>open()</i> for reading-only shall block the calling thread until a thread opens the file for writing. An <i>open()</i> for writing-only shall block the calling thread until a thread opens the file for reading.
44485			
44486			
44487			
44488			When opening a block special or character special file that supports non-blocking opens:
44489			
44490			• If O_NONBLOCK is set, the <i>open()</i> function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
44491			
44492			
44493			• If O_NONBLOCK is clear, the <i>open()</i> function shall block the calling thread until the device is ready or available before returning.
44494			
44495			Otherwise, the behavior of O_NONBLOCK is unspecified.
44496	SIO	O_RSYNC	Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in <i>oflag</i> , all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
44497			
44498			
44499			
44500			
44501			
44502			
44503	XSI SIO	O_SYNC	Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
44504			
44505	XSI		The O_SYNC flag shall be supported for regular files, even if the Synchronized Input and Output option is not supported.
44506			
44507		O_TRUNC	If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files
44508			
44509			

open()

44510		or terminal device files. Its effect on other file types is implementation-
44511		defined. The result of using O_TRUNC without either O_RDWR or
44512		O_WRONLY is undefined.
44513		If O_CREAT is set and the file did not previously exist, upon successful completion, <i>open()</i> shall
44514		mark for update the last data access, last data modification, and last file status change
44515		timestamps of the file and the last data modification and last file status change timestamps of
44516		the parent directory.
44517		If O_TRUNC is set and the file did previously exist, upon successful completion, <i>open()</i> shall
44518		mark for update the last data modification and last file status change timestamps of the file.
44519	SIO	If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.
44520	OB XSR	If <i>path</i> refers to a STREAMS file, <i>oflag</i> may be constructed from O_NONBLOCK OR'ed with
44521		either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS
44522		devices and shall have no effect on them. The value O_NONBLOCK affects the operation of
44523		STREAMS drivers and certain functions applied to file descriptors associated with STREAMS
44524		files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.
44525	XSI	If <i>path</i> names the master side of a pseudo-terminal device, then it is unspecified whether <i>open()</i>
44526		locks the slave side so that it cannot be opened. Conforming applications shall call <i>unlockpt()</i>
44527		before opening the slave side.
44528		The largest value that can be represented correctly in an object of type off_t shall be established
44529		as the offset maximum in the open file description.
44530		The <i>openat()</i> function shall be equivalent to the <i>open()</i> function except in the case where <i>path</i>
44531		specifies a relative path. In this case the file to be opened is determined relative to the directory
44532		associated with the file descriptor <i>fd</i> instead of the current working directory. It is unspecified
44533		whether directory searches are permitted based on whether the file was opened with search
44534		permission or on the current permissions of the directory underlying the file descriptor.
44535		The <i>oflag</i> parameter and the optional fourth parameter correspond exactly to the parameters of
44536		<i>open()</i> .
44537		If <i>openat()</i> is passed the special value AT_FDCWD in the <i>fd</i> parameter, the current working
44538		directory is used and the behavior shall be identical to a call to <i>open()</i> .
44539		RETURN VALUE
44540		Upon successful completion, these functions shall open the file and return a non-negative
44541		integer representing the lowest numbered unused file descriptor. Otherwise, these functions
44542		shall return -1 and set <i>errno</i> to indicate the error. If $-$ is returned, no files shall be created or
44543		modified.
44544		ERRORS
44545		These functions shall fail if:
44546	[EACCES]	Search permission is denied on a component of the path prefix, or the file
44547		exists and the permissions specified by <i>oflag</i> are denied, or the file does not
44548		exist and write permission is denied for the parent directory of the file to be
44549		created, or O_TRUNC is specified and write permission is denied.
44550	[EEXIST]	O_CREAT and O_EXCL are set, and the named file exists.
44551	[EINTR]	A signal was caught during <i>open()</i> .
44552	SIO [EINVAL]	The implementation does not support synchronized I/O for this file.

44553	OB XSR	[EIO]	The <i>path</i> argument names a STREAMS file and a hangup or error occurred during the <i>open()</i> .
44554			
44555		[EISDIR]	The named file is a directory and <i>oflag</i> includes O_WRONLY or O_RDWR.
44556		[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument, or O_NOFOLLOW was specified and the <i>path</i> argument names a symbolic link.
44557			
44558			
44559		[EMFILE]	All file descriptors available to the process are currently open.
44560		[ENAMETOOLONG]	
44561			The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
44562			
44563		[ENFILE]	The maximum allowable number of files is currently open in the system.
44564		[ENOENT]	O_CREAT is not set and the named file does not exist; or O_CREAT is set and either the path prefix does not exist or the <i>path</i> argument points to an empty string.
44565			
44566			
44567	OB XSR	[ENOSR]	The <i>path</i> argument names a STREAMS-based file and the system is unable to allocate a STREAM.
44568			
44569		[ENOSPC]	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.
44570			
44571		[ENOTDIR]	A component of the path prefix is not a directory, or O_DIRECTORY was specified and the <i>path</i> argument does not name a directory.
44572			
44573		[ENXIO]	O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.
44574			
44575		[ENXIO]	The named file is a character special or block special file, and the device associated with this special file does not exist.
44576			
44577		[EOVERFLOW]	The named file is a regular file and the size of the file cannot be represented correctly in an object of type off_t .
44578			
44579		[EROFS]	The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the <i>oflag</i> argument.
44580			
44581			
44582			The <i>openat()</i> function shall fail if:
44583		[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading.
44584			
44585			These functions may fail if:
44586	XSI	[EAGAIN]	The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.
44587			
44588		[EINVAL]	The value of the <i>oflag</i> argument is not valid.
44589		[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
44590			
44591		[ENAMETOOLONG]	
44592			As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.
44593			
44594	OB XSR	[ENOMEM]	The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.
44595			

44596 [ETXTBSY] The file is a pure procedure (shared text) file that is being executed and *oflag* is
 44597 O_WRONLY or O_RDWR.

44598 The *openat()* function may fail if:

44599 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
 44600 file descriptor associated with a directory.

44601 EXAMPLES

44602 Opening a File for Writing by the Owner

44603 The following example opens the file */tmp/file*, either by creating it (if it does not already exist),
 44604 or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file,
 44605 the access permission bits in the file mode of the file are set to permit reading and writing by the
 44606 owner, and to permit reading only by group members and others.

44607 If the call to *open()* is successful, the file is opened for writing.

```
44608 #include <fcntl.h>
44609 ...
44610 int fd;
44611 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
44612 char *filename = "/tmp/file";
44613 ...
44614 fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);
44615 ...
```

44616 Opening a File Using an Existence Check

44617 The following example uses the *open()* function to try to create the **LOCKFILE** file and open it
 44618 for writing. Since the *open()* function specifies the O_EXCL flag, the call fails if the file already
 44619 exists. In that case, the program assumes that someone else is updating the password file and
 44620 exits.

```
44621 #include <fcntl.h>
44622 #include <stdio.h>
44623 #include <stdlib.h>
44624 #define LOCKFILE "/etc/ptmp"
44625 ...
44626 int pfd; /* Integer for file descriptor returned by open() call. */
44627 ...
44628 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
44629     S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
44630 {
44631     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
44632     exit(1);
44633 }
44634 ...
```

44635

Opening a File for Writing

44636

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

44637

44638

```
#include <fcntl.h>
```

44639

```
#include <stdio.h>
```

44640

```
#include <stdlib.h>
```

44641

```
#define LOCKFILE "/etc/ptmp"
```

44642

```
...
```

44643

```
int pfd;
```

44644

```
char filename[PATH_MAX+1];
```

44645

```
...
```

44646

```
if ((pfd = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
```

44647

```
 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
```

44648

```
{
```

44649

```
    perror("Cannot open output file\n"); exit(1);
```

44650

```
}
```

44651

```
...
```

44652

APPLICATION USAGE

44653

None.

44654

RATIONALE

44655

Except as specified in this volume of POSIX.1-200x, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously.

44656

44657

Some implementations permit opening FIFOs with `O_RDWR`. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, this possibility is left as undefined.

44658

44659

44660

See [getgroups\(\)](#) about the group of a newly created file.

44661

The use of `open()` to create a regular file is preferable to the use of `creat()`, because the latter is redundant and included only for historical reasons.

44662

44663

The use of the `O_TRUNC` flag on FIFOs and directories (pipes cannot be `open()`-ed) must be permissible without unexpected side effects (for example, `creat()` on a FIFO must not remove data). Since terminal special files might have type-ahead data stored in the buffer, `O_TRUNC` should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead. Other file types, particularly implementation-defined ones, are left implementation-defined.

44664

44665

44666

44667

44668

44669

POSIX.1-200x permits [EACCES] to be returned for conditions other than those explicitly listed.

44670

The `O_NOCTTY` flag was added to allow applications to avoid unintentionally acquiring a controlling terminal as a side effect of opening a terminal file. This volume of POSIX.1-200x does not specify how a controlling terminal is acquired, but it allows an implementation to provide this on `open()` if the `O_NOCTTY` flag is not set and other conditions specified in XBD [Chapter 11](#) (on page 185) are met. The `O_NOCTTY` flag is an effective no-op if the file being opened is not a terminal device.

44671

44672

44673

44674

44675

44676

In historical implementations the value of `O_RDONLY` is zero. Because of that, it is not possible to detect the presence of `O_RDONLY` and another option. Future implementations should encode `O_RDONLY` and `O_WRONLY` as bit flags so that:

44677

44678

44679

```
O_RDONLY | O_WRONLY == O_RDWR
```

44680

`O_EXEC` is specified as one of the four file access modes. On implementations where none of `O_RDONLY`, `O_WRONLY`, or `O_RDWR` is zero, applications may open a directory with `O_EXEC`

44681

44682 OR'd in with one of the other three file access modes. On many historical implementations, this
44683 cannot be done since O_RDONLY has been defined to be zero.

44684 In general, the *open()* function follows the symbolic link if *path* names a symbolic link. However,
44685 the *open()* function, when called with O_CREAT and O_EXCL, is required to fail with [EEXIST]
44686 if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This
44687 behavior is required so that privileged applications can create a new file in a known location
44688 without the possibility that a symbolic link might cause the file to be created in a different
44689 location.

44690 In addition, the *open()* function refuses to open non-directories if the O_DIRECTORY flag is set.
44691 This avoids race conditions whereby a user might compromise the system by substituting a hard
44692 link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where
44693 opening a file even for read access might have undesirable side-effects.

44694 In addition, the *open()* function does not follow symbolic links if the O_NOFOLLOW flag is set.
44695 This avoids race conditions whereby a user might compromise the system by substituting a
44696 symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where
44697 opening a file even for read access might have undesirable side-effects.

44698 For example, a privileged application that must create a file with a predictable name in a user-
44699 writable directory, such as the user's home directory, could be compromised if the user creates a
44700 symbolic link with that name that refers to a nonexistent file in a system directory. If the user can
44701 influence the contents of a file, the user could compromise the system by creating a new system
44702 configuration or spool file that would then be interpreted by the system. The test for a symbolic
44703 link which refers to a nonexistent file must be atomic with the creation of a new file.

44704 The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group
44705 ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required
44706 that implementations provide a way to have the group ID be set to the group ID of the
44707 containing directory, but did not prohibit implementations also supporting a way to set the
44708 group ID to the effective group ID of the creating process. Conforming applications should not
44709 assume which group ID will be used. If it matters, an application can use *chown()* to set the
44710 group ID after the file is created, or determine under what conditions the implementation will
44711 set the desired group ID.

44712 The purpose of the *openat()* function is to enable opening files in directories other than the
44713 current working directory without exposure to race conditions. Any part of the path of a file
44714 could be changed in parallel to a call to *open()*, resulting in unspecified behavior. By opening a
44715 file descriptor for the target directory and using the *openat()* function it can be guaranteed that
44716 the opened file is located relative to the desired directory. Some implementations use the
44717 *openat()* function for other purposes as well. In some cases, if the *oflag* parameter has the
44718 O_XATTR bit set, the returned file descriptor provides access to extended attributes. This
44719 functionality is not standardized here.

44720 FUTURE DIRECTIONS

44721 The meaning of the O_EXEC flag on directories may be specified in a future version.

44722 SEE ALSO

44723 *chmod*, *close()*, *creat()*, *dirfd()*, *dup()*, *exec*, *fcntl()*, *fdopendir()*, *link*, *lseek()*, *mkdtemp()*, *mknod()*,
44724 *read*, *symlink()*, *umask*, *unlockpt()*, *write*

44725 XBD Chapter 11 (on page 185), [<fcntl.h>](#), [<sys/stat.h>](#), [<sys/types.h>](#)

44726 CHANGE HISTORY

44727 First released in Issue 1. Derived from Issue 1 of the SVID.

44728	Issue 5	The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.		
44729				
44730				
44731		Large File Summit extensions are added.		
44732	Issue 6	In the SYNOPSIS, the optional include of the <code><sys/types.h></code> header is removed.		
44733		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:		
44734		<ul style="list-style-type: none"> • The requirement to include <code><sys/types.h></code> has been removed. Although <code><sys/types.h></code> was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications. • In the DESCRIPTION, <code>O_CREAT</code> is amended to state that the group ID of the file is set to the group ID of the file's parent directory or to the effective group ID of the process. This is a FIPS requirement. • In the DESCRIPTION, text is added to indicate setting of the offset maximum in the open file description. This change is to support large files. • In the ERRORS section, the <code>[EOVERFLOW]</code> condition is added. This change is to support large files. • The <code>[ENXIO]</code> mandatory error condition is added. • The <code>[EINVAL]</code>, <code>[ENAMETOOLONG]</code>, and <code>[ETXTBSY]</code> optional error conditions are added. 		
44736				
44737				
44738				
44739				
44740				
44741				
44742				
44743				
44744				
44745				
44746				
44747				
44748		The DESCRIPTION and ERRORS sections are updated so that items related to the optional XSI STREAMS Option Group are marked.		
44749				
44750		The following changes were made to align with the IEEE P1003.1a draft standard:		
44751		<ul style="list-style-type: none"> • An explanation is added of the effect of the <code>O_CREAT</code> and <code>O_EXCL</code> flags when the path refers to a symbolic link. • The <code>[ELOOP]</code> optional error condition is added. 		
44752				
44753				
44754		The normative text is updated to avoid use of the term "must" for application requirements.		
44755		The DESCRIPTION of <code>O_EXCL</code> is updated in response to IEEE PASC Interpretation 1003.1c #48.		
44756	Issue 7	Austin Group Interpretation 1003.1-2001 #113 is applied.	+	
44757				
44758		SD5-XBD-ERN-4 is applied, changing the definition of the <code>[EMFILE]</code> error.		
44759		This page is revised and the <code>openat()</code> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.		
44760				
44761		Functionality relating to the XSI STREAMS option is marked obsolescent.		
44762		Changes are made related to support for finegrained timestamps.		

44763 **NAME**

44764 open_memstream, open_wmemstream — open a dynamic memory buffer stream

44765 **SYNOPSIS**

```
44766 CX #include <stdio.h>
44767 FILE *open_memstream(char **bufp, size_t *sizep);
44768 #include <wchar.h>
44769 FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

44770 **DESCRIPTION**

44771 The *open_memstream()* and *open_wmemstream()* functions shall create an I/O stream associated
 44772 with a dynamically allocated memory buffer. The stream shall be opened for writing and shall
 44773 be seekable.

44774 The stream associated with a call to *open_memstream()* shall be byte-oriented.

44775 The stream associated with a call to *open_wmemstream()* shall be wide-oriented.

44776 The stream shall maintain a current position in the allocated buffer and a current buffer length.
 44777 The position shall be initially set to zero (the start of the buffer). Each write to the stream shall
 44778 start at the current position and move this position by the number of successfully written bytes
 44779 for *open_memstream()* or the number of successfully written wide characters for
 44780 *open_wmemstream()*. The length shall be initially set to zero. If a write moves the position to a
 44781 value larger than the current length, the current length shall be set to this position. In this case a
 44782 null character for *open_memstream()* or a null wide character for *open_wmemstream()* shall be
 44783 appended to the current buffer. For both functions the terminating null is not included in the
 44784 calculation of the buffer length.

44785 After a successful *fflush()* or *fclose()*, the pointer referenced by *bufp* shall contain the address of
 44786 the buffer, and the variable pointed to by *sizep* shall contain the number of bytes for
 44787 *open_memstream()* or the number of wide characters for *open_wmemstream()*, between the
 44788 beginning of the buffer and the current file position indicator. The buffer shall be terminated by
 44789 a null character for *open_memstream()* or a null wide character for *open_wmemstream()*, at the
 44790 current file position indicator.

44791 After a successful *fflush()* the pointer referenced by *bufp* and the variable referenced by *sizep*
 44792 remain valid only until the next write operation on the stream or a call to *fclose()*.

44793 **RETURN VALUE**

44794 Upon successful completion, these functions shall return a pointer to the object controlling the
 44795 stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

44796 **ERRORS**

44797 These functions may fail if:

- 44798 [EINVAL] *bufp* or *sizep* are NULL.
- 44799 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.
- 44800 [ENOMEM] Memory for the stream or the buffer could not be allocated.

EXAMPLES

```

44801
44802     #include <stdio.h>
44803     #include <stdlib.h>
44804
44805     int
44806     main (void)
44807     {
44808         FILE *stream;
44809         char *buf;
44810         size_t len;
44811         off_t eob;
44812
44813         stream = open_memstream (&buf, &len);
44814         if (stream == NULL)
44815             /* handle error */ ;
44816         fprintf (stream, "hello my world");
44817         fflush (stream);
44818         printf ("buf=%s, len=%zu\n", buf, len);
44819         eob = ftello(stream);
44820         fseeko (stream, 0, SEEK_SET);
44821         fprintf (stream, "good-bye");
44822         fseeko (stream, eob, SEEK_SET);
44823         fclose (stream);
44824         printf ("buf=%s, len=%zu\n", buf, len);
44825         free (buf);
44826         return 0;
44827     }

```

This program produces the following output:

```

44827     buf=hello my world, len=14
44828     buf=good-bye world, len=14

```

APPLICATION USAGE

The buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free()*.

RATIONALE

These functions are similar to *fnemopen()* except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose(), *fdopen()*, *fflush()*, *fnemopen()*, *fopen()*, *free()*, *freopen()*

XBD **<stdio.h>**

CHANGE HISTORY

First released in Issue 7.

44842 **NAME**
44843 open_wmemstream — open a dynamic memory buffer stream

44844 **SYNOPSIS**

```
44845 CX       #include <wchar.h>  
44846       FILE *open_wmemstream(wchar_t **bufp, size_t *sizep);
```

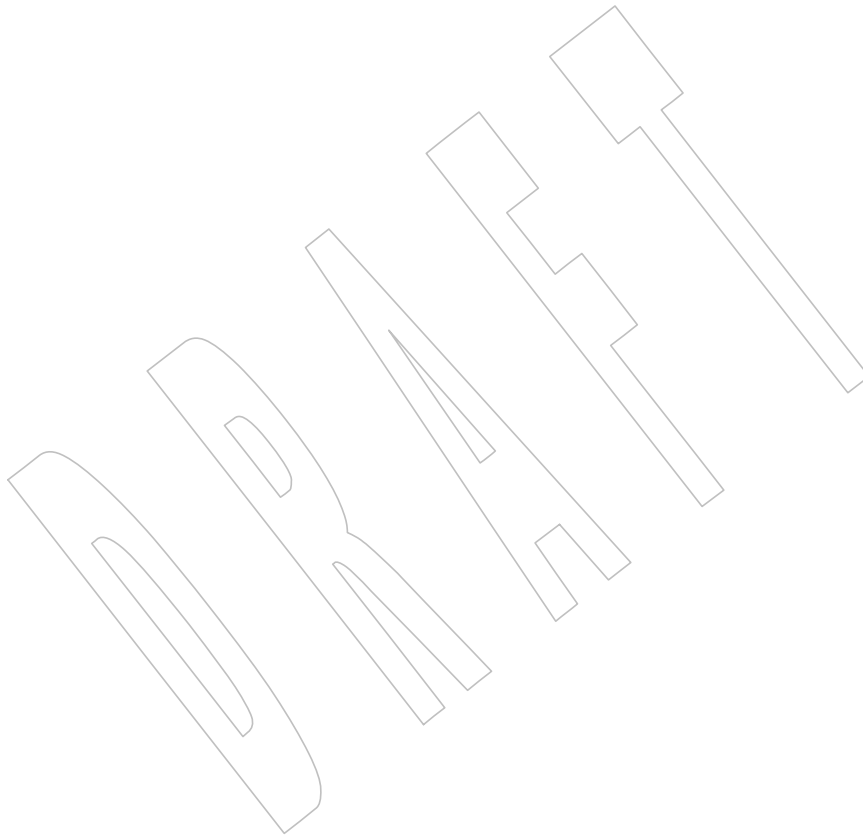
44847 **DESCRIPTION**

44848 Refer to *open_memstream()*.

44849 **NAME**
44850 `openat` — open file relative to directory file descriptor

44851 **SYNOPSIS**
44852 `#include <fcntl.h>`
44853 `int openat(int fd, const char *path, int oflag, ...);`

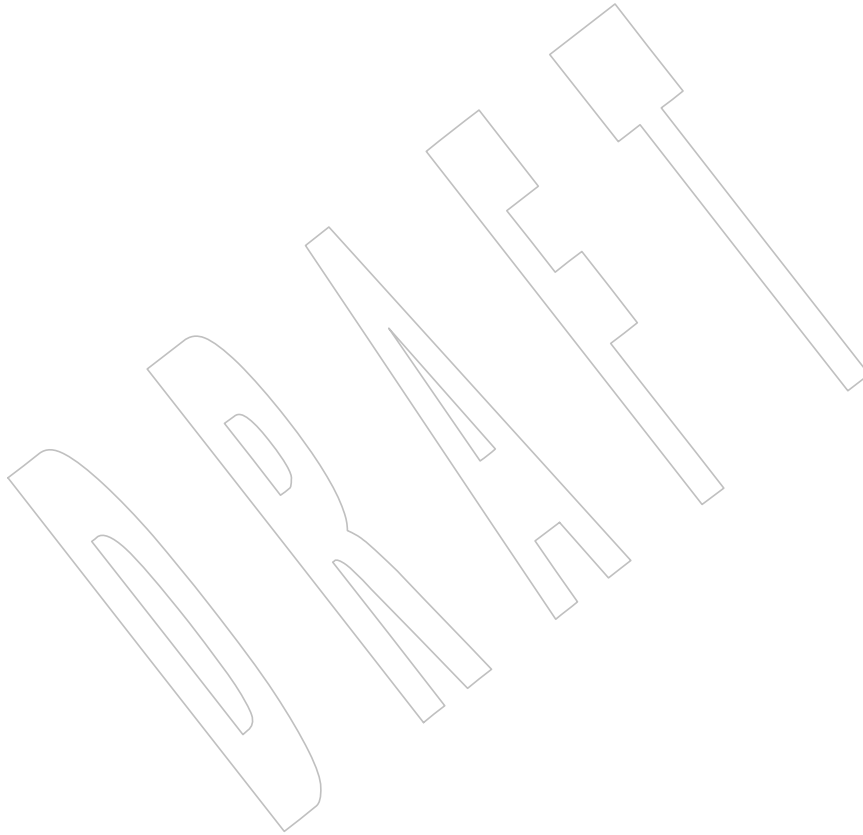
44854 **DESCRIPTION**
44855 Refer to `open()`.



44856 **NAME**
44857 opendir — open directory associated with file descriptor

44858 **SYNOPSIS**
44859 #include <dirent.h>
44860 DIR *opendir(const char *dirname);

44861 **DESCRIPTION**
44862 Refer to *fdopendir()*.



44863 **NAME**
44864 `openlog` — open a connection to the logging facility

44865 **SYNOPSIS**

44866 XSI `#include <syslog.h>`
44867 `void openlog(const char *ident, int logopt, int facility);`

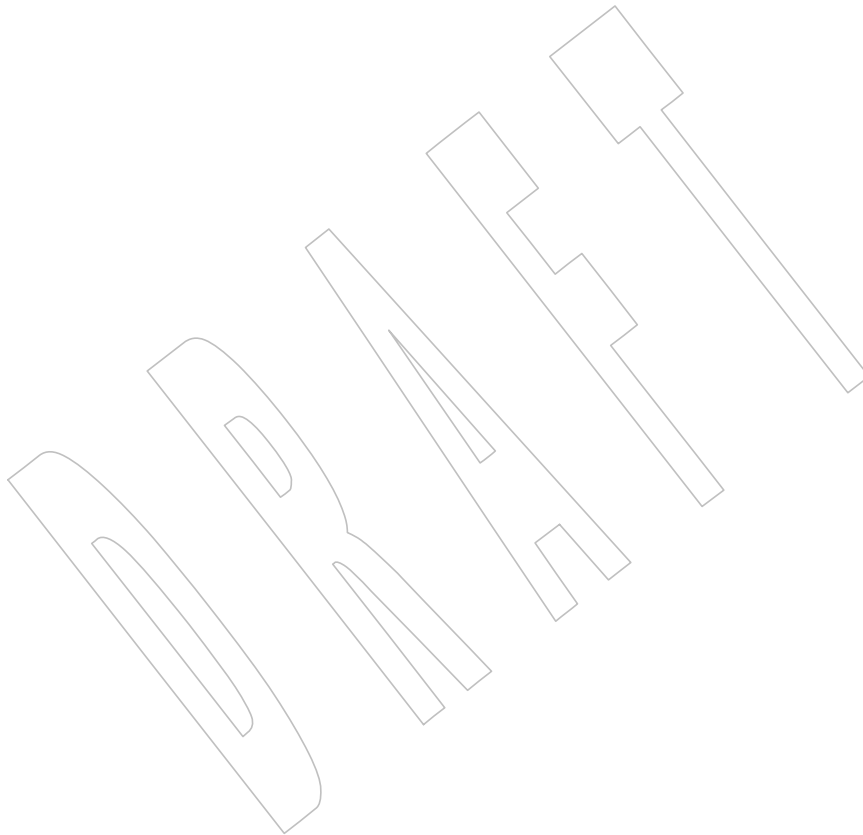
44868 **DESCRIPTION**

44869 Refer to *closelog()*.

44870 **NAME**
44871 `optarg, opterr, optind, optopt` — options parsing variables

44872 **SYNOPSIS**
44873 `#include <unistd.h>`
44874 `extern char *optarg;`
44875 `extern int opterr, optind, optopt;`

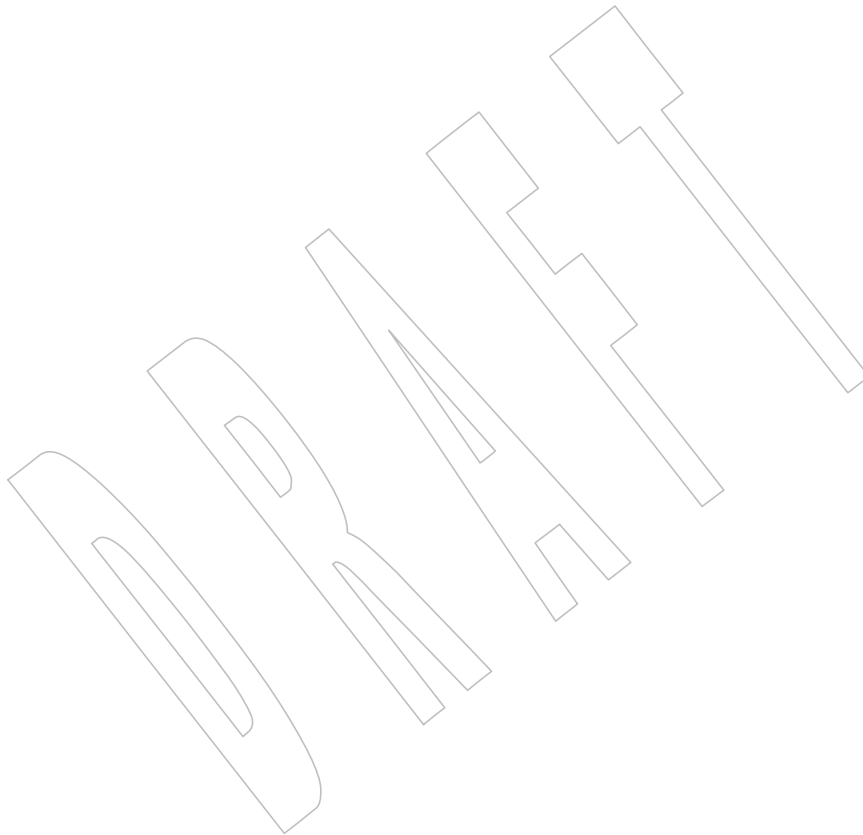
44876 **DESCRIPTION**
44877 Refer to *getopt()*.



44878 **NAME**
44879 `pathconf` — get configurable pathname variables

44880 **SYNOPSIS**
44881 `#include <unistd.h>`
44882 `long pathconf(const char *path, int name);`

44883 **DESCRIPTION**
44884 Refer to *[fpathconf\(\)](#)*.



44885 **NAME**
 44886 `pause` — suspend the thread until a signal is received

44887 **SYNOPSIS**
 44888 `#include <unistd.h>`
 44889 `int pause(void);`

44890 **DESCRIPTION**
 44891 The `pause()` function shall suspend the calling thread until delivery of a signal whose action is
 44892 either to execute a signal-catching function or to terminate the process.

44893 If the action is to terminate the process, `pause()` shall not return.

44894 If the action is to execute a signal-catching function, `pause()` shall return after the signal-catching
 44895 function returns.

44896 **RETURN VALUE**
 44897 Since `pause()` suspends thread execution indefinitely unless interrupted by a signal, there is no
 44898 successful completion return value. A value of `-1` shall be returned and `errno` set to indicate the
 44899 error.

44900 **ERRORS**
 44901 The `pause()` function shall fail if:
 44902 [EINTR] A signal is caught by the calling process and control is returned from the
 44903 signal-catching function.

44904 **EXAMPLES**
 44905 None.

44906 **APPLICATION USAGE**
 44907 Many common uses of `pause()` have timing windows. The scenario involves checking a
 44908 condition related to a signal and, if the signal has not occurred, calling `pause()`. When the signal
 44909 occurs between the check and the call to `pause()`, the process often blocks indefinitely. The
 44910 `sigprocmask()` and `sigsuspend()` functions can be used to avoid this type of problem.

44911 **RATIONALE**
 44912 None.

44913 **FUTURE DIRECTIONS**
 44914 None.

44915 **SEE ALSO**
 44916 [sigsuspend\(\)](#)
 44917 XBD [<unistd.h>](#)

44918 **CHANGE HISTORY**
 44919 First released in Issue 1. Derived from Issue 1 of the SVID.

44920 **Issue 5**
 44921 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

44922 **Issue 6**
 44923 The APPLICATION USAGE section is added.

44924 **NAME**
 44925 `pclose` — close a pipe stream to or from a process

44926 **SYNOPSIS**

44927 CX

```
#include <stdio.h>
```


 44928

```
int pclose(FILE *stream);
```

44929 **DESCRIPTION**

44930 The `pclose()` function shall close a stream that was opened by `popen()`, wait for the command to
 44931 terminate, and return the termination status of the process that was running the command
 44932 language interpreter. However, if a call caused the termination status to be unavailable to
 44933 `pclose()`, then `pclose()` shall return `-1` with `errno` set to `[ECHILD]` to report this situation. This can
 44934 happen if the application calls one of the following functions:

- 44935 • `wait()`
- 44936 • `waitpid()` with a `pid` argument less than or equal to 0 or equal to the process ID of the
 44937 command line interpreter
- 44938 • Any other function not defined in this volume of POSIX.1-200x that could do one of the
 44939 above

44940 In any case, `pclose()` shall not return before the child process created by `popen()` has terminated.

44941 If the command language interpreter cannot be executed, the child termination status returned
 44942 by `pclose()` shall be as if the command language interpreter terminated using `exit(127)` or
 44943 `_exit(127)`.

44944 The `pclose()` function shall not affect the termination status of any child of the calling process
 44945 other than the one created by `popen()` for the associated stream.

44946 If the argument `stream` to `pclose()` is not a pointer to a stream created by `popen()`, the result of
 44947 `pclose()` is undefined.

44948 **RETURN VALUE**

44949 Upon successful return, `pclose()` shall return the termination status of the command language
 44950 interpreter. Otherwise, `pclose()` shall return `-1` and set `errno` to indicate the error.

44951 **ERRORS**

44952 The `pclose()` function shall fail if:

44953 `[ECHILD]` The status of the child process could not be obtained, as described above.

44954 **EXAMPLES**

44955 None.

44956 **APPLICATION USAGE**

44957 None.

44958 **RATIONALE**

44959 There is a requirement that `pclose()` not return before the child process terminates. This is
 44960 intended to disallow implementations that return `[EINTR]` if a signal is received while waiting.
 44961 If `pclose()` returned before the child terminated, there would be no way for the application to
 44962 discover which child used to be associated with the stream, and it could not do the cleanup
 44963 itself.

44964 If the stream pointed to by `stream` was not created by `popen()`, historical implementations of
 44965 `pclose()` return `-1` without setting `errno`. To avoid requiring `pclose()` to set `errno` in this case,

44966 POSIX.1-200x makes the behavior unspecified. An application should not use *pclose()* to close
44967 an stream that was not created by *popen()*.

44968 Some historical implementations of *pclose()* either block or ignore the signals SIGINT, SIGQUIT,
44969 and SIGHUP while waiting for the child process to terminate. Since this behavior is not
44970 described for the *pclose()* function in POSIX.1-200x, such implementations are not conforming.
44971 Also, some historical implementations return [EINTR] if a signal is received, even though the
44972 child process has not terminated. Such implementations are also considered non-conforming.

44973 Consider, for example, an application that uses:

```
44974 popen("command", "r")
```

44975 to start *command*, which is part of the same application. The parent writes a prompt to its
44976 standard output (presumably the terminal) and then reads from the *popen()*ed stream. The child
44977 reads the response from the user, does some transformation on the response (pathname
44978 expansion, perhaps) and writes the result to its standard output. The parent process reads the
44979 result from the pipe, does something with it, and prints another prompt. The cycle repeats.
44980 Assuming that both processes do appropriate buffer flushing, this would be expected to work.

44981 To conform to POSIX.1-200x, *pclose()* must use *waitpid()*, or some similar function, instead of
44982 *wait()*.

44983 The code sample below illustrates how the *pclose()* function might be implemented on a system
44984 conforming to POSIX.1-200x.

```
44985 int pclose(FILE *stream)
44986 {
44987     int stat;
44988     pid_t pid;
44989     pid = <pid for process created for stream by popen(>
44990         (void) fclose(stream);
44991     while (waitpid(pid, &stat, 0) == -1) {
44992         if (errno != EINTR){
44993             stat = -1;
44994             break;
44995         }
44996     }
44997     return(stat);
44998 }
```

44999 FUTURE DIRECTIONS

45000 None.

45001 SEE ALSO

45002 *fork()*, *popen()*, *wait*

45003 XBD <stdio.h>

45004 CHANGE HISTORY

45005 First released in Issue 1. Derived from Issue 1 of the SVID.

45006 **NAME**
 45007 `perror` — write error messages to standard error

45008 **SYNOPSIS**
 45009 `#include <stdio.h>`

45010 `void perror(const char *s);`

45011 **DESCRIPTION**

45012 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 45013 conflict between the requirements described here and the ISO C standard is unintentional. This
 45014 volume of POSIX.1-200x defers to the ISO C standard.

45015 The `perror()` function shall map the error number accessed through the symbol `errno` to a
 45016 language-dependent error message, which shall be written to the standard error stream as
 45017 follows:

- 45018 • First (if `s` is not a null pointer and the character pointed to by `s` is not the null byte), the
 45019 string pointed to by `s` followed by a colon and a `<space>`.
- 45020 • Then an error message string followed by a `<newline>`.

45021 The contents of the error message strings shall be the same as those returned by `strerror()` with
 45022 argument `errno`.

45023 CX The `perror()` function shall mark the file associated with the standard error stream as having
 45024 been written (the last data modification and last file status change timestamps marked for
 45025 update) at some time between its successful completion and `exit()`, `abort()`, or the completion of
 45026 `fflush()` or `fclose()` on `stderr`.

45027 The `perror()` function shall not change the orientation of the standard error stream.

45028 **RETURN VALUE**

45029 The `perror()` function shall not return a value.

45030 **ERRORS**

45031 No errors are defined.

45032 **EXAMPLES**

45033 **Printing an Error Message for a Function**

45034 The following example replaces `bufptr` with a buffer that is the necessary size. If an error occurs,
 45035 the `perror()` function prints a message and the program exits.

```

45036 #include <stdio.h>
45037 #include <stdlib.h>
45038 ...
45039 char *bufptr;
45040 size_t szbuf;
45041 ...
45042 if ((bufptr = malloc(szbuf)) == NULL) {
45043     perror("malloc"); exit(2);
45044 }
45045 ...
  
```

perror()

45046 **APPLICATION USAGE**
 45047 None.

45048 **RATIONALE**
 45049 None.

45050 **FUTURE DIRECTIONS**
 45051 None.

45052 **SEE ALSO**
 45053 *psiginfo()*, *strerror()*
 45054 XBD <stdio.h>

45055 **CHANGE HISTORY**
 45056 First released in Issue 1. Derived from Issue 1 of the SVID.

45057 **Issue 5**
 45058 A paragraph is added to the DESCRIPTION indicating that *perror()* does not change the
 45059 orientation of the standard error stream.

45060 **Issue 6**
 45061 Extensions beyond the ISO C standard are marked.

45062 **Issue 7**
 45063 SD5-XSH-ERN-95 is applied.
 45064 Changes are made related to support for finegrained timestamps.

45065 **NAME**

45066 pipe — create an interprocess channel

45067 **SYNOPSIS**

45068 #include <unistd.h>

45069 int pipe(int *fildes*[2]);45070 **DESCRIPTION**

45071 The *pipe()* function shall create a pipe and place two file descriptors, one each into the
 45072 arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write
 45073 ends of the pipe. Their integer values shall be the two lowest available at the time of the *pipe()*
 45074 call. The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The
 45075 *fcntl()* function can be used to set both these flags.)

45076 Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A
 45077 read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a
 45078 first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether
 45079 *fildes*[1] is also open for reading.

45080 A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open
 45081 that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

45082 The pipe's user ID shall be set to the effective user ID of the calling process.

45083 The pipe's group ID shall be set to the effective group ID of the calling process.

45084 Upon successful completion, *pipe()* shall mark for update the last data access, last data
 45085 modification, and last file status change timestamps of the pipe.

45086 **RETURN VALUE**

45087 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 45088 indicate the error.

45089 **ERRORS**

45090 The *pipe()* function shall fail if:

45091 [EMFILE] All, or all but one, of the file descriptors available to the process are currently
 45092 open.

45093 [ENFILE] The number of simultaneously open files in the system would exceed a
 45094 system-imposed limit.

45095 **EXAMPLES**45096 **Using a Pipe to Pass Data Between a Parent Process and a Child Process**

45097 The following example demonstrates the use of a pipe to transfer data between a parent process
 45098 and a child process. Error handling is excluded, but otherwise this code demonstrates good
 45099 practice when using pipes: after the *fork()* the two processes close the unused ends of the pipe
 45100 before they commence transferring data.

```
45101 #include <stdlib.h>
45102 #include <unistd.h>
45103 ...
45104 int fildes[2];
45105 const int BSIZE = 100;
45106 char buf[BSIZE];
45107 ssize_t nbytes;
```

```

45108     int status;
45109
45109     status = pipe(fildes);
45110     if (status == -1 ) {
45111         /* an error occurred */
45112         ...
45113     }
45114
45114     switch (fork()) {
45115     case -1: /* Handle error */
45116         break;
45117
45117     case 0: /* Child - reads from pipe */
45118         close(fildes[1]); /* Write end is unused */
45119         nbytes = read(fildes[0], buf, BSIZE); /* Get data from pipe */
45120         /* At this point, a further read would see end of file ... */
45121         close(fildes[0]); /* Finished with pipe */
45122         exit(EXIT_SUCCESS);
45123
45123     default: /* Parent - writes to pipe */
45124         close(fildes[0]); /* Read end is unused */
45125         write(fildes[1], "Hello world\n", 12); /* Write data on pipe */
45126         close(fildes[1]); /* Child will see EOF */
45127         exit(EXIT_SUCCESS);
45128     }

```

APPLICATION USAGE

None.

RATIONALE

The wording carefully avoids using the verb “to open” in order to avoid any implication of use of *open()*; see also *write*.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl(), *read*, *write*

XBD [<fcntl.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate that certain dispositions of *fildes[0]* and *fildes[1]* are unspecified.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/65 is applied, adding the example to the EXAMPLES section.

Issue 7

SD5-XSH-ERN-156 is applied, updating the DESCRIPTION to state the setting of the pipe’s user ID and group ID.

Changes are made related to support for finegrained timestamps.

45152 **NAME**

45153 poll — input/output multiplexing

45154 **SYNOPSIS**

45155 #include <poll.h>

45156 int poll(struct pollfd *fds*[], nfds_t *nfds*, int *timeout*);45157 **DESCRIPTION**

45158 The *poll()* function provides applications with a mechanism for multiplexing input/output over
 45159 a set of file descriptors. For each member of the array pointed to by *fds*, *poll()* shall examine the
 45160 given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the
 45161 *fds* array is specified by *nfds*. The *poll()* function shall identify those file descriptors on which an
 45162 application can read or write data, or on which certain events have occurred.

45163 The *fds* argument specifies the file descriptors to be examined and the events of interest for each
 45164 file descriptor. It is a pointer to an array with one member for each open file descriptor of
 45165 interest. The array's members are **pollfd** structures within which *fd* specifies an open file
 45166 descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the
 45167 following event flags:

45168	POLLIN	Data other than high-priority data may be read without blocking.
45169	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
45170		This flag shall be equivalent to POLLRDNORM POLLRDBAND.
45171	POLLRDNORM	Normal data may be read without blocking.
45172	OB XSR	For STREAMS, data on priority band 0 may be read without blocking. This
45173		flag is set in <i>revents</i> even if the message is of zero length.
45174	POLLRDBAND	Priority data may be read without blocking.
45175	OB XSR	For STREAMS, data on priority bands greater than 0 may be read without
45176		blocking. This flag is set in <i>revents</i> even if the message is of zero length.
45177	POLLPRI	High-priority data may be read without blocking.
45178	OB XSR	For STREAMS, this flag is set in <i>revents</i> even if the message is of zero length.
45179	POLLOUT	Normal data may be written without blocking.
45180	OB XSR	For STREAMS, data on priority band 0 may be written without blocking.
45181	POLLWRNORM	Equivalent to POLLOUT.
45182	POLLWRBAND	Priority data may be written.
45183	OB XSR	For STREAMS, data on priority bands greater than 0 may be written without
45184		blocking. If any priority band has been written to on this STREAM, this event
45185		only examines bands that have been written to at least once.
45186	POLLERR	An error has occurred on the device or stream. This flag is only valid in the
45187		<i>revents</i> bitmask; it shall be ignored in the <i>events</i> member.
45188	POLLHUP	The device has been disconnected. This event and POLLOUT are mutually-
45189		exclusive; a stream can never be writable if a hangup has occurred. However,
45190		this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are
45191		not mutually-exclusive. This flag is only valid in the <i>revents</i> bitmask; it shall be
45192		ignored in the <i>events</i> member.

poll()

- 45193 POLLNVAL The specified *fd* value is invalid. This flag is only valid in the *revents* member;
45194 it shall ignored in the *events* member.
- 45195 The significance and semantics of normal, priority, and high-priority data are file and device-
45196 specific.
- 45197 If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on
45198 return from *poll()*.
- 45199 In each **pollfd** structure, *poll()* shall clear the *revents* member, except that where the application
45200 requested a report on a condition by setting one of the bits of *events* listed above, *poll()* shall set
45201 the corresponding bit in *revents* if the requested condition is true. In addition, *poll()* shall set the
45202 POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the
45203 application did not set the corresponding bit in *events*.
- 45204 If none of the defined events have occurred on any selected file descriptor, *poll()* shall wait at
45205 least *timeout* milliseconds for an event to occur on any of the selected file descriptors. If the value
45206 of *timeout* is 0, *poll()* shall return immediately. If the value of *timeout* is -1, *poll()* shall block until
45207 a requested event occurs or until the call is interrupted.
- 45208 Implementations may place limitations on the granularity of timeout intervals. If the requested
45209 timeout interval requires a finer granularity than the implementation supports, the actual
45210 timeout interval shall be rounded up to the next supported value.
- 45211 The *poll()* function shall not be affected by the O_NONBLOCK flag.
- 45212 The *poll()* function shall support regular files, terminal and pseudo-terminal devices, FIFOs,
45213 OB XSR pipes, sockets and STREAMS-based files. The behavior of *poll()* on elements of *fds* that refer to
45214 other types of file is unspecified.
- 45215 Regular files shall always poll TRUE for reading and writing.
- 45216 A file descriptor for a socket that is listening for connections shall indicate that it is ready for
45217 reading, once connections are available. A file descriptor for a socket that is connecting
45218 asynchronously shall indicate that it is ready for writing, once a connection has been established.

RETURN VALUE

- 45219 Upon successful completion, *poll()* shall return a non-negative value. A positive value indicates
45220 the total number of file descriptors that have been selected (that is, file descriptors for which the
45221 *revents* member is non-zero). A value of 0 indicates that the call timed out and no file descriptors
45222 have been selected. Upon failure, *poll()* shall return -1 and set *errno* to indicate the error.

ERRORS

- 45224 The *poll()* function shall fail if:
- 45225 [EAGAIN] The allocation of internal data structures failed but a subsequent request may
45226 succeed.
- 45227 [EINTR] A signal was caught during *poll()*.
- 45228 [EINVAL] OB XSR The *nfds* argument is greater than {OPEN_MAX}, or one of the *fd* members
45229 refers to a STREAM or multiplexer that is linked (directly or indirectly)
45230 downstream from a multiplexer.
45231

EXAMPLES

Checking for Events on a Stream

The following example opens a pair of STREAMS devices and then waits for either one to become writable. This example proceeds as follows:

1. Sets the *timeout* parameter to 500 milliseconds.
2. Opens the STREAMS devices */dev/dev0* and */dev/dev1*, and then polls them, specifying POLLOUT and POLLWRBAND as the events of interest.

The STREAMS device names */dev/dev0* and */dev/dev1* are only examples of how STREAMS devices can be named; STREAMS naming conventions may vary among systems conforming to the POSIX.1-200x.

3. Uses the *ret* variable to determine whether an event has occurred on either of the two STREAMS. The *poll()* function is given 500 milliseconds to wait for an event to occur (if it has not occurred prior to the *poll()* call).
4. Checks the returned value of *ret*. If a positive value is returned, one of the following can be done:
 - a. Priority data can be written to the open STREAM on priority bands greater than 0, because the POLLWRBAND event occurred on the open STREAM (*fds[0]* or *fds[1]*).
 - b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT event occurred on the open STREAM (*fds[0]* or *fds[1]*).
5. If the returned value is not a positive value, permission to write data to the open STREAM (on any priority band) is denied.
6. If the POLLHUP event occurs on the open STREAM (*fds[0]* or *fds[1]*), the device on the open STREAM has disconnected.

```

45255 #include <stropts.h>
45256 #include <poll.h>
45257 ...
45258 struct pollfd fds[2];
45259 int timeout_msecs = 500;
45260 int ret;
45261 int i;

45262 /* Open STREAMS device. */
45263 fds[0].fd = open("/dev/dev0", ...);
45264 fds[1].fd = open("/dev/dev1", ...);
45265 fds[0].events = POLLOUT | POLLWRBAND;
45266 fds[1].events = POLLOUT | POLLWRBAND;

45267 ret = poll(fds, 2, timeout_msecs);

45268 if (ret > 0) {
45269     /* An event on one of the fds has occurred. */
45270     for (i=0; i<2; i++) {
45271         if (fds[i].revents & POLLWRBAND) {
45272             /* Priority data may be written on device number i. */
45273             ...
45274         }
45275         if (fds[i].revents & POLLOUT) {
45276             /* Data may be written on device number i. */
45277             ...

```

```

45278         }
45279         if (fds[i].revents & POLLHUP) {
45280             /* A hangup has occurred on device number i. */
45281             ...
45282         }
45283     }
45284 }

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6 (on page 473), *getmsg()*, *pselect()*, *putmsg()*, *read*, *write*, the Base Definitions volume of POSIX.1-200x, `<poll.h>`, `<stropts.h>`

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

The description of POLLWRBAND is updated.

Issue 6

Text referring to sockets is added to the DESCRIPTION.

Functionality relating to the XSI STREAMS Option Group is marked.

The Open Group Corrigendum U055/3 is applied, updating the DESCRIPTION of POLLWRBAND.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/66 is applied, correcting the spacing in the EXAMPLES section.

Issue 7

The *poll()* function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

45309 **NAME**
 45310 popen — initiate pipe streams to or from a process

45311 **SYNOPSIS**

```
45312 CX #include <stdio.h>
45313 FILE *popen(const char *command, const char *mode);
```

45314 **DESCRIPTION**

45315 The *popen()* function shall execute the command specified by the string *command*. It shall create
 45316 a pipe between the calling program and the executed command, and shall return a pointer to a
 45317 stream that can be used to either read from or write to the pipe.

45318 The environment of the executed command shall be as if a child process were created within the
 45319 *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

```
45320 execl(shell_path, "sh", "-c", command, (char *)0);
```

45321 where *shell_path* is an unspecified pathname for the *sh* utility.

45322 The *popen()* function shall ensure that any streams from previous *popen()* calls that remain open
 45323 in the parent process are closed in the new child process.

45324 The *mode* argument to *popen()* is a string that specifies I/O mode:

- 45325 1. If *mode* is *r*, when the child process is started, its file descriptor `STDOUT_FILENO` shall be
 45326 the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 45327 where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the
 45328 pipe.
- 45329 2. If *mode* is *w*, when the child process is started its file descriptor `STDIN_FILENO` shall be
 45330 the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process,
 45331 where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the
 45332 pipe.
- 45333 3. If *mode* is any other value, the result is unspecified.

45334 After *popen()*, both the parent and the child process shall be capable of executing independently
 45335 before either terminates.

45336 Pipe streams are byte-oriented.

45337 **RETURN VALUE**

45338 Upon successful completion, *popen()* shall return a pointer to an open stream that can be used to
 45339 read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate
 45340 the error.

45341 **ERRORS**

45342 The *popen()* function shall fail if: +

45343 [EMFILE] {STREAM_MAX} streams are currently open in the calling process. +

45344 The *popen()* function may fail if:

45345 [EMFILE] {FOPEN_MAX} streams are currently open in the calling process. -

45346 [EINVAL] The *mode* argument is invalid.

45347 The *popen()* function may also set *errno* values as described by *fork()* or *pipe()*.

EXAMPLES

Using popen() to Obtain a List of Files from the ls Utility

The following example demonstrates the use of *popen()* and *pclose()* to execute the command *ls** in order to obtain a list of files in the current directory:

```

45352 #include <stdio.h>
45353 ...
45354 FILE *fp;
45355 int status;
45356 char path[PATH_MAX];
45357 fp = popen("ls *", "r");
45358 if (fp == NULL)
45359     /* Handle error */;
45360 while (fgets(path, PATH_MAX, fp) != NULL)
45361     printf("%s", path);
45362 status = pclose(fp);
45363 if (status == -1) {
45364     /* Error reported by pclose() */
45365     ...
45366 } else {
45367     /* Use macros described under wait() to inspect 'status' in order
45368        to determine success/failure of command executed by popen() */
45369     ...
45370 }

```

APPLICATION USAGE

Since open files are shared, a mode *r* command can be used as an input filter and a mode *w* command as an output filter.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be prevented by careful buffer flushing; for example, with *fflush()*.

A stream opened by *popen()* should be closed by *pclose()*.

The behavior of *popen()* is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and *wb* might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen()* only check to see if the first character of *mode* is *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be treated as *mode w*.

If the application calls *waitpid()* or *waitid()* with a *pid* argument greater than 0, and it still has a stream that was called with *popen()* open, it must ensure that *pid* does not refer to the process started by *popen()*.

To determine whether or not the environment specified in the Shell and Utilities volume of POSIX.1-200x is present, use the function call:

```
sysconf(_SC_2_VERSION)
```

(See *sysconf()*).

45390
45391
45392
45393
45394

45395
45396
45397

45398
45399

45400
45401
45402
45403

45404
45405

45406
45407

45408
45409
45410
45411
45412
45413

45414
45415
45416

45417
45418

RATIONALE

The *popen()* function should not be used by programs that have set user (or group) ID privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

If the original and *popen()*ed processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions (*fread()*, *fwrite()*, and so on), the rules for sharing file handles must be observed (see [Section 2.5.1](#), on page 470).

FUTURE DIRECTIONS

None.

SEE ALSO

fork(), *pclose()*, *pipe()*, *sysconf()*, *system()*, *wait*, *waitid()*

XBD [<stdio.h>](#)

XCU *sh*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

A statement is added to the DESCRIPTION indicating that pipe streams are byte-oriented.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The optional [EMFILE] error condition is added.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/67 is applied, adding the example to the EXAMPLES section.

Issue 7

Austin Group Interpretation 1003.1-2001 #029 is applied, clarifying the values for *mode* in the DESCRIPTION.

SD5-XSH-ERN-149 is applied, changing the {STREAM_MAX} [EMFILE] error condition from a "may fail" to a "shall fail".

45419 **NAME**45420 posix_fadvise — file advisory information (**ADVANCED REALTIME**)45421 **SYNOPSIS**

```
45422 ADV #include <fcntl.h>
45423 int posix_fadvise(int fd, off_t offset, off_t len, int advice);
```

45424 **DESCRIPTION**

45425 The *posix_fadvise()* function shall advise the implementation on the expected behavior of the
 45426 application with respect to the data in the file associated with the open file descriptor, *fd*, starting
 45427 at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len*
 45428 is zero, all data following *offset* is specified. The implementation may use this information to
 45429 optimize handling of the specified data. The *posix_fadvise()* function shall have no effect on the
 45430 semantics of other operations on the specified data, although it may affect the performance of
 45431 other operations.

45432 The advice to be applied to the data is specified by the *advice* parameter and may be one of the
 45433 following values:

45434 **POSIX_FADV_NORMAL**

45435 Specifies that the application has no advice to give on its behavior with respect to the
 45436 specified data. It is the default characteristic if no advice is given for an open file.

45437 **POSIX_FADV_SEQUENTIAL**

45438 Specifies that the application expects to access the specified data sequentially from lower
 45439 offsets to higher offsets.

45440 **POSIX_FADV_RANDOM**

45441 Specifies that the application expects to access the specified data in a random order.

45442 **POSIX_FADV_WILLNEED**

45443 Specifies that the application expects to access the specified data in the near future.

45444 **POSIX_FADV_DONTNEED**

45445 Specifies that the application expects that it will not access the specified data in the near
 45446 future.

45447 **POSIX_FADV_NOREUSE**

45448 Specifies that the application expects to access the specified data once and then not reuse it
 45449 thereafter.

45450 These values are defined in **<fcntl.h>**.

45451 **RETURN VALUE**

45452 Upon successful completion, *posix_fadvise()* shall return zero; otherwise, an error number shall
 45453 be returned to indicate the error.

45454 **ERRORS**

45455 The *posix_fadvise()* function shall fail if:

- | | | |
|-------|----------|--|
| 45456 | [EBADF] | The <i>fd</i> argument is not a valid file descriptor. |
| 45457 | [EINVAL] | The value of <i>advice</i> is invalid, or the value of <i>len</i> is less than zero. |
| 45458 | [ESPIPE] | The <i>fd</i> argument is associated with a pipe or FIFO. |

45459 **EXAMPLES**

45460 None.

45461 **APPLICATION USAGE**45462 The *posix_fadvise()* function is part of the Advisory Information option and need not be
45463 provided on all implementations.45464 **RATIONALE**

45465 None.

45466 **FUTURE DIRECTIONS**

45467 None.

45468 **SEE ALSO**45469 *posix_madvise()*

45470 XBD <fcntl.h>

45471 **CHANGE HISTORY**

45472 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

45473 In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

45474 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/68 is applied, changing the function
45475 prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the
45476 standard developers felt it acceptable to make this change before implementations of this
45477 function become widespread.45478 **Issue 7**45479 Austin Group Interpretation 1003.1-2001 #024 is applied, changing the definition of the
45480 [EINVAL] error.

45481 **NAME**45482 posix_fallocate — file space control (**ADVANCED REALTIME**)45483 **SYNOPSIS**

45484 ADV #include <fcntl.h>

45485 int posix_fallocate(int *fd*, off_t *offset*, off_t *len*);45486 **DESCRIPTION**

45487 The *posix_fallocate()* function shall ensure that any required storage for regular file data starting
 45488 at *offset* and continuing for *len* bytes is allocated on the file system storage media. If
 45489 *posix_fallocate()* returns successfully, subsequent writes to the specified file data shall not fail due
 45490 to the lack of free space on the file system storage media.

45491 If the *offset+len* is beyond the current file size, then *posix_fallocate()* shall adjust the file size to
 45492 *offset+len*. Otherwise, the file size shall not be changed.

45493 It is implementation-defined whether a previous *posix_fadvise()* call influences allocation
 45494 strategy.

45495 Space allocated via *posix_fallocate()* shall be freed by a successful call to *creat()* or *open()* that
 45496 truncates the size of the file. Space allocated via *posix_fallocate()* may be freed by a successful call
 45497 to *truncate()* that reduces the file size to a size smaller than *offset+len*.

45498 **RETURN VALUE**

45499 Upon successful completion, *posix_fallocate()* shall return zero; otherwise, an error number shall
 45500 be returned to indicate the error.

45501 **ERRORS**45502 The *posix_fallocate()* function shall fail if:

- | | | |
|-------|----------|--|
| 45503 | [EBADF] | The <i>fd</i> argument is not a valid file descriptor. |
| 45504 | [EBADF] | The <i>fd</i> argument references a file that was opened without write permission. |
| 45505 | [EFBIG] | The value of <i>offset+len</i> is greater than the maximum file size. |
| 45506 | [EINTR] | A signal was caught during execution. |
| 45507 | [EINVAL] | The <i>len</i> argument is less than or equal to zero, or the <i>offset</i> argument is less than zero, or the underlying file system does not support this operation. |
| 45508 | | |
| 45509 | [EIO] | An I/O error occurred while reading from or writing to a file system. |
| 45510 | [ENODEV] | The <i>fd</i> argument does not refer to a regular file. |
| 45511 | [ENOSPC] | There is insufficient free space remaining on the file system storage media. |
| 45512 | [ESPIPE] | The <i>fd</i> argument is associated with a pipe or FIFO. |

45513 **EXAMPLES**

45514 None.

45515 **APPLICATION USAGE**

45516 The *posix_fallocate()* function is part of the Advisory Information option and need not be
 45517 provided on all implementations.

45518
45519
45520
45521
45522
45523
45524
45525
45526
45527
45528
45529
45530
45531
45532
45533
45534

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

creat(), *ftruncate()*, *open()*, *unlink*

XBD <fcntl.h>

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

In the SYNOPSIS, the inclusion of <sys/types.h> is no longer required.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/69 is applied, changing the function prototype in the SYNOPSIS section. The previous prototype was not large file-aware, and the standard developers felt it acceptable to make this change before implementations of this function become widespread.

Issue 7

Austin Group Interpretations 1003.1-2001 #022 and #024 are applied, changing the definition of the [EINVAL] error.

DRAFT

45535 **NAME**

45536 `posix_madvise` — memory advisory information and alignment control (**ADVANCED**
 45537 **REALTIME**)

45538 **SYNOPSIS**

```
45539 ADV #include <sys/mman.h>
45540 int posix_madvise(void *addr, size_t len, int advice);
```

45541 **DESCRIPTION**

45542 The `posix_madvise()` function shall advise the implementation on the expected behavior of the
 45543 application with respect to the data in the memory starting at address `addr`, and continuing for
 45544 `len` bytes. The implementation may use this information to optimize handling of the specified
 45545 data. The `posix_madvise()` function shall have no effect on the semantics of access to memory in
 45546 the specified range, although it may affect the performance of access.

45547 The implementation may require that `addr` be a multiple of the page size, which is the value
 45548 returned by `sysconf()` when the name value `_SC_PAGESIZE` is used.

45549 The advice to be applied to the memory range is specified by the `advice` parameter and may be
 45550 one of the following values:

45551 **POSIX_MADV_NORMAL**

45552 Specifies that the application has no advice to give on its behavior with respect to the
 45553 specified range. It is the default characteristic if no advice is given for a range of memory.

45554 **POSIX_MADV_SEQUENTIAL**

45555 Specifies that the application expects to access the specified range sequentially from lower
 45556 addresses to higher addresses.

45557 **POSIX_MADV_RANDOM**

45558 Specifies that the application expects to access the specified range in a random order.

45559 **POSIX_MADV_WILLNEED**

45560 Specifies that the application expects to access the specified range in the near future.

45561 **POSIX_MADV_DONTNEED**

45562 Specifies that the application expects that it will not access the specified range in the near
 45563 future.

45564 These values are defined in the `<sys/mman.h>` header.

45565 **RETURN VALUE**

45566 Upon successful completion, `posix_madvise()` shall return zero; otherwise, an error number shall
 45567 be returned to indicate the error.

45568 **ERRORS**

45569 The `posix_madvise()` function shall fail if:

45570 [EINVAL] The value of `advice` is invalid.

45571 [ENOMEM] Addresses in the range starting at `addr` and continuing for `len` bytes are partly
 45572 or completely outside the range allowed for the address space of the calling
 45573 process.

45574 The *posix_madvise()* function may fail if:

45575 [EINVAL] The value of *addr* is not a multiple of the value returned by *sysconf()* when the
45576 name value *_SC_PAGESIZE* is used.

45577 [EINVAL] The value of *len* is zero.

EXAMPLES

45578 None.
45579

APPLICATION USAGE

45580 The *posix_madvise()* function is part of the Advisory Information option and need not be
45581 provided on all implementations.
45582

RATIONALE

45583 None.
45584

FUTURE DIRECTIONS

45585 None.
45586

SEE ALSO

45587 *mmap()*, *posix_fadvise()*, *sysconf()*
45588

45589 XBD <[sys/mman.h](#)>

CHANGE HISTORY

45590 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
45591

45592 IEEE PASC Interpretation 1003.1 #102 is applied.

45593 **NAME**

45594 **posix_mem_offset** — find offset and length of a mapped typed memory block (**ADVANCED**
 45595 **REALTIME**)

45596 **SYNOPSIS**

```
45597 TYM #include <sys/mman.h>
45598
45598 int posix_mem_offset(const void *restrict addr, size_t len,
45599 off_t *restrict off, size_t *restrict contig_len,
45600 int *restrict fildes);
```

45601 **DESCRIPTION**

45602 The *posix_mem_offset()* function shall return in the variable pointed to by *off* a value that
 45603 identifies the offset (or location), within a memory object, of the memory block currently
 45604 mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used
 45605 (via *mmap()*) to establish the mapping which contains *addr*. If that descriptor was closed since
 45606 the mapping was established, the returned value of *fildes* shall be -1 . The *len* argument specifies
 45607 the length of the block of the memory object the user wishes the offset for; upon return, the
 45608 value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block
 45609 of the memory object that is currently mapped to the calling process starting at *addr*, whichever
 45610 is smaller.

45611 If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len*
 45612 values obtained by calling *posix_mem_offset()* are used in a call to *mmap()* with a file descriptor
 45613 that refers to the same memory pool as *fildes* (either through the same port or through a different
 45614 port), and that was opened with neither the `POSIX_TYPED_MEM_ALLOCATE` nor the
 45615 `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag, the typed memory area that is mapped shall
 45616 be exactly the same area that was mapped at *addr* in the address space of the process that called
 45617 *posix_mem_offset()*.

45618 If the memory object specified by *fildes* is not a typed memory object, then the behavior of this
 45619 function is implementation-defined.

45620 **RETURN VALUE**

45621 Upon successful completion, the *posix_mem_offset()* function shall return zero; otherwise, the
 45622 corresponding error status value shall be returned.

45623 **ERRORS**

45624 The *posix_mem_offset()* function shall fail if:

45625 [EACCES] The process has not mapped a memory object supported by this function at
 45626 the given address *addr*.

45627 This function shall not return an error code of [EINTR].

45628 **EXAMPLES**

45629 None.

45630 **APPLICATION USAGE**

45631 None.

45632 **RATIONALE**

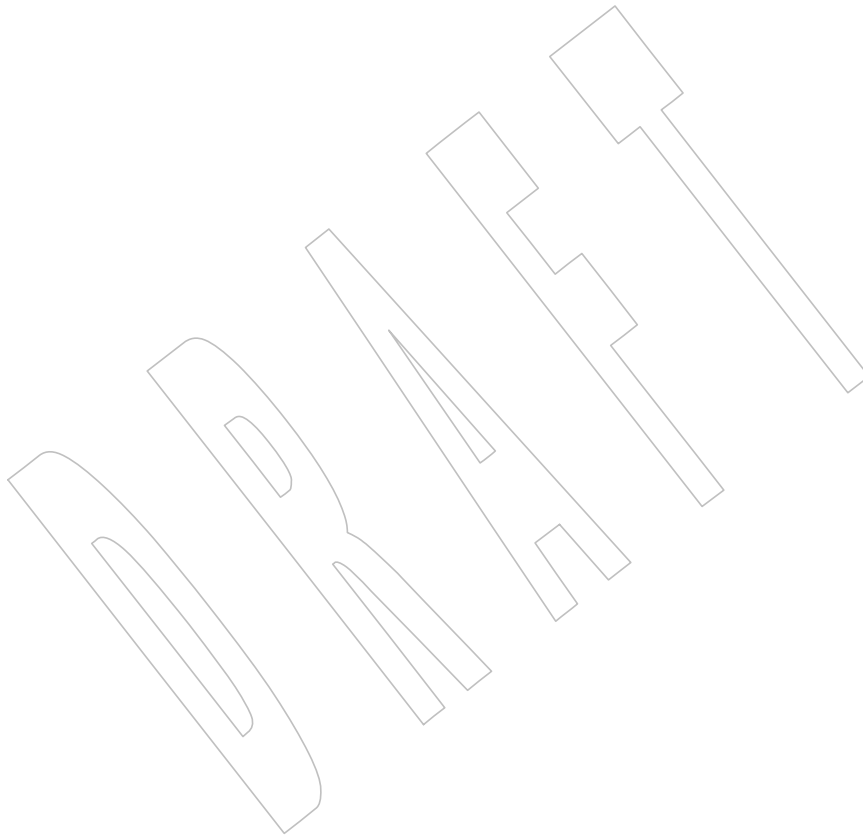
45633 None.

45634
45635
45636
45637
45638
45639
45640**FUTURE DIRECTIONS**

None.

SEE ALSO*mmap()*, *posix_typed_mem_open()*XBD <[sys/mman.h](#)>**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.



45641 **NAME**45642 posix_memalign — aligned memory allocation (**ADVANCED REALTIME**)45643 **SYNOPSIS**

```
45644 ADV #include <stdlib.h>
45645 int posix_memalign(void **memptr, size_t alignment, size_t size);
```

45646 **DESCRIPTION**

45647 The *posix_memalign()* function shall allocate *size* bytes aligned on a boundary specified by
 45648 *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment*
 45649 shall be a power of two multiple of *sizeof(void *)*. Upon successful completion, the value
 45650 pointed to by *memptr* shall be a multiple of *alignment*.

45651 CX The *free()* function shall deallocate memory that has previously been allocated by
 45652 *posix_memalign()*.

45653 **RETURN VALUE**

45654 Upon successful completion, *posix_memalign()* shall return zero; otherwise, an error number
 45655 shall be returned to indicate the error.

45656 **ERRORS**

45657 The *posix_memalign()* function shall fail if:

- | | | |
|-------|----------|--|
| 45658 | [EINVAL] | The value of the alignment parameter is not a power of two multiple of <i>sizeof(void *)</i> . |
| 45659 | | |
| 45660 | [ENOMEM] | There is insufficient memory available with the requested alignment. |

45661 **EXAMPLES**

45662 None.

45663 **APPLICATION USAGE**

45664 The *posix_memalign()* function is part of the Advisory Information option and need not be
 45665 provided on all implementations.

45666 **RATIONALE**

45667 None.

45668 **FUTURE DIRECTIONS**

45669 None.

45670 **SEE ALSO**

45671 *free()*, *malloc()*

45672 XBD [<stdlib.h>](#)

45673 **CHANGE HISTORY**

45674 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

45675 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

45676 **Issue 7**

45677 Austin Group Interpretation 1003.1-2001 #058 is applied, clarifying the value of the *alignment*
 45678 argument in the DESCRIPTION.

45679 **NAME**
 45680 posix_openpt — open a pseudo-terminal device

45681 **SYNOPSIS**

```
45682 XSI #include <stdlib.h>
45683 #include <fcntl.h>
45684 int posix_openpt(int oflag);
```

45685 **DESCRIPTION**

45686 The *posix_openpt()* function shall establish a connection between a master device for a pseudo-
 45687 terminal and a file descriptor. The file descriptor is used by other I/O functions that refer to that
 45688 pseudo-terminal.

45689 The file status flags and file access modes of the open file description shall be set according to
 45690 the value of *oflag*.

45691 Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined
 45692 in **<fcntl.h>**:

45693 O_RDWR Open for reading and writing.
 45694 O_NOCTTY If set *posix_openpt()* shall not cause the terminal device to become the
 45695 controlling terminal for the process.

45696 The behavior of other values for the *oflag* argument is unspecified.

45697 **RETURN VALUE**

45698 Upon successful completion, the *posix_openpt()* function shall open a master pseudo-terminal
 45699 device and return a non-negative integer representing the lowest numbered unused file
 45700 descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error.

45701 **ERRORS**

45702 The *posix_openpt()* function shall fail if:

45703 [EMFILE] All file descriptors available to the process are currently open.
 45704 [ENFILE] The maximum allowable number of files is currently open in the system.

45705 The *posix_openpt()* function may fail if:

45706 [EINVAL] The value of *oflag* is not valid.
 45707 [EAGAIN] Out of pseudo-terminal resources.

45708 OB XSR [ENOSR] Out of STREAMS resources.

45709 **EXAMPLES**

45710 **Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File**
 45711 **Descriptor**

```
45712 #include <fcntl.h>
45713 #include <stdio.h>
45714 int masterfd, slavefd;
45715 char *slavedevice;
45716 masterfd = posix_openpt(O_RDWR|O_NOCTTY);
45717 if (masterfd == -1
```

posix_openpt()

```

45718     || grantpt (masterfd) == -1
45719     || unlockpt (masterfd) == -1
45720     || (slavedevice = ptsname (masterfd)) == NULL)
45721     return -1;

45722     printf("slave device is: %s\n", slavedevice);

45723     slavefd = open(slavedevice, O_RDWR|O_NOCTTY);
45724     if (slavefd < 0)
45725         return -1;

```

APPLICATION USAGE

45726 This function is a method for portably obtaining a file descriptor of a master terminal device for
 45727 a pseudo-terminal. The *grantpt()* and *ptsname()* functions can be used to manipulate mode and
 45728 ownership permissions, and to obtain the name of the slave device, respectively.
 45729

RATIONALE

45730 The standard developers considered the matter of adding a special device for cloning master
 45731 pseudo-terminals: the */dev/ptmx* device. However, consensus could not be reached, and it was
 45732 felt that adding a new function would permit other implementations. The *posix_openpt()*
 45733 function is designed to complement the *grantpt()*, *ptsname()*, and *unlockpt()* functions.
 45734

45735 On implementations supporting the */dev/ptmx* clone device, opening the master device of a
 45736 pseudo-terminal is simply:

```

45737     mfdp = open("/dev/ptmx", oflag );
45738     if (mfdp < 0)
45739         return -1;

```

FUTURE DIRECTIONS

45740 None.
 45741

SEE ALSO

45742 *grantpt()*, *open()*, *ptsname()*, *unlockpt()*

45743 XBD <*fcntl.h*>

CHANGE HISTORY

45744 First released in Issue 6.

Issue 7

45745 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

45746 SD5-XSH-ERN-51 is applied, correcting an error in the EXAMPLES section.
 45747
 45748
 45749

NAME

posix_spawn, posix_spawnnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

```
SPN    #include <spawn.h>

45754    int posix_spawn(pid_t *restrict pid, const char *restrict path,
45755                const posix_spawn_file_actions_t *file_actions,
45756                const posix_spawnattr_t *restrict attrp,
45757                char *const argv[restrict], char *const envp[restrict]);
45758    int posix_spawnnp(pid_t *restrict pid, const char *restrict file,
45759                const posix_spawn_file_actions_t *file_actions,
45760                const posix_spawnattr_t *restrict attrp,
45761                char *const argv[restrict], char *const envp[restrict]);
```

DESCRIPTION

The *posix_spawn()* and *posix_spawnnp()* functions shall create a new process (child process) from the specified process image. The new process image shall be constructed from a regular executable file called the new process image file.

When a C program is executed as the result of this call, it shall be entered as a C-language function call as follows:

```
int main(int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
extern char **environ;
```

shall be initialized as a pointer to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array shall be a null pointer and is not counted in *argc*. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process image being started by the *posix_spawn()* or *posix_spawnnp()* function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The number of bytes available for the combined argument and environment lists of the child process is {ARG_MAX}. The implementation shall specify in the system documentation (see XBD Chapter 2, on page 15) whether any list overhead, such as length words, null terminators, pointers, or alignment bytes, is included in this total.

The *path* argument to *posix_spawn()* is a pathname that identifies the new process image file to execute.

The *file* parameter to *posix_spawnnp()* shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a slash character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable *PATH* (see XBD Chapter 8, on page 159). If this environment variable is not defined, the results of the search are implementation-defined.

If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open

45794 in the child process, except for those whose close-on-exec flag `FD_CLOEXEC` is set (see `fcntl()`).
 45795 For those file descriptors that remain open, all attributes of the corresponding open file
 45796 descriptions, including file locks (see `fcntl()`), shall remain unchanged.

45797 If `file_actions` is not `NULL`, then the file descriptors open in the child process shall be those open
 45798 in the calling process as modified by the spawn file actions object pointed to by `file_actions` and
 45799 the `FD_CLOEXEC` flag of each remaining open file descriptor after the spawn file actions have
 45800 been processed. The effective order of processing the spawn file actions shall be:

- 45801 1. The set of open file descriptors for the child process shall initially be the same set as is
 45802 open for the calling process. All attributes of the corresponding open file descriptions,
 45803 including file locks (see `fcntl()`), shall remain unchanged.
- 45804 2. The signal mask, signal default actions, and the effective user and group IDs for the child
 45805 process shall be changed as specified in the attributes object referenced by `attrp`.
- 45806 3. The file actions specified by the spawn file actions object shall be performed in the order
 45807 in which they were added to the spawn file actions object.
- 45808 4. Any file descriptor that has its `FD_CLOEXEC` flag set (see `fcntl()`) shall be closed.

45809 The `posix_spawnattr_t` spawn attributes object type is defined in `<spawn.h>`. It shall contain at
 45810 least the attributes defined below.

45811 If the `POSIX_SPAWN_SETPGROUP` flag is set in the `spawn_flags` attribute of the object referenced
 45812 by `attrp`, and the `spawn-pgroup` attribute of the same object is non-zero, then the child's process
 45813 group shall be as specified in the `spawn-pgroup` attribute of the object referenced by `attrp`.

45814 As a special case, if the `POSIX_SPAWN_SETPGROUP` flag is set in the `spawn_flags` attribute of
 45815 the object referenced by `attrp`, and the `spawn-pgroup` attribute of the same object is set to zero,
 45816 then the child shall be in a new process group with a process group ID equal to its process ID.

45817 If the `POSIX_SPAWN_SETPGROUP` flag is not set in the `spawn_flags` attribute of the object
 45818 referenced by `attrp`, the new child process shall inherit the parent's process group.

45819 PS If the `POSIX_SPAWN_SETSCHEDPARAM` flag is set in the `spawn_flags` attribute of the object
 45820 referenced by `attrp`, but `POSIX_SPAWN_SETSCHEDULER` is not set, the new process image
 45821 shall initially have the scheduling policy of the calling process with the scheduling parameters
 45822 specified in the `spawn-schedparam` attribute of the object referenced by `attrp`.

45823 If the `POSIX_SPAWN_SETSCHEDULER` flag is set in the `spawn_flags` attribute of the object
 45824 referenced by `attrp` (regardless of the setting of the `POSIX_SPAWN_SETSCHEDPARAM` flag),
 45825 the new process image shall initially have the scheduling policy specified in the `spawn-`
 45826 `schedpolicy` attribute of the object referenced by `attrp` and the scheduling parameters specified in
 45827 the `spawn-schedparam` attribute of the same object.

45828 The `POSIX_SPAWN_RESETPID` flag in the `spawn_flags` attribute of the object referenced by `attrp`
 45829 governs the effective user ID of the child process. If this flag is not set, the child process shall
 45830 inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the
 45831 child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit
 45832 of the new process image file is set, the effective user ID of the child process shall become that
 45833 file's owner ID before the new process image begins execution.

45834 The `POSIX_SPAWN_RESETPID` flag in the `spawn_flags` attribute of the object referenced by `attrp`
 45835 also governs the effective group ID of the child process. If this flag is not set, the child process
 45836 shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID
 45837 of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID
 45838 mode bit of the new process image file is set, the effective group ID of the child process shall
 45839 become that file's group ID before the new process image begins execution.

45840 If the `POSIX_SPAWN_SETSIGMASK` flag is set in the `spawn_flags` attribute of the object

45841 referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-*
 45842 *sigmask* attribute of the object referenced by *attrp*.

45843 If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced
 45844 by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to
 45845 their default actions in the child process. Signals set to the default action in the parent process
 45846 shall be set to the default action in the child process.

45847 Signals set to be caught by the calling process shall be set to the default action in the child
 45848 process.

45849 Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be
 45850 ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag
 45851 being set in the *spawn-flags* attribute of the object referenced by *attrp* and the signals being
 45852 indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

45853 If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the
 45854 SIGCHLD signal is set to be ignored or to the default action in the child process, unless
 45855 otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn-flags*
 45856 attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the
 45857 *spawn-sigdefault* attribute of the object referenced by *attrp*.

45858 If the value of the *attrp* pointer is NULL, then the default values are used.

45859 All process attributes, other than those influenced by the attributes set in the object referenced
 45860 by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall
 45861 appear in the new process image as though *fork()* had been called to create a child process and
 45862 then a member of the *exec* family of functions had been called by the child process to execute the
 45863 new process image.

45864 It is implementation-defined whether the fork handlers are run when *posix_spawn()* or
 45865 *posix_spawnnp()* is called.

45866 **RETURN VALUE**

45867 Upon successful completion, *posix_spawn()* and *posix_spawnnp()* shall return the process ID of the
 45868 child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and
 45869 shall return zero as the function return value. Otherwise, no child process shall be created, the
 45870 value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number
 45871 shall be returned as the function return value to indicate the error. If the *pid* argument is a null
 45872 pointer, the process ID of the child is not returned to the caller.

45873 **ERRORS**

45874 The *posix_spawn()* and *posix_spawnnp()* functions may fail if:

45875 [EINVAL] The value specified by *file_actions* or *attrp* is invalid.

45876 If this error occurs after the calling process successfully returns from the *posix_spawn()* or
 45877 *posix_spawnnp()* function, the child process may exit with exit status 127.

45878 If *posix_spawn()* or *posix_spawnnp()* fail for any of the reasons that would cause *fork()* or one of
 45879 the *exec* family of functions to fail, an error value shall be returned as described by *fork()* and
 45880 *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child
 45881 process shall exit with exit status 127).

45882 If POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by
 45883 *attrp*, and *posix_spawn()* or *posix_spawnnp()* fails while changing the child's process group, an
 45884 error value shall be returned as described by *setpgid()* (or, if the error occurs after the calling
 45885 process successfully returns, the child process shall exit with exit status 127).

45886 PS If POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is not set in
 45887 the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix_spawn()* or *posix_spawnnp()*

45888 fails for any of the reasons that would cause *sched_setparam()* to fail, an error value shall be
 45889 returned as described by *sched_setparam()* (or, if the error occurs after the calling process
 45890 successfully returns, the child process shall exit with exit status 127).

45891 If `POSIX_SPAWN_SETSCHEDULER` is set in the *spawn-flags* attribute of the object referenced by
 45892 *attrp*, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
 45893 *sched_setscheduler()* to fail, an error value shall be returned as described by *sched_setscheduler()*
 45894 (or, if the error occurs after the calling process successfully returns, the child process shall exit
 45895 with exit status 127).

45896 If the *file_actions* argument is not `NULL`, and specifies any *close*, *dup2*, or *open* actions to be
 45897 performed, and if *posix_spawn()* or *posix_spawnnp()* fails for any of the reasons that would cause
 45898 *close()*, *dup2()*, or *open()* to fail, an error value shall be returned as described by *close()*, *dup2()*,
 45899 and *open()*, respectively (or, if the error occurs after the calling process successfully returns, the
 45900 child process shall exit with exit status 127). An open file action may, by itself, result in any of
 45901 the errors described by *close()* or *dup2()*, in addition to those described by *open()*.

EXAMPLES

45902 None.
 45903

APPLICATION USAGE

45904 These functions are part of the Spawn option and need not be provided on all implementations.
 45905

RATIONALE

45906 The *posix_spawn()* function and its close relation *posix_spawnnp()* have been introduced to
 45907 overcome the following perceived difficulties with *fork()*: the *fork()* function is difficult or
 45908 impossible to implement without swapping or dynamic address translation.
 45909

- 45910 • Swapping is generally too slow for a realtime environment.
- 45911 • Dynamic address translation is not available everywhere that POSIX might be useful.
- 45912 • Processes are too useful to simply option out of POSIX whenever it must run without
 45913 address translation or other MMU services.

45914 Thus, POSIX needs process creation and file execution primitives that can be efficiently
 45915 implemented without address translation or other MMU services.

45916 The *posix_spawn()* function is implementable as a library routine, but both *posix_spawn()* and
 45917 *posix_spawnnp()* are designed as kernel operations. Also, although they may be an efficient
 45918 replacement for many *fork()/exec* pairs, their goal is to provide useful process creation
 45919 primitives for systems that have difficulty with *fork()*, not to provide drop-in replacements for
 45920 *fork()/exec*.

45921 This view of the role of *posix_spawn()* and *posix_spawnnp()* influenced the design of their API. It
 45922 does not attempt to provide the full functionality of *fork()/exec* in which arbitrary user-specified
 45923 operations of any sort are permitted between the creation of the child process and the execution
 45924 of the new process image; any attempt to reach that level would need to provide a programming
 45925 language as parameters. Instead, *posix_spawn()* and *posix_spawnnp()* are process creation
 45926 primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package
 45927 *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX
 45928 systems, but with some POSIX-specific additions.

45929 To achieve its coverage goals, *posix_spawn()* and *posix_spawnnp()* have control of six types of
 45930 inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and
 45931 whether each signal ignored in the parent will remain ignored in the child, or be reset to its
 45932 default action in the child.

45933 Control of file descriptors is required to allow an independently written child process image to
 45934 access data streams opened by and even generated or read by the parent process without being
 45935 specifically coded to know which parent files and file descriptors are to be used. Control of the

45936 process group ID is required to control how the job control of the child process relates to that of
45937 the parent.

45938 Control of the signal mask and signal defaulting is sufficient to support the implementation of
45939 *system()*. Although support for *system()* is not explicitly one of the goals for *posix_spawn()* and
45940 *posix_spawnnp()*, it is covered under the “at least 50%” coverage goal.

45941 The intention is that the normal file descriptor inheritance across *fork()*, the subsequent effect of
45942 the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec*
45943 family of functions should fully specify open file inheritance. The implementation need make no
45944 decisions regarding the set of open file descriptors when the child process image begins
45945 execution, those decisions having already been made by the caller and expressed as the set of
45946 open file descriptors and their FD_CLOEXEC flags at the time of the call and the spawn file
45947 actions object specified in the call. We have been assured that in cases where the POSIX
45948 *Start_Process* Ada primitives have been implemented in a library, this method of controlling file
45949 descriptor inheritance may be implemented very easily.

45950 We can identify several problems with *posix_spawn()* and *posix_spawnnp()*, but there does not
45951 appear to be a solution that introduces fewer problems. Environment modification for child
45952 process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the
45953 parent process, and since the parent generally wants to save its context, it is more costly than
45954 similar functionality with *fork()/exec*. It is also complicated to modify the environment of a
45955 multi-threaded process temporarily, since all threads must agree when it is safe for the
45956 environment to be changed. However, this cost is only borne by those invocations of
45957 *posix_spawn()* and *posix_spawnnp()* that use the additional functionality. Since extensive
45958 modifications are not the usual case, and are particularly unlikely in time-critical code, keeping
45959 much of the environment control out of *posix_spawn()* and *posix_spawnnp()* is appropriate design.

45960 The *posix_spawn()* and *posix_spawnnp()* functions do not have all the power of *fork()/exec*. This is
45961 to be expected. The *fork()* function is a wonderfully powerful operation. We do not expect to
45962 duplicate its functionality in a simple, fast function with no special hardware requirements. It is
45963 worth noting that *posix_spawn()* and *posix_spawnnp()* are very similar to the process creation
45964 operations on many operating systems that are not UNIX systems.

45965 Requirements

45966 The requirements for *posix_spawn()* and *posix_spawnnp()* are:

- 45967 • They must be implementable without an MMU or unusual hardware.
- 45968 • They must be compatible with existing POSIX standards.

45969 Additional goals are:

- 45970 • They should be efficiently implementable.
- 45971 • They should be able to replace at least 50% of typical executions of *fork()*.
- 45972 • A system with *posix_spawn()* and *posix_spawnnp()* and without *fork()* should be useful, at
45973 least for realtime applications.
- 45974 • A system with *fork()* and the *exec* family should be able to implement *posix_spawn()* and
45975 *posix_spawnnp()* as library routines.

45976

Two-Syntax

45977

45978

45979

45980

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix_spawn**() functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix_spawn**() functions.

45981

45982

The parameter list does not differ between *posix_spawn*() and *posix_spawnp*(); *posix_spawnp*() interprets the second parameter more elaborately than *posix_spawn*() .

45983

Compatibility with POSIX.5 (Ada)

45984

45985

45986

45987

45988

45989

45990

45991

45992

45993

45994

45995

45996

45997

45998

The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork*() and *exec* functionality in a manner similar to that of *posix_spawn*() and *posix_spawnp*() . Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix_spawn*() and *posix_spawnp*() to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain non-default capabilities were not supported. However, based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix_spawn*() and *posix_spawnp*() should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix_spawn*() and *posix_spawnp*() that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

45999

46000

46001

46002

46003

46004

For the Ada language binding for *Start_Process* to be implemented with *posix_spawn*() , that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix_spawn*() whenever the caller of *Start_Process* allowed these arguments to default, since *posix_spawn*() does not provide such defaults. The ability of *Start_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix_spawn*() .

46005

Process Group

46006

46007

46008

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid*() mechanism will place the child process in a new process group.

46009

Threads

46010

46011

46012

46013

46014

46015

46016

46017

46018

Without the *posix_spawn*() and *posix_spawnp*() functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix_spawn*() and *posix_spawnp*() functions are considerably "heavier" than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

46019

46020

46021

46022

46023

The *posix_spawn*() and *posix_spawnp*() functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among "tiny kernels". There should, therefore, probably

46024 continue to be AEPs for operating systems with only one process.

46025 **Asynchronous Error Notification**

46026 A library implementation of *posix_spawn()* or *posix_spawnp()* may not be able to detect all
46027 possible errors before it forks the child process. POSIX.1-200x provides for an error indication
46028 returned from a child process which could not successfully complete the spawn operation via a
46029 special exit status which may be detected using the status value returned by *wait()*, *waitid()*, and
46030 *waitpid()*.

46031 The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of
46032 returning API errors, but they are the only path available to a library implementation. Thus, an
46033 implementation may cause the child process to exit with exit status 127 for any error detected
46034 during the spawn process after the *posix_spawn()* or *posix_spawnp()* function has successfully
46035 returned.

46036 The standard developers had proposed using two additional macros to interpret *stat_val*. The
46037 first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because
46038 of an error detected during the *posix_spawn()* or *posix_spawnp()* operations rather than during
46039 actual execution of the child process image; the second, WSPAWNERRNO, would have
46040 extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group
46041 strongly opposed this because it would make a library implementation of *posix_spawn()* or
46042 *posix_spawnp()* dependent on kernel modifications to *waitpid()* to be able to embed special
46043 information in *stat_val* to indicate a spawn failure.

46044 The 8 bits of child process exit status that are guaranteed by POSIX.1-200x to be accessible to the
46045 waiting parent process are insufficient to disambiguate a spawn error from any other kind of
46046 error that may be returned by an arbitrary process image. No other bits of the exit status are
46047 required to be visible in *stat_val*, so these macros could not be strictly implemented at the library
46048 level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this
46049 value by *system()* and *popen()* to signal failures in these operations that occur after the function
46050 has returned but before a shell is able to execute. The exit status of 127 does not uniquely
46051 identify this class of error, nor does it provide any detailed information on the nature of the
46052 failure. Note that a kernel implementation of *posix_spawn()* or *posix_spawnp()* is permitted (and
46053 encouraged) to return any possible error as the function value, thus providing more detailed
46054 failure information to the parent process.

46055 Thus, no special macros are available to isolate asynchronous *posix_spawn()* or *posix_spawnp()*
46056 errors. Instead, errors detected by the *posix_spawn()* or *posix_spawnp()* operations in the context
46057 of the child process before the new process image executes are reported by setting the child's exit
46058 status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the
46059 *stat_val* stored by the *wait()* or *waitpid()* functions to detect spawn failures to the extent that
46060 other status values with which the child process image may exit (before the parent can
46061 conclusively determine that the child process image has begun execution) are distinct from exit
46062 status 127.

46063 **FUTURE DIRECTIONS**

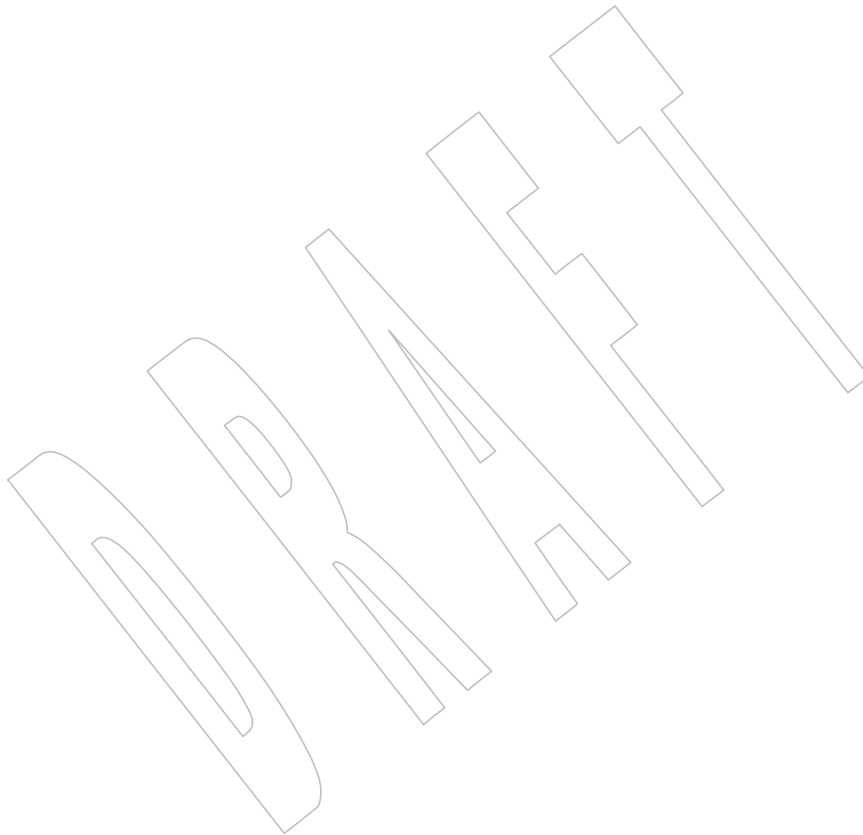
46064 None.

46065 **SEE ALSO**

46066 *alarm()*, *chmod*, *close()*, *dup()*, *exec*, *exit()*, *fcntl()*, *fork()*, *fstatat()*, *kill*, *open()*,
46067 *posix_spawn_file_actions_addclose()*, *posix_spawn_file_actions_adddup2()*, -
46068 *posix_spawn_file_actions_destroy()*, *posix_spawnattr_destroy()*, *posix_spawnattr_getsigdefault()*, -
46069 *posix_spawnattr_getflags()*, *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*,
46070 *posix_spawnattr_getschedpolicy()*, *posix_spawnattr_getsigmask()*, *sched_setparam()*, -
46071 *sched_setscheduler()*, *setpgid()*, *setuid()*, *times()*, *wait*, *waitid()*

46072 XBD Chapter 8 (on page 159), [<spawn.h>](#) +

	CHANGE HISTORY
46073	
46074	First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
46075	IEEE PASC Interpretation 1003.1 #103 is applied, noting that the signal default actions are
46076	changed as well as the signal mask in step 2.
46077	IEEE PASC Interpretation 1003.1 #132 is applied.
46078	Functionality relating to the Threads option is moved to the Base.



46079 **NAME**

46080 `posix_spawn_file_actions_addclose`, `posix_spawn_file_actions_addopen` — add close or open
 46081 action to spawn file actions object (**ADVANCED REALTIME**)

46082 **SYNOPSIS**

```
46083 SPN #include <spawn.h>
46084
46084 int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
46085 *file_actions, int fildes);
46086 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
46087 *restrict file_actions, int fildes,
46088 const char *restrict path, int oflag, mode_t mode);
```

46089 **DESCRIPTION**

46090 These functions shall add or delete a close or open action to a spawn file actions object.

46091 A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in `<spawn.h>`) and is
 46092 used to specify a series of actions to be performed by a `posix_spawn()` or `posix_spawnwp()`
 46093 operation in order to arrive at the set of open file descriptors for the child process given the set
 46094 of open file descriptors of the parent. POSIX.1-200x does not define comparison or assignment
 46095 operators for the type **posix_spawn_file_actions_t**.

46096 A spawn file actions object, when passed to `posix_spawn()` or `posix_spawnwp()`, shall specify how
 46097 the set of open file descriptors in the calling process is transformed into a set of potentially open
 46098 file descriptors for the spawned process. This transformation shall be as if the specified sequence
 46099 of actions was performed exactly once, in the context of the spawned process (prior to execution
 46100 of the new process image), in the order in which the actions were added to the object;
 46101 additionally, when the new process image is executed, any file descriptor (from this new set)
 46102 which has its `FD_CLOEXEC` flag set shall be closed (see `posix_spawn()`).

46103 The `posix_spawn_file_actions_addclose()` function shall add a *close* action to the object referenced
 46104 by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if `close(fildes)` had been
 46105 called) when a new process is spawned using this file actions object.

46106 The `posix_spawn_file_actions_addopen()` function shall add an *open* action to the object referenced
 46107 by *file_actions* that shall cause the file named by *path* to be opened (as if `open(path, oflag, mode)`
 46108 had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a
 46109 new process is spawned using this file actions object. If *fildes* was already an open file descriptor,
 46110 it shall be closed before the new file is opened.

46111 The string described by *path* shall be copied by the `posix_spawn_file_actions_addopen()` function.

46112 **RETURN VALUE**

46113 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 46114 be returned to indicate the error.

46115 **ERRORS**

46116 These functions shall fail if:

46117 [EBADF] The value specified by *fildes* is negative or greater than or equal to
 46118 {OPEN_MAX}.

46119 These functions may fail if:

46120 [EINVAL] The value specified by *file_actions* is invalid.

46121 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

46122 It shall not be considered an error for the *filides* argument passed to these functions to specify a
 46123 file descriptor for which the specified operation could not be performed at the time of the call.
 46124 Any such error will be detected when the associated file actions object is later used during a
 46125 *posix_spawn()* or *posix_spawnnp()* operation.

46126 EXAMPLES

46127 None.

46128 APPLICATION USAGE

46129 These functions are part of the Spawn option and need not be provided on all implementations.

46130 RATIONALE

46131 A spawn file actions object may be initialized to contain an ordered sequence of *close()*, *dup2()*,
 46132 and *open()* operations to be used by *posix_spawn()* or *posix_spawnnp()* to arrive at the set of open
 46133 file descriptors inherited by the spawned process from the set of open file descriptors in the
 46134 parent at the time of the *posix_spawn()* or *posix_spawnnp()* call. It had been suggested that the
 46135 *close()* and *dup2()* operations alone are sufficient to rearrange file descriptors, and that files
 46136 which need to be opened for use by the spawned process can be handled either by having the
 46137 calling process open them before the *posix_spawn()* or *posix_spawnnp()* call (and close them after),
 46138 or by passing filenames to the spawned process (in *argv*) so that it may open them itself. The
 46139 standard developers recommend that applications use one of these two methods when practical,
 46140 since detailed error status on a failed open operation is always available to the application this
 46141 way. However, the standard developers feel that allowing a spawn file actions object to specify
 46142 open operations is still appropriate because:

- 46143 1. It is consistent with equivalent POSIX.5 (Ada) functionality.
- 46144 2. It supports the I/O redirection paradigm commonly employed by POSIX programs
 46145 designed to be invoked from a shell. When such a program is the child process, it may not
 46146 be designed to open files on its own.
- 46147 3. It allows file opens that might otherwise fail or violate file ownership/access rights if
 46148 executed by the parent process.

46149 Regarding 2. above, note that the spawn open file action provides to *posix_spawn()* and
 46150 *posix_spawnnp()* the same capability that the shell redirection operators provide to *system()*, only
 46151 without the intervening execution of a shell; for example:

```
46152 system ("myprog <file1 3<file2");
```

46153 Regarding 3. above, note that if the calling process needs to open one or more files for access by
 46154 the spawned process, but has insufficient spare file descriptors, then the open action is necessary
 46155 to allow the *open()* to occur in the context of the child process after other file descriptors have
 46156 been closed (that must remain open in the parent).

46157 Additionally, if a parent is executed from a file having a "set-user-id" mode bit set and the
 46158 POSIX_SPAWN_RESETEUIDS flag is set in the spawn attributes, a file created within the parent
 46159 process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file
 46160 created via an *open()* action during *posix_spawn()* or *posix_spawnnp()* will have the parent's real
 46161 ID as its owner; and an open by the parent process may successfully open a file to which the real
 46162 user should not have access or fail to open a file to which the real user should have access.

46163

File Descriptor Mapping46164
46165
46166
46167
46168
46169
46170

The standard developers had originally proposed using an array which specified the mapping of child file descriptors back to those of the parent. It was pointed out by the ballot group that it is not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn()* or *posix_spawnp()* without provision for one or more spare file descriptor entries (which simply may not be available). Such an array requires that an implementation develop a complex strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor at the wrong time.

46171
46172
46173
46174
46175
46176
46177

It was noted by a member of the Ada Language Bindings working group that the approved Ada Language *Start_Process* family of POSIX process primitives use a caller-specified set of file actions to alter the normal *fork()/exec* semantics for inheritance of file descriptors in a very flexible way, yet no such problems exist because the burden of determining how to achieve the final file descriptor mapping is completely on the application. Furthermore, although the file actions interface appears frightening at first glance, it is actually quite simple to implement in either a library or the kernel.

46178

FUTURE DIRECTIONS

46179

None.

46180

SEE ALSO46181
46182

close(), *dup()*, *open()*, *posix_spawn()*, *posix_spawn_file_actions_adddup2()*,
posix_spawn_file_actions_destroy()

46183

XBD <[spawn.h](#)>

46184

CHANGE HISTORY

46185

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46186

46187

IEEE PASC Interpretation 1003.1 #105 is applied, adding a note to the DESCRIPTION that the string pointed to by *path* is copied by the *posix_spawn_file_actions_addopen()* function.

46188 **NAME**

46189 `posix_spawn_file_actions_adddup2` — add `dup2` action to spawn file actions object
 46190 (**ADVANCED REALTIME**)

46191 **SYNOPSIS**

```
46192 SPN #include <spawn.h>
46193
46193 int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
46194     *file_actions, int fildes, int newfildes);
```

46195 **DESCRIPTION**

46196 The `posix_spawn_file_actions_adddup2()` function shall add a `dup2()` action to the object
 46197 referenced by `file_actions` that shall cause the file descriptor `fildes` to be duplicated as `newfildes` (as
 46198 if `dup2(fildes, newfildes)` had been called) when a new process is spawned using this file actions
 46199 object.

46200 A spawn file actions object is as defined in [`posix_spawn_file_actions_addclose\(\)`](#).

46201 **RETURN VALUE**

46202 Upon successful completion, the `posix_spawn_file_actions_adddup2()` function shall return zero;
 46203 otherwise, an error number shall be returned to indicate the error.

46204 **ERRORS**

46205 The `posix_spawn_file_actions_adddup2()` function shall fail if:

46206 [EBADF] The value specified by `fildes` or `newfildes` is negative or greater than or equal to
 46207 {OPEN_MAX}.

46208 [ENOMEM] Insufficient memory exists to add to the spawn file actions object.

46209 The `posix_spawn_file_actions_adddup2()` function may fail if:

46210 [EINVAL] The value specified by `file_actions` is invalid.

46211 It shall not be considered an error for the `fildes` argument passed to the
 46212 `posix_spawn_file_actions_adddup2()` function to specify a file descriptor for which the specified
 46213 operation could not be performed at the time of the call. Any such error will be detected when
 46214 the associated file actions object is later used during a `posix_spawn()` or `posix_spawnnp()`
 46215 operation.

46216 **EXAMPLES**

46217 None.

46218 **APPLICATION USAGE**

46219 The `posix_spawn_file_actions_adddup2()` function is part of the Spawn option and need not be
 46220 provided on all implementations.

46221 **RATIONALE**

46222 Refer to the RATIONALE in [`posix_spawn_file_actions_addclose\(\)`](#).

46223 **FUTURE DIRECTIONS**

46224 None.

46225 **SEE ALSO**

46226 [`dup\(\)`](#), [`posix_spawn\(\)`](#), [`posix_spawn_file_actions_addclose\(\)`](#), [`posix_spawn_file_actions_destroy\(\)`](#)

46227 XBD [<spawn.h>](#)

46228

CHANGE HISTORY

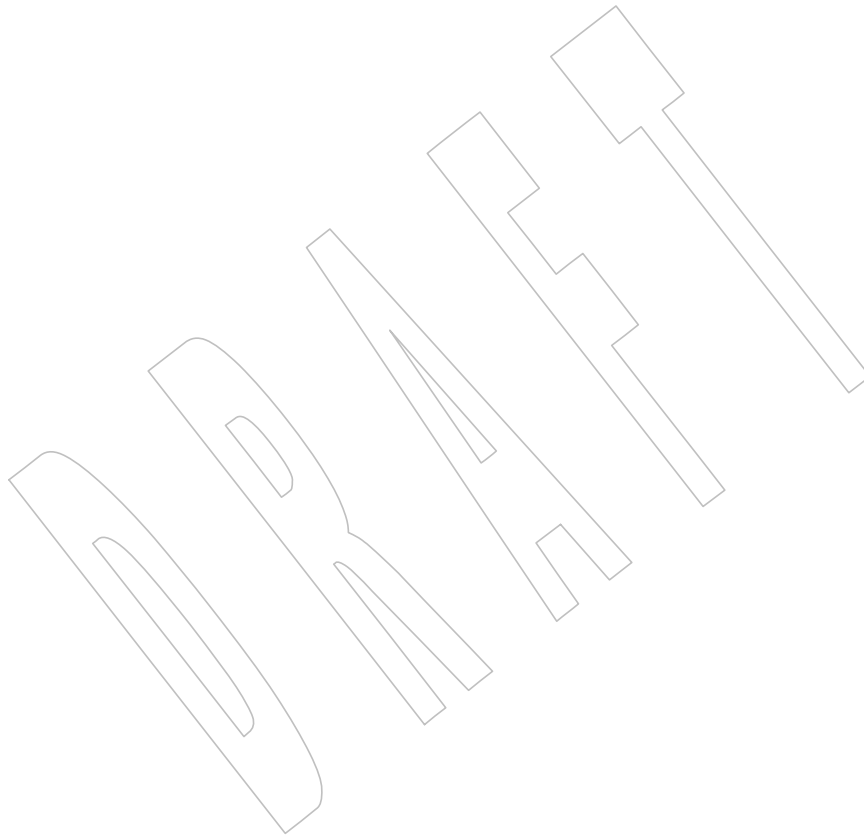
46229

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46230

IEEE PASC Interpretation 1003.1 #104 is applied, noting that the [EBADF] error can apply to the *newfildes* argument in addition to *fildes*.

46231



46232 **NAME**

46233 `posix_spawn_file_actions_addopen` — add open action to spawn file actions object
46234 (ADVANCED REALTIME)

46235 **SYNOPSIS**

```
46236 SPN #include <spawn.h>  
46237 int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t  
46238 *restrict file_actions, int fildes,  
46239 const char *restrict path, int oflag, mode_t mode);
```

46240 **DESCRIPTION**

46241 Refer to [*posix_spawn_file_actions_addclose\(\)*](#).

46242 **NAME**

46243 `posix_spawn_file_actions_destroy`, `posix_spawn_file_actions_init` — destroy and initialize
 46244 spawn file actions object (**ADVANCED REALTIME**)

46245 **SYNOPSIS**

```
46246 SPN #include <spawn.h>
46247
46247 int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
46248 *file_actions);
46249 int posix_spawn_file_actions_init(posix_spawn_file_actions_t
46250 *file_actions);
```

46251 **DESCRIPTION**

46252 The `posix_spawn_file_actions_destroy()` function shall destroy the object referenced by `file_actions`;
 46253 the object becomes, in effect, uninitialized. An implementation may cause
 46254 `posix_spawn_file_actions_destroy()` to set the object referenced by `file_actions` to an invalid value. A
 46255 destroyed spawn file actions object can be reinitialized using `posix_spawn_file_actions_init()`; the
 46256 results of otherwise referencing the object after it has been destroyed are undefined.

46257 The `posix_spawn_file_actions_init()` function shall initialize the object referenced by `file_actions` to
 46258 contain no file actions for `posix_spawn()` or `posix_spawnnp()` to perform.

46259 A spawn file actions object is as defined in [`posix_spawn_file_actions_addclose\(\)`](#).

46260 The effect of initializing an already initialized spawn file actions object is undefined.

46261 **RETURN VALUE**

46262 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 46263 be returned to indicate the error.

46264 **ERRORS**

46265 The `posix_spawn_file_actions_init()` function shall fail if:

46266 [ENOMEM] Insufficient memory exists to initialize the spawn file actions object.

46267 The `posix_spawn_file_actions_destroy()` function may fail if:

46268 [EINVAL] The value specified by `file_actions` is invalid.

46269 **EXAMPLES**

46270 None.

46271 **APPLICATION USAGE**

46272 These functions are part of the Spawn option and need not be provided on all implementations.

46273 **RATIONALE**

46274 Refer to the RATIONALE in [`posix_spawn_file_actions_addclose\(\)`](#).

46275 **FUTURE DIRECTIONS**

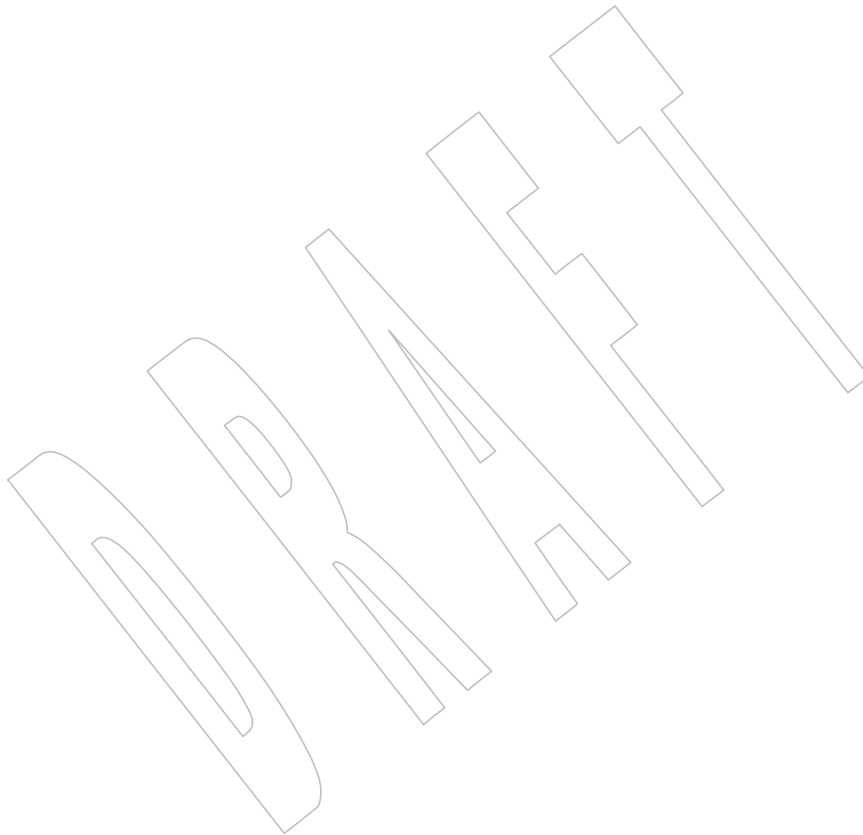
46276 None.

46277 **SEE ALSO**

46278 [`posix_spawn\(\)`](#)

46279 XBD [<spawn.h>](#)

46280	CHANGE HISTORY
46281	First released in Issue 6. Derived from IEEE Std 1003.1d-1999.
46282	In the SYNOPSIS, the inclusion of <code><sys/types.h></code> is no longer required.



46283 **NAME**

46284 `posix_spawnattr_destroy`, `posix_spawnattr_init` — destroy and initialize spawn attributes object
 46285 (**ADVANCED REALTIME**)

46286 **SYNOPSIS**

```
46287 SPN #include <spawn.h>
46288
46288 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
46289 int posix_spawnattr_init(posix_spawnattr_t *attr);
```

46290 **DESCRIPTION**

46291 The `posix_spawnattr_destroy()` function shall destroy a spawn attributes object. A destroyed `attr`
 46292 attributes object can be reinitialized using `posix_spawnattr_init()`; the results of otherwise
 46293 referencing the object after it has been destroyed are undefined. An implementation may cause
 46294 `posix_spawnattr_destroy()` to set the object referenced by `attr` to an invalid value.

46295 The `posix_spawnattr_init()` function shall initialize a spawn attributes object `attr` with the default
 46296 value for all of the individual attributes used by the implementation. Results are undefined if
 46297 `posix_spawnattr_init()` is called specifying an already initialized `attr` attributes object.

46298 A spawn attributes object is of type **posix_spawnattr_t** (defined in **<spawn.h>**) and is used to
 46299 specify the inheritance of process attributes across a spawn operation. POSIX.1-200x does not
 46300 define comparison or assignment operators for the type **posix_spawnattr_t**.

46301 Each implementation shall document the individual attributes it uses and their default values
 46302 unless these values are defined by POSIX.1-200x. Attributes not defined by POSIX.1-200x, their
 46303 default values, and the names of the associated functions to get and set those attribute values are
 46304 implementation-defined.

46305 The resulting spawn attributes object (possibly modified by setting individual attribute values),
 46306 is used to modify the behavior of `posix_spawn()` or `posix_spawnnp()`. After a spawn attributes
 46307 object has been used to spawn a process by a call to a `posix_spawn()` or `posix_spawnnp()`, any
 46308 function affecting the attributes object (including destruction) shall not affect any process that
 46309 has been spawned in this way.

46310 **RETURN VALUE**

46311 Upon successful completion, `posix_spawnattr_destroy()` and `posix_spawnattr_init()` shall return
 46312 zero; otherwise, an error number shall be returned to indicate the error.

46313 **ERRORS**

46314 The `posix_spawnattr_init()` function shall fail if:

46315 [ENOMEM] Insufficient memory exists to initialize the spawn attributes object.

46316 The `posix_spawnattr_destroy()` function may fail if:

46317 [EINVAL] The value specified by `attr` is invalid.

46318 **EXAMPLES**

46319 None.

46320 **APPLICATION USAGE**

46321 These functions are part of the Spawn option and need not be provided on all implementations.

46322 **RATIONALE**

46323 The original spawn interface proposed in POSIX.1-200x defined the attributes that specify the
 46324 inheritance of process attributes across a spawn operation as a structure. In order to be able to
 46325 separate optional individual attributes under their appropriate options (that is, the *spawn-*
 46326 *schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and
 46327 also for extensibility and consistency with the newer POSIX interfaces, the attributes interface
 46328 has been changed to an opaque data type. This interface now consists of the type
 46329 **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to
 46330 initialize or destroy the attributes object, and to set or get each individual attribute. Although the
 46331 new object-oriented interface is more verbose than the original structure, it is simple to use,
 46332 more extensible, and easy to implement.

46333 **FUTURE DIRECTIONS**

46334 None.

46335 **SEE ALSO**

46336 *posix_spawn()*, *posix_spawnattr_getsigdefault()*, *posix_spawnattr_getflags()*,
 46337 *posix_spawnattr_getpgroup()*, *posix_spawnattr_getschedparam()*, *posix_spawnattr_getschedpolicy()*,
 46338 *posix_spawnattr_getsigmask()*

46339 XBD <spawn.h>

46340 **CHANGE HISTORY**

46341 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

46342 IEEE PASC Interpretation 1003.1 #106 is applied, noting that the effect of initializing an already
 46343 initialized spawn attributes option is undefined.

46344

NAME

46345

posix_spawnattr_getflags, posix_spawnattr_setflags — get and set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**)

46346

46347

SYNOPSIS

46348

```
SPN #include <spawn.h>
```

46349

```
int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
    short *restrict flags);
```

46350

46351

```
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

46352

DESCRIPTION

46353

The *posix_spawnattr_getflags()* function shall obtain the value of the *spawn-flags* attribute from the attributes object referenced by *attr*.

46354

46355

The *posix_spawnattr_setflags()* function shall set the *spawn-flags* attribute in an initialized attributes object referenced by *attr*.

46356

46357

The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the new process image when invoking *posix_spawn()* or *posix_spawnnp()*. It is the bitwise-inclusive OR of zero or more of the following flags:

46358

46359

46360

```
POSIX_SPAWN_RESETIDS
```

46361

```
POSIX_SPAWN_SETPGROUP
```

46362

```
POSIX_SPAWN_SETSIGDEF
```

46363

```
POSIX_SPAWN_SETSIGMASK
```

46364

PS

```
POSIX_SPAWN_SETSCHEDPARAM
```

46365

```
POSIX_SPAWN_SETSCHEDULER
```

46366

These flags are defined in **<spawn.h>**. The default value of this attribute shall be as if no flags were set.

46367

46368

RETURN VALUE

46369

Upon successful completion, *posix_spawnattr_getflags()* shall return zero and store the value of the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an error number shall be returned to indicate the error.

46370

46371

46372

Upon successful completion, *posix_spawnattr_setflags()* shall return zero; otherwise, an error number shall be returned to indicate the error.

46373

46374

ERRORS

46375

These functions may fail if:

46376

[EINVAL] The value specified by *attr* is invalid.

46377

The *posix_spawnattr_setflags()* function may fail if:

46378

[EINVAL] The value of the attribute being set is not valid.

46379

EXAMPLES

46380

None.

46381

APPLICATION USAGE

46382

These functions are part of the Spawn option and need not be provided on all implementations.

46383

RATIONALE

46384

None.

46385

FUTURE DIRECTIONS

46386

None.

46387

SEE ALSO

46388

posix_spawn(), posix_spawnattr_destroy(), posix_spawnattr_getsigdefault(),

46389

posix_spawnattr_getpgroup(), posix_spawnattr_getschedparam(), posix_spawnattr_getschedpolicy(),

46390

posix_spawnattr_getsigmask()

46391

XBD <spawn.h>

46392

CHANGE HISTORY

46393

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

DRAFT

46394 **NAME**

46395 `posix_spawnattr_getpgroup`, `posix_spawnattr_setpgroup` — get and set the spawn-pgroup
 46396 attribute of a spawn attributes object (**ADVANCED REALTIME**)

46397 **SYNOPSIS**

```
46398 SPN #include <spawn.h>
46399
46399 int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
46400 pid_t *restrict pgroup);
46401 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

46402 **DESCRIPTION**

46403 The `posix_spawnattr_getpgroup()` function shall obtain the value of the `spawn-pgroup` attribute
 46404 from the attributes object referenced by `attr`.

46405 The `posix_spawnattr_setpgroup()` function shall set the `spawn-pgroup` attribute in an initialized
 46406 attributes object referenced by `attr`.

46407 The `spawn-pgroup` attribute represents the process group to be joined by the new process image
 46408 in a spawn operation (if `POSIX_SPAWN_SETPGROUP` is set in the `spawn-flags` attribute). The
 46409 default value of this attribute shall be zero.

46410 **RETURN VALUE**

46411 Upon successful completion, `posix_spawnattr_getpgroup()` shall return zero and store the value of
 46412 the `spawn-pgroup` attribute of `attr` into the object referenced by the `pgroup` parameter; otherwise,
 46413 an error number shall be returned to indicate the error.

46414 Upon successful completion, `posix_spawnattr_setpgroup()` shall return zero; otherwise, an error
 46415 number shall be returned to indicate the error.

46416 **ERRORS**

46417 These functions may fail if:

46418 [EINVAL] The value specified by `attr` is invalid.

46419 The `posix_spawnattr_setpgroup()` function may fail if:

46420 [EINVAL] The value of the attribute being set is not valid.

46421 **EXAMPLES**

46422 None.

46423 **APPLICATION USAGE**

46424 These functions are part of the Spawn option and need not be provided on all implementations.

46425 **RATIONALE**

46426 None.

46427 **FUTURE DIRECTIONS**

46428 None.

46429 **SEE ALSO**

46430 [posix_spawn\(\)](#), [posix_spawnattr_destroy\(\)](#), [posix_spawnattr_getsigdefault\(\)](#),
 46431 [posix_spawnattr_getflags\(\)](#), [posix_spawnattr_getschedparam\(\)](#), [posix_spawnattr_getschedpolicy\(\)](#),
 46432 [posix_spawnattr_getsigmask\(\)](#)

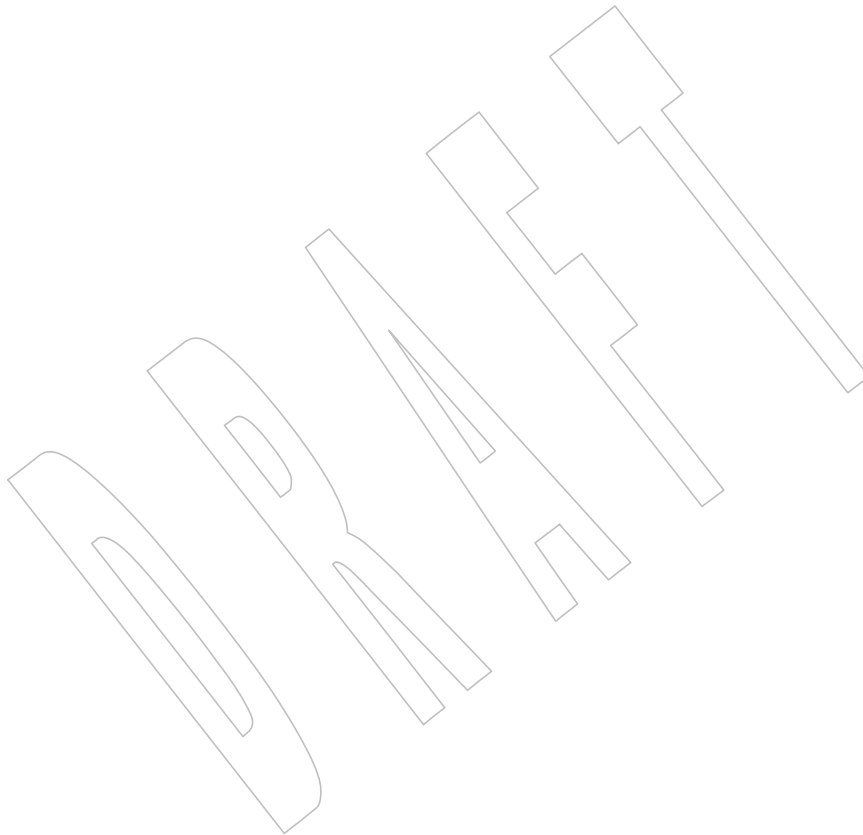
46433 XBD [<spawn.h>](#)

46434

46435

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.



46436 **NAME**

46437 `posix_spawnattr_getschedparam`, `posix_spawnattr_setschedparam` — get and set the spawn-
 46438 `schedparam` attribute of a spawn attributes object (**ADVANCED REALTIME**)

46439 **SYNOPSIS**

```
46440 SPN PS #include <spawn.h>
46441 #include <sched.h>
46442
46442 int posix_spawnattr_getschedparam(const posix_spawnattr_t
46443     *restrict attr, struct sched_param *restrict schedparam);
46444 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
46445     const struct sched_param *restrict schedparam);
```

46446 **DESCRIPTION**

46447 The `posix_spawnattr_getschedparam()` function shall obtain the value of the `spawn-schedparam`
 46448 attribute from the attributes object referenced by `attr`.

46449 The `posix_spawnattr_setschedparam()` function shall set the `spawn-schedparam` attribute in an
 46450 initialized attributes object referenced by `attr`.

46451 The `spawn-schedparam` attribute represents the scheduling parameters to be assigned to the new
 46452 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` or
 46453 `POSIX_SPAWN_SETSCHEDPARAM` is set in the `spawn-flags` attribute). The default value of this
 46454 attribute is unspecified.

46455 **RETURN VALUE**

46456 Upon successful completion, `posix_spawnattr_getschedparam()` shall return zero and store the
 46457 value of the `spawn-schedparam` attribute of `attr` into the object referenced by the `schedparam`
 46458 parameter; otherwise, an error number shall be returned to indicate the error.

46459 Upon successful completion, `posix_spawnattr_setschedparam()` shall return zero; otherwise, an
 46460 error number shall be returned to indicate the error.

46461 **ERRORS**

46462 These functions may fail if:

46463 [EINVAL] The value specified by `attr` is invalid.

46464 The `posix_spawnattr_setschedparam()` function may fail if:

46465 [EINVAL] The value of the attribute being set is not valid.

46466 **EXAMPLES**

46467 None.

46468 **APPLICATION USAGE**

46469 These functions are part of the Spawn and Process Scheduling options and need not be provided
 46470 on all implementations.

46471 **RATIONALE**

46472 None.

46473 **FUTURE DIRECTIONS**

46474 None.

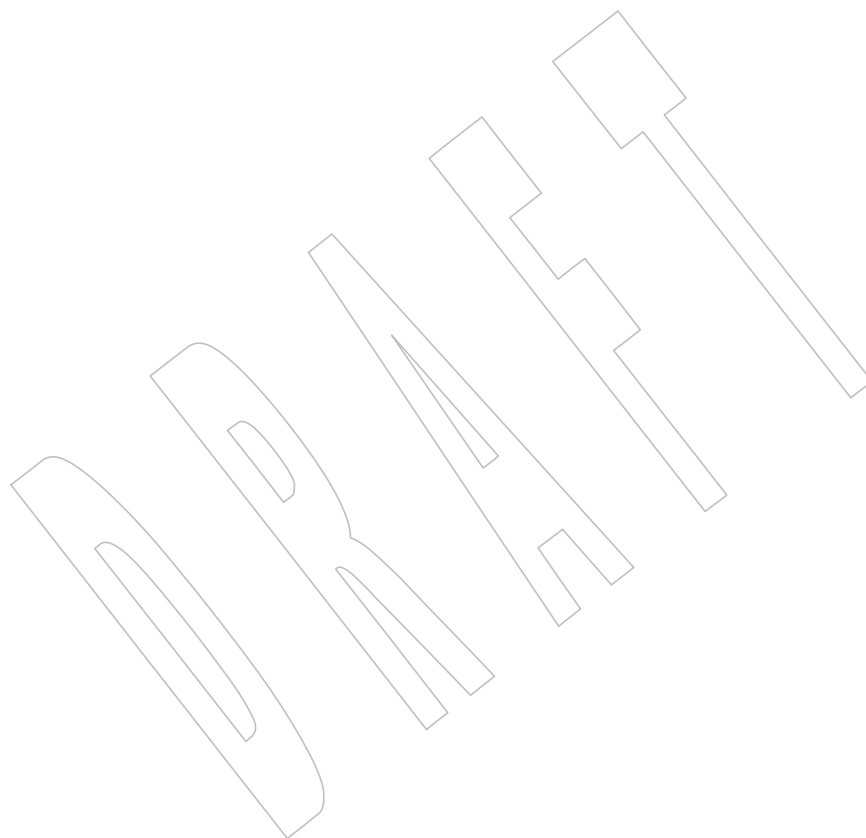
posix_spawnattr_getschedparam()*System Interfaces*46475 **SEE ALSO**

46476 *posix_spawn(), posix_spawnattr_destroy(), posix_spawnattr_getsigdefault(),* -
 46477 *posix_spawnattr_getflags(), posix_spawnattr_getpgroup(), posix_spawnattr_getschedpolicy(),*
 46478 *posix_spawnattr_getsigmask()*

46479 XBD [<sched.h>](#), [<spawn.h>](#) +

46480 **CHANGE HISTORY**

46481 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.



46482 **NAME**

46483 `posix_spawnattr_getschedpolicy`, `posix_spawnattr_setschedpolicy` — get and set the spawn-
 46484 `schedpolicy` attribute of a spawn attributes object (**ADVANCED REALTIME**)

46485 **SYNOPSIS**

```
46486 SPN PS #include <spawn.h>
46487 #include <sched.h>
46488
46488 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
46489 *restrict attr, int *restrict schedpolicy);
46490
46490 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
46491 int schedpolicy);
```

46492 **DESCRIPTION**

46493 The `posix_spawnattr_getschedpolicy()` function shall obtain the value of the `spawn-schedpolicy`
 46494 attribute from the attributes object referenced by `attr`.

46495 The `posix_spawnattr_setschedpolicy()` function shall set the `spawn-schedpolicy` attribute in an
 46496 initialized attributes object referenced by `attr`.

46497 The `spawn-schedpolicy` attribute represents the scheduling policy to be assigned to the new
 46498 process image in a spawn operation (if `POSIX_SPAWN_SETSCHEDULER` is set in the `spawn-`
 46499 `flags` attribute). The default value of this attribute is unspecified.

46500 **RETURN VALUE**

46501 Upon successful completion, `posix_spawnattr_getschedpolicy()` shall return zero and store the
 46502 value of the `spawn-schedpolicy` attribute of `attr` into the object referenced by the `schedpolicy`
 46503 parameter; otherwise, an error number shall be returned to indicate the error.

46504 Upon successful completion, `posix_spawnattr_setschedpolicy()` shall return zero; otherwise, an
 46505 error number shall be returned to indicate the error.

46506 **ERRORS**

46507 These functions may fail if:

46508 [EINVAL] The value specified by `attr` is invalid.

46509 The `posix_spawnattr_setschedpolicy()` function may fail if:

46510 [EINVAL] The value of the attribute being set is not valid.

46511 **EXAMPLES**

46512 None.

46513 **APPLICATION USAGE**

46514 These functions are part of the Spawn and Process Scheduling options and need not be provided
 46515 on all implementations.

46516 **RATIONALE**

46517 None.

46518 **FUTURE DIRECTIONS**

46519 None.

46520 **SEE ALSO**

46521 [*posix_spawn\(\)*](#), [*posix_spawnattr_destroy\(\)*](#), [*posix_spawnattr_getsigdefault\(\)*](#),
 46522 [*posix_spawnattr_getflags\(\)*](#), [*posix_spawnattr_getpgroup\(\)*](#), [*posix_spawnattr_getschedparam\(\)*](#),
 46523 [*posix_spawnattr_getsigmask\(\)*](#)

46524

XBD <sched.h>, <spawn.h>

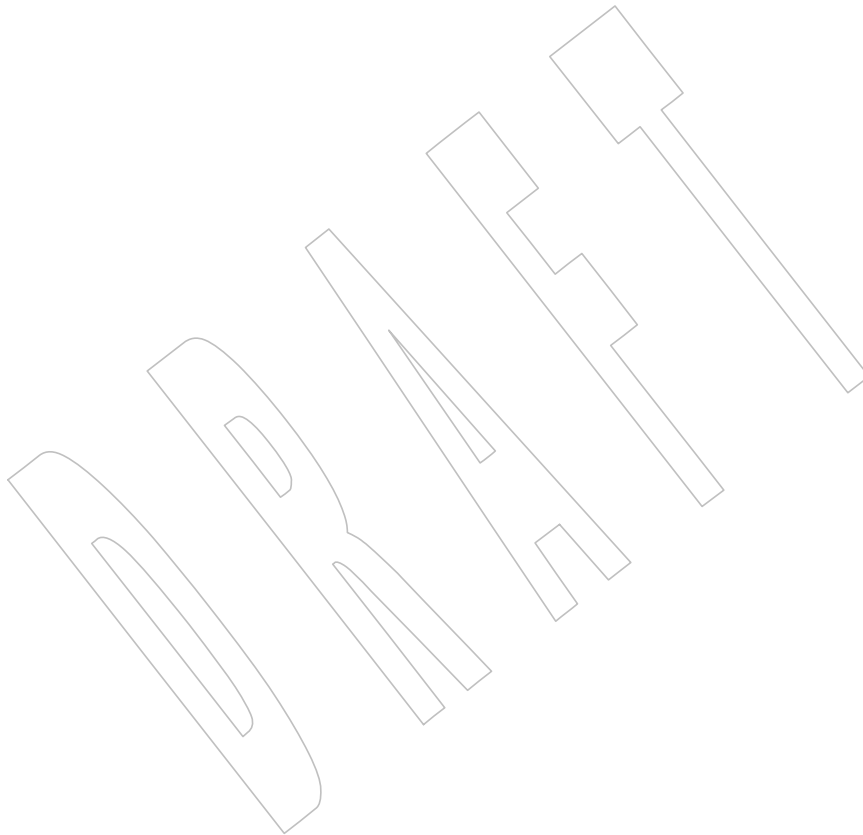
+

46525

CHANGE HISTORY

46526

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.



46527 **NAME**

46528 `posix_spawnattr_getsigdefault`, `posix_spawnattr_setsigdefault` — get and set the spawn-
 46529 sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

46530 **SYNOPSIS**

```
46531 SPN #include <signal.h>
46532 #include <spawn.h>
46533
46533 int posix_spawnattr_getsigdefault(const posix_spawnattr_t
46534 *restrict attr, sigset_t *restrict sigdefault);
46535 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
46536 const sigset_t *restrict sigdefault);
```

46537 **DESCRIPTION**

46538 The `posix_spawnattr_getsigdefault()` function shall obtain the value of the `spawn-sigdefault`
 46539 attribute from the attributes object referenced by `attr`.

46540 The `posix_spawnattr_setsigdefault()` function shall set the `spawn-sigdefault` attribute in an
 46541 initialized attributes object referenced by `attr`.

46542 The `spawn-sigdefault` attribute represents the set of signals to be forced to default signal handling
 46543 in the new process image (if `POSIX_SPAWN_SETSIGDEF` is set in the `spawn-flags` attribute) by a
 46544 spawn operation. The default value of this attribute shall be an empty signal set.

46545 **RETURN VALUE**

46546 Upon successful completion, `posix_spawnattr_getsigdefault()` shall return zero and store the value
 46547 of the `spawn-sigdefault` attribute of `attr` into the object referenced by the `sigdefault` parameter;
 46548 otherwise, an error number shall be returned to indicate the error.

46549 Upon successful completion, `posix_spawnattr_setsigdefault()` shall return zero; otherwise, an error
 46550 number shall be returned to indicate the error.

46551 **ERRORS**

46552 These functions may fail if:

46553 [EINVAL] The value specified by `attr` is invalid.

46554 The `posix_spawnattr_setsigdefault()` function may fail if:

46555 [EINVAL] The value of the attribute being set is not valid.

46556 **EXAMPLES**

46557 None.

46558 **APPLICATION USAGE**

46559 These functions are part of the Spawn option and need not be provided on all implementations.

46560 **RATIONALE**

46561 None.

46562 **FUTURE DIRECTIONS**

46563 None.

46564 **SEE ALSO**

46565 [posix_spawn\(\)](#), [posix_spawnattr_destroy\(\)](#), [posix_spawnattr_getflags\(\)](#), [posix_spawnattr_getpgroup\(\)](#), -
 46566 [posix_spawnattr_getschedparam\(\)](#), [posix_spawnattr_getschedpolicy\(\)](#), [posix_spawnattr_getsigmask\(\)](#)

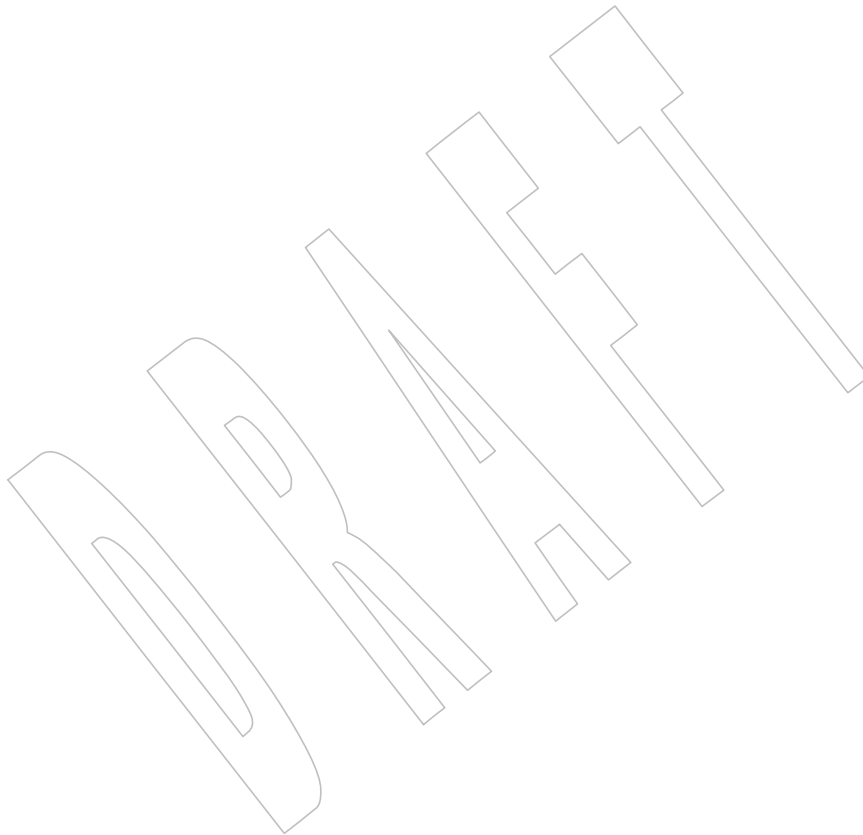
46567 XBD [<signal.h>](#), [<spawn.h>](#) +

46568

46569

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.



46570 **NAME**

46571 `posix_spawnattr_getsigmask`, `posix_spawnattr_setsigmask` — get and set the spawn-sigmask
 46572 attribute of a spawn attributes object (**ADVANCED REALTIME**)

46573 **SYNOPSIS**

```
46574 SPN #include <signal.h>
46575 #include <spawn.h>
46576
46576 int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
46577 sigset_t *restrict sigmask);
46578 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
46579 const sigset_t *restrict sigmask);
```

46580 **DESCRIPTION**

46581 The `posix_spawnattr_getsigmask()` function shall obtain the value of the `spawn-sigmask` attribute
 46582 from the attributes object referenced by `attr`.

46583 The `posix_spawnattr_setsigmask()` function shall set the `spawn-sigmask` attribute in an initialized
 46584 attributes object referenced by `attr`.

46585 The `spawn-sigmask` attribute represents the signal mask in effect in the new process image of a
 46586 spawn operation (if `POSIX_SPAWN_SETSIGMASK` is set in the `spawn-flags` attribute). The
 46587 default value of this attribute is unspecified.

46588 **RETURN VALUE**

46589 Upon successful completion, `posix_spawnattr_getsigmask()` shall return zero and store the value
 46590 of the `spawn-sigmask` attribute of `attr` into the object referenced by the `sigmask` parameter;
 46591 otherwise, an error number shall be returned to indicate the error.

46592 Upon successful completion, `posix_spawnattr_setsigmask()` shall return zero; otherwise, an error
 46593 number shall be returned to indicate the error.

46594 **ERRORS**

46595 These functions may fail if:

46596 [EINVAL] The value specified by `attr` is invalid.

46597 The `posix_spawnattr_setsigmask()` function may fail if:

46598 [EINVAL] The value of the attribute being set is not valid.

46599 **EXAMPLES**

46600 None.

46601 **APPLICATION USAGE**

46602 These functions are part of the Spawn option and need not be provided on all implementations.

46603 **RATIONALE**

46604 None.

46605 **FUTURE DIRECTIONS**

46606 None.

46607 **SEE ALSO**

46608 [posix_spawn\(\)](#), [posix_spawnattr_destroy\(\)](#), [posix_spawnattr_getsigdefault\(\)](#),
 46609 [posix_spawnattr_getflags\(\)](#), [posix_spawnattr_getpgroup\(\)](#), [posix_spawnattr_getschedparam\(\)](#),
 46610 [posix_spawnattr_getschedpolicy\(\)](#)

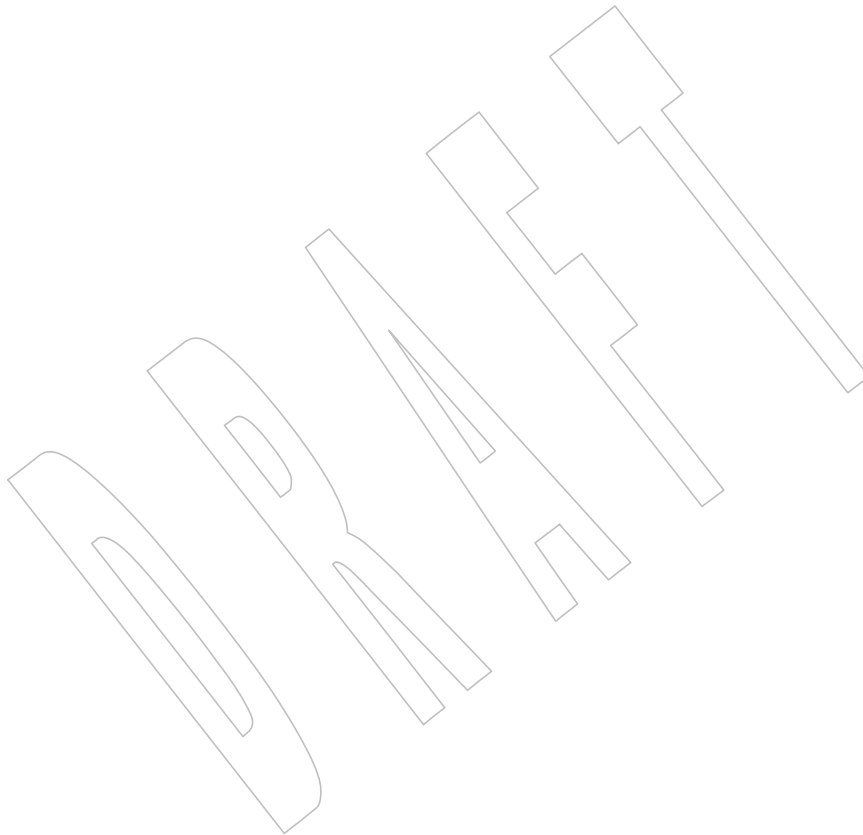
46611 XBD [<signal.h>](#), [<spawn.h>](#)

46612

46613

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.



46614 **NAME**46615 posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)46616 **SYNOPSIS**46617 SPN

```
#include <spawn.h>
```


46618

```
int posix_spawnattr_init(posix_spawnattr_t *attr);
```

46619 **DESCRIPTION**46620 Refer to [posix_spawnattr_destroy\(\)](#).

46621 **NAME**

46622 `posix_spawnattr_setflags` — set the spawn-flags attribute of a spawn attributes object
46623 (**ADVANCED REALTIME**)

46624 **SYNOPSIS**

```
46625 SPN #include <spawn.h>  
46626 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

46627 **DESCRIPTION**

46628 Refer to [posix_spawnattr_getflags\(\)](#).

46629 **NAME**

46630 `posix_spawnattr_setpgroup` — set the spawn-pgroup attribute of a spawn attributes object

46631 (**ADVANCED REALTIME**)

46632 **SYNOPSIS**

46633 SPN `#include <spawn.h>`

46634 `int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);`

46635 **DESCRIPTION**

46636 Refer to *posix_spawnattr_getpgroup()*.

46637 **NAME**

46638 `posix_spawnattr_setschedparam` — set the spawn-schedparam attribute of a spawn attributes
46639 object (**ADVANCED REALTIME**)

46640 **SYNOPSIS**

```
46641 SPN PS #include <sched.h>  
46642 #include <spawn.h>  
46643 int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,  
46644     const struct sched_param *restrict schedparam);
```

46645 **DESCRIPTION**

46646 Refer to [posix_spawnattr_getschedparam\(\)](#).

46647 **NAME**

46648 `posix_spawnattr_setschedpolicy` — set the spawn-schedpolicy attribute of a spawn attributes
46649 object (**ADVANCED REALTIME**)

46650 **SYNOPSIS**

```
46651 SPN PS #include <sched.h>  
46652 #include <spawn.h>  
46653 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,  
46654 int schedpolicy);
```

46655 **DESCRIPTION**

46656 Refer to [posix_spawnattr_getschedpolicy\(\)](#).

46657 **NAME**

46658 `posix_spawnattr_setsigdefault` — set the spawn-sigdefault attribute of a spawn attributes object
46659 (**ADVANCED REALTIME**)

46660 **SYNOPSIS**

```
46661 SPN #include <signal.h>  
46662 #include <spawn.h>  
  
46663 int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,  
46664     const sigset_t *restrict sigdefault);
```

46665 **DESCRIPTION**

46666 Refer to [posix_spawnattr_getsigdefault\(\)](#).

46667 **NAME**

46668 `posix_spawnattr_setsigmask` — set the spawn-sigmask attribute of a spawn attributes object
46669 (**ADVANCED REALTIME**)

46670 **SYNOPSIS**

```
46671 SPN #include <signal.h>  
46672 #include <spawn.h>  
  
46673 int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,  
46674     const sigset_t *restrict sigmask);
```

46675 **DESCRIPTION**

46676 Refer to [posix_spawnattr_getsigmask\(\)](#).

46677 **NAME**46678 posix_spawn — spawn a process (**ADVANCED REALTIME**)46679 **SYNOPSIS**

```
46680 SPN #include <spawn.h>
46681 int posix_spawn(pid_t *restrict pid, const char *restrict file,
46682 const posix_spawn_file_actions_t *file_actions,
46683 const posix_spawnattr_t *restrict attrp,
46684 char *const argv[restrict], char *const envp[restrict]);
```

46685 **DESCRIPTION**46686 Refer to *posix_spawn()*.

46687 **NAME**

46688 `posix_trace_attr_destroy`, `posix_trace_attr_init` — destroy and initialize the trace stream
 46689 attributes object (**TRACING**)

46690 **SYNOPSIS**

```
46691 OB TRC #include <trace.h>
46692
46692 int posix_trace_attr_destroy(trace_attr_t *attr);
46693 int posix_trace_attr_init(trace_attr_t *attr);
```

46694 **DESCRIPTION**

46695 The `posix_trace_attr_destroy()` function shall destroy an initialized trace attributes object. A
 46696 destroyed `attr` attributes object can be reinitialized using `posix_trace_attr_init()`; the results of
 46697 otherwise referencing the object after it has been destroyed are undefined.

46698 The `posix_trace_attr_init()` function shall initialize a trace attributes object `attr` with the default
 46699 value for all of the individual attributes used by a given implementation. The read-only
 46700 *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object
 46701 shall be set to their appropriate values (see [Section 2.11.1.2](#), on page 513).

46702 Results are undefined if `posix_trace_attr_init()` is called specifying an already initialized `attr`
 46703 attributes object.

46704 Implementations may add extensions to the trace attributes object structure as permitted in XBD
 46705 [Chapter 2](#) (on page 15).

46706 The resulting attributes object (possibly modified by setting individual attributes values), when
 46707 used by `posix_trace_create()`, defines the attributes of the trace stream created. A single attributes
 46708 object can be used in multiple calls to `posix_trace_create()`. After one or more trace streams have
 46709 been created using an attributes object, any function affecting that attributes object, including
 46710 destruction, shall not affect any trace stream previously created. An initialized attributes object
 46711 also serves to receive the attributes of an existing trace stream or trace log when calling the
 46712 `posix_trace_get_attr()` function.

46713 **RETURN VALUE**

46714 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 46715 return the corresponding error number.

46716 **ERRORS**

46717 The `posix_trace_attr_destroy()` function may fail if:

46718 [EINVAL] The value of `attr` is invalid.

46719 The `posix_trace_attr_init()` function shall fail if:

46720 [ENOMEM] Insufficient memory exists to initialize the trace attributes object.

46721 **EXAMPLES**

46722 None.

46723 **APPLICATION USAGE**

46724 None.

46725 **RATIONALE**

46726 None.

posix_trace_attr_destroy()

System Interfaces

46727

FUTURE DIRECTIONS

46728

The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions may be removed in a future version.

46729

46730

SEE ALSO

46731

posix_trace_create(), *posix_trace_get_attr()*, *uname*

46732

XBD <[trace.h](#)>

46733

CHANGE HISTORY

46734

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

46735

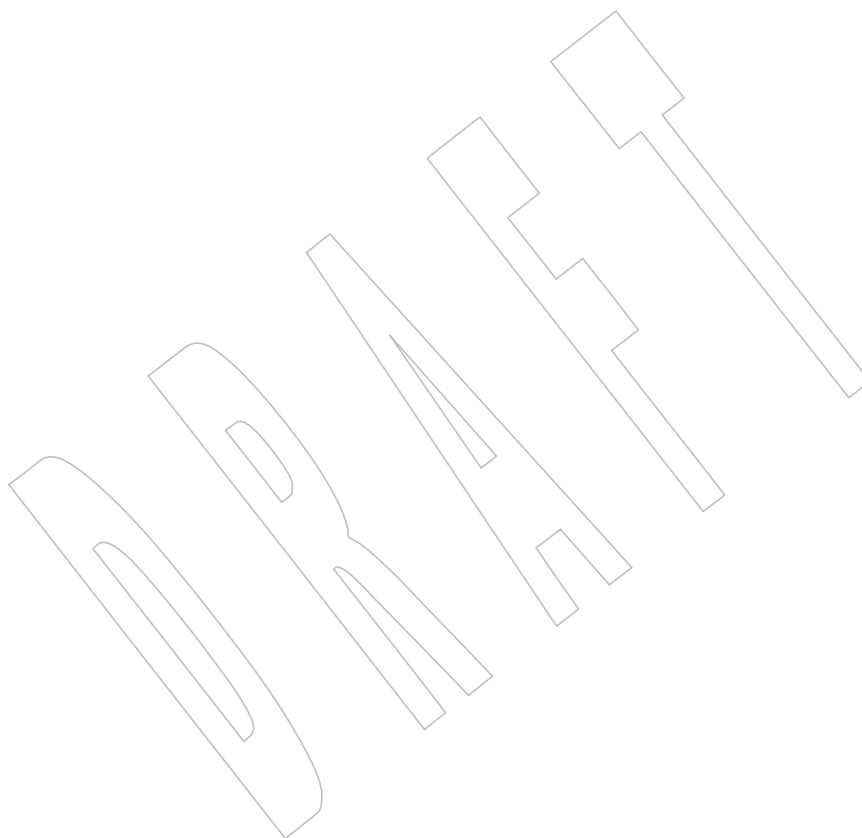
IEEE PASC Interpretation 1003.1 #123 is applied.

46736

Issue 7

46737

The *posix_trace_attr_destroy()* and *posix_trace_attr_init()* functions are marked obsolescent.



46738 **NAME**

46739 `posix_trace_attr_getclockres`, `posix_trace_attr_getcreatetime`, `posix_trace_attr_getgenversion`,
 46740 `posix_trace_attr_getname`, `posix_trace_attr_setname` — retrieve and set information about a
 46741 trace stream (**TRACING**)

46742 **SYNOPSIS**

```
46743 OB TRC #include <time.h>
46744 #include <trace.h>
46745
46746 int posix_trace_attr_getclockres(const trace_attr_t *attr,
46747 struct timespec *resolution);
46748
46749 int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
46750 struct timespec *createtime);
46751
46752 #include <trace.h>
46753
46754 int posix_trace_attr_getgenversion(const trace_attr_t *attr,
46755 char *genversion);
46756
46757 int posix_trace_attr_getname(const trace_attr_t *attr,
46758 char *tracename);
46759
46760 int posix_trace_attr_setname(trace_attr_t *attr,
46761 const char *tracename);
```

46762 **DESCRIPTION**

46763 The `posix_trace_attr_getclockres()` function shall copy the clock resolution of the clock used to
 46764 generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the
 46765 *attr* argument into the structure pointed to by the *resolution* argument.

46766 The `posix_trace_attr_getcreatetime()` function shall copy the trace stream creation time from the
 46767 *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure
 46768 pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of
 46769 creation of the trace stream.

46770 The `posix_trace_attr_getgenversion()` function shall copy the string containing version information
 46771 from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into
 46772 the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of
 46773 a character array which can store at least {TRACE_NAME_MAX} characters.

46774 The `posix_trace_attr_getname()` function shall copy the string containing the trace name from the
 46775 *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string
 46776 pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character
 46777 array which can store at least {TRACE_NAME_MAX} characters.

46778 The `posix_trace_attr_setname()` function shall set the name in the *trace-name* attribute of the
 46779 attributes object pointed to by the *attr* argument, using the trace name string supplied by the
 46780 *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX}
 46781 characters, the name copied into the *trace-name* attribute may be truncated to one less than the
 46782 length of {TRACE_NAME_MAX} characters. The default value is a null string.

46783 **RETURN VALUE**

46784 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 46785 return the corresponding error number.

46786 If successful, the `posix_trace_attr_getclockres()` function stores the *clock-resolution* attribute value in
 46787 the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

46788 If successful, the `posix_trace_attr_getcreatetime()` function stores the trace stream creation time in

posix_trace_attr_getclockres()

46783 the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

46784 If successful, the *posix_trace_attr_getgenversion()* function stores the trace version information in
46785 the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

46786 If successful, the *posix_trace_attr_getname()* function stores the trace name in the string pointed
46787 to by *tracename*. Otherwise, the content of this string is unspecified.

ERRORS

46788 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
46789 and *posix_trace_attr_getname()* functions may fail if:

46791 [EINVAL] The value specified by one of the arguments is invalid.

EXAMPLES

46792 None.
46793

APPLICATION USAGE

46794 None.
46795

RATIONALE

46796 None.
46797

FUTURE DIRECTIONS

46798 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
46799 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions may be removed in a future
46800 version.
46801

SEE ALSO

46802 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*, *uname*

46803 XBD [<time.h>](#), [<trace.h>](#)
46804

CHANGE HISTORY

46805 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
46806

Issue 7

46807 The *posix_trace_attr_getclockres()*, *posix_trace_attr_getcreatetime()*, *posix_trace_attr_getgenversion()*,
46808 *posix_trace_attr_getname()*, and *posix_trace_attr_setname()* functions are marked obsolescent.
46809

46810 **NAME**

46811 posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy,
 46812 posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited,
 46813 posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy — retrieve and set the
 46814 behavior of a trace stream (**TRACING**)

46815 **SYNOPSIS**

```
46816 OB TRC #include <trace.h>
46817 TRI int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
46818 int *restrict inheritancepolicy);
46819 TRI int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
46820 int *restrict logpolicy);
46821 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
46822 attr, int *restrict streampolicy);
46823 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
46824 int inheritancepolicy);
46825 TRI int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
46826 int logpolicy);
46827 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
46828 int streampolicy);
```

46829 **DESCRIPTION**

46830 TRI The *posix_trace_attr_getinherited()* and *posix_trace_attr_setinherited()* functions, respectively, shall
 46831 get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the
 46832 *fork()* and *spawn()* operations. The *inheritance* attribute of the attributes object pointed to by the
 46833 *attr* argument shall be set to one of the following values defined by manifest constants in the
 46834 **<trace.h>** header:

46835 **POSIX_TRACE_CLOSE_FOR_CHILD**

46836 After a *fork()* or *spawn()* operation, the child shall not be traced, and tracing of the parent
 46837 shall continue.

46838 **POSIX_TRACE_INHERITED**

46839 After a *fork()* or *spawn()* operation, if the parent is being traced, its child shall be
 46840 concurrently traced using the same trace stream.

46841 The default value for the *inheritance* attribute is **POSIX_TRACE_CLOSE_FOR_CHILD**.

46842 TRI The *posix_trace_attr_getlogfullpolicy()* and *posix_trace_attr_setlogfullpolicy()* functions,
 46843 respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the
 46844 attributes object pointed to by the *attr* argument.

46845 The *log-full-policy* attribute shall be set to one of the following values defined by manifest
 46846 constants in the **<trace.h>** header:

46847 **POSIX_TRACE_LOOP**

46848 The trace log shall loop until the associated trace stream is stopped. This policy means that
 46849 when the trace log gets full, the file system shall reuse the resources allocated to the oldest
 46850 trace events that were recorded. In this way, the trace log will always contain the most
 46851 recent trace events flushed.

46852 **POSIX_TRACE_UNTIL_FULL**

46853 The trace stream shall be flushed to the trace log until the trace log is full. This condition can
 46854 be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info**
 46855 structure defined in **<trace.h>**). The last recorded trace event shall be the
 46856 **POSIX_TRACE_STOP** trace event.

- 46857 POSIX_TRACE_APPEND
 46858 The associated trace stream shall be flushed to the trace log without log size limitation. If
 46859 the application specifies POSIX_TRACE_APPEND, the implementation shall ignore the *log-*
 46860 *max-size* attribute.
- 46861 The default value for the *log-full-policy* attribute is POSIX_TRACE_LOOP.
- 46862 The *posix_trace_attr_getstreamfullpolicy()* and *posix_trace_attr_setstreamfullpolicy()* functions,
 46863 respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute
 46864 of the attributes object pointed to by the *attr* argument.
- 46865 The *stream-full-policy* attribute shall be set to one of the following values defined by manifest
 46866 constants in the **<trace.h>** header:
- 46867 POSIX_TRACE_LOOP
 46868 The trace stream shall loop until explicitly stopped by the *posix_trace_stop()* function. This
 46869 policy means that when the trace stream is full, the trace system shall reuse the resources
 46870 allocated to the oldest trace events recorded. In this way, the trace stream will always
 46871 contain the most recent trace events recorded.
- 46872 POSIX_TRACE_UNTIL_FULL
 46873 The trace stream will run until the trace stream resources are exhausted. Then the trace
 46874 stream will stop. This condition can be deduced from *posix_stream_status* and
 46875 *posix_stream_full_status* (see the **posix_trace_status_info** structure defined in **<trace.h>**).
 46876 When this trace stream is read, a POSIX_TRACE_STOP trace event shall be reported after
 46877 reporting the last recorded trace event. The trace system shall reuse the resources allocated
 46878 to any trace events already reported—see the *posix_trace_getnext_event()*,
 46879 *posix_trace_trygetnext_event()*, and *posix_trace_timedgetnext_event()* functions—or already
 46880 flushed for an active trace stream with log if the Trace Log option is supported; see the
 46881 *posix_trace_flush()* function. The trace system shall restart the trace stream when it is empty
 46882 and may restart it sooner. A POSIX_TRACE_START trace event shall be reported before
 46883 reporting the next recorded trace event.
- 46884 TRL POSIX_TRACE_FLUSH
 46885 If the Trace Log option is supported, this policy is identical to the
 46886 POSIX_TRACE_UNTIL_FULL trace stream full policy except that the trace stream shall be
 46887 flushed regularly as if *posix_trace_flush()* had been explicitly called. Defining this policy for
 46888 an active trace stream without log shall be invalid.
- 46889 The default value for the *stream-full-policy* attribute shall be POSIX_TRACE_LOOP for an active
 46890 trace stream without log.
- 46891 TRL If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be
 46892 POSIX_TRACE_FLUSH for an active trace stream with log.
- 46893 **RETURN VALUE**
 46894 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 46895 return the corresponding error number.
- 46896 TRI If successful, the *posix_trace_attr_getinherited()* function shall store the *inheritance* attribute value
 46897 in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.
- 46898 TRL If successful, the *posix_trace_attr_getlogfullpolicy()* function shall store the *log-full-policy* attribute
 46899 value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.
- 46900 If successful, the *posix_trace_attr_getstreamfullpolicy()* function shall store the *stream-full-policy*
 46901 attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is
 46902 undefined.

46903 ERRORS

46904 These functions may fail if:

46905 [EINVAL] The value specified by at least one of the arguments is invalid.

46906 EXAMPLES

46907 None.

46908 APPLICATION USAGE

46909 None.

46910 RATIONALE

46911 None.

46912 FUTURE DIRECTIONS

46913 The following functions:

46914 `posix_trace_attr_getinherited()`
 46915 `posix_trace_attr_getlogfullpolicy()`
 46916 `posix_trace_attr_getstreamfullpolicy()`
 46917 `posix_trace_attr_setinherited()`
 46918 `posix_trace_attr_setlogfullpolicy()`
 46919 `posix_trace_attr_setstreamfullpolicy()`

46920 may be removed in a future version.

46921 SEE ALSO

46922 [fork\(\)](#), [posix_trace_attr_destroy\(\)](#), [posix_trace_create\(\)](#), [posix_trace_get_attr\(\)](#),
 46923 [posix_trace_getnext_event\(\)](#), [posix_trace_start\(\)](#)

46924 XBD <[trace.h](#)>

46925 CHANGE HISTORY

46926 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

46927 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/39 is applied, adding the TRL and TRC
 46928 margin codes to the `posix_trace_attr_setlogfullpolicy()` function.

46929 Issue 7

46930 SD5-XSH-ERN-116 is applied, adding the missing **restrict** keyword to the
 46931 `posix_trace_attr_getstreamfullpolicy()` function declaration.

46932 These functions are marked obsolescent.

46933 NAME

46934 posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize,
 46935 posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize,
 46936 posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize,
 46937 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (TRACING)

46938 SYNOPSIS

```
46939 OB TRC #include <sys/types.h>
46940 #include <trace.h>

46941 TRL int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
46942 size_t *restrict logsize);
46943 int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
46944 size_t *restrict maxdatasize);
46945 int posix_trace_attr_getmaxsystemeventsize(
46946 const trace_attr_t *restrict attr,
46947 size_t *restrict eventsize);
46948 int posix_trace_attr_getmaxusereventsize(
46949 const trace_attr_t *restrict attr,
46950 size_t data_len, size_t *restrict eventsize);
46951 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
46952 size_t *restrict streamsize);
46953 TRL int posix_trace_attr_setlogsize(trace_attr_t *attr,
46954 size_t logsize);
46955 int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
46956 size_t maxdatasize);
46957 int posix_trace_attr_setstreamsize(trace_attr_t *attr,
46958 size_t streamsize);
```

46959 DESCRIPTION

46960 TRL The *posix_trace_attr_getlogsize()* function shall copy the log size, in bytes, from the *log-max-size*
 46961 attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by
 46962 the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for
 46963 system and user trace events in the trace log. The default value for the *log-max-size* attribute is
 46964 implementation-defined.

46965 The *posix_trace_attr_setlogsize()* function shall set the maximum allowed size, in bytes, in the *log-*
 46966 *max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value
 46967 supplied by the *logsize* argument.

46968 The trace log size shall be used if the *log-full-policy* attribute is set to `POSIX_TRACE_LOOP` or
 46969 `POSIX_TRACE_UNTIL_FULL`. If the *log-full-policy* attribute is set to `POSIX_TRACE_APPEND`,
 46970 the implementation shall ignore the *log-max-size* attribute.

46971 The *posix_trace_attr_getmaxdatasize()* function shall copy the maximum user trace event data
 46972 size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr*
 46973 argument into the variable pointed to by the *maxdatasize* argument. The default value for the
 46974 *max-data-size* attribute is implementation-defined.

46975 The *posix_trace_attr_getmaxsystemeventsize()* function shall calculate the maximum memory size,
 46976 in bytes, required to store a single system trace event. This value is calculated for the trace
 46977 stream attributes object pointed to by the *attr* argument and is returned in the variable pointed
 46978 to by the *eventsize* argument.

46979 The values returned as the maximum memory sizes of the user and system trace events shall be
 46980 such that if the sum of the maximum memory sizes of a set of the trace events that may be

46981 recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace
 46982 stream, the system provides the necessary resources for recording all those trace events, without
 46983 loss.

46984 The *posix_trace_attr_getmaxusereventsize()* function shall calculate the maximum memory size, in
 46985 bytes, required to store a single user trace event generated by a call to *posix_trace_event()* with a
 46986 *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for
 46987 the trace stream attributes object pointed to by the *attr* argument and is returned in the variable
 46988 pointed to by the *eventsize* argument.

46989 The *posix_trace_attr_getstreamsize()* function shall copy the stream size, in bytes, from the *stream-*
 46990 *min-size* attribute of the attributes object pointed to by the *attr* argument into the variable
 46991 pointed to by the *streamsize* argument.

46992 This stream size is the current total memory size reserved for system and user trace events in the
 46993 trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The
 46994 stream size refers to memory used to store trace event records. Other stream data (for example,
 46995 trace attribute values) shall not be included in this size.

46996 The *posix_trace_attr_setmaxdatasize()* function shall set the maximum allowed size, in bytes, in
 46997 the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 46998 value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size
 46999 for the user data argument which may be passed to *posix_trace_event()*. The implementation
 47000 shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

47001 The *posix_trace_attr_setstreamsize()* function shall set the minimum allowed size, in bytes, in the
 47002 *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size
 47003 value supplied by the *streamsize* argument.

47004 RETURN VALUE

47005 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47006 return the corresponding error number.

47007 TRL The *posix_trace_attr_getlogsize()* function stores the maximum trace log allowed size in the object
 47008 pointed to by *logsize*, if successful.

47009 The *posix_trace_attr_getmaxdatasize()* function stores the maximum trace event record memory
 47010 size in the object pointed to by *maxdatasize*, if successful.

47011 The *posix_trace_attr_getmaxsystemeventsize()* function stores the maximum memory size to store a
 47012 single system trace event in the object pointed to by *eventsize*, if successful.

47013 The *posix_trace_attr_getmaxusereventsize()* function stores the maximum memory size to store a
 47014 single user trace event in the object pointed to by *eventsize*, if successful.

47015 The *posix_trace_attr_getstreamsize()* function stores the maximum trace stream allowed size in the
 47016 object pointed to by *streamsize*, if successful.

47017 ERRORS

47018 These functions may fail if:

47019 [EINVAL] The value specified by one of the arguments is invalid.

posix_trace_attr_getlogsize()

47020
47021
47022
47023
47024
47025
47026
47027
47028
47029
47030
47031
47032
47033
47034
47035
47036
47037
47038
47039
47040
47041
47042
47043

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The following functions:

posix_trace_attr_getlogsize()
posix_trace_attr_getmaxdatasize()
posix_trace_attr_getmaxsystemeventszize()
posix_trace_attr_getmaxusereventsizze()
posix_trace_attr_getstreamsize()
posix_trace_attr_setlogsize()
posix_trace_attr_setmaxdatasize()
posix_trace_attr_setstreamsize()

may be removed in a future version.

SEE ALSO

posix_trace_attr_destroy(), *posix_trace_create()*, *posix_trace_event()*, *posix_trace_get_attr()*

XBD [<sys/types.h>](#), [<trace.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7

These functions are marked obsolescent.

47044 **NAME**
47045 `posix_trace_attr_getname` — retrieve and set information about a trace stream (**TRACING**)

47046 **SYNOPSIS**

```
47047 OB TRC #include <trace.h>  
47048 int posix_trace_attr_getname(const trace_attr_t *attr,  
47049 char *tracename);
```

47050 **DESCRIPTION**

47051 Refer to [posix_trace_attr_getclockres\(\)](#).

47052 **NAME**

47053 `posix_trace_attr_getstreamfullpolicy` — retrieve and set the behavior of a trace stream
47054 (**TRACING**)

47055 **SYNOPSIS**

47056 OB TRC `#include <trace.h>`

```
47057 int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict  
47058 attr, int *restrict streampolicy);
```

47059 **DESCRIPTION**

47060 Refer to [posix_trace_attr_getinherited\(\)](#).

47061 **NAME**
47062 `posix_trace_attr_getstreamsize` — retrieve and set trace stream size attributes (**TRACING**)

47063 **SYNOPSIS**

```
47064 OB TRC #include <sys/types.h>  
47065 #include <trace.h>  
  
47066 int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,  
47067 size_t *restrict streamsize);
```

47068 **DESCRIPTION**

47069 Refer to [posix_trace_attr_getlogsize\(\)](#).

47070 **NAME**47071 `posix_trace_attr_init` — initialize the trace stream attributes object (**TRACING**)47072 **SYNOPSIS**47073 OB TRC `#include <trace.h>`47074 `int posix_trace_attr_init(trace_attr_t *attr);`47075 **DESCRIPTION**47076 Refer to [posix_trace_attr_destroy\(\)](#).

47077 **NAME**

47078 `posix_trace_attr_setinherited`, `posix_trace_attr_setlogfullpolicy` — retrieve and set the behavior
47079 of a trace stream (**TRACING**)

47080 **SYNOPSIS**

```
47081 OB TRC #include <trace.h>
47082 TRI int posix_trace_attr_setinherited(trace_attr_t *attr,
47083 int inheritancepolicy);
47084 TRL int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
47085 int logpolicy);
```

47086 **DESCRIPTION**

47087 Refer to [posix_trace_attr_getinherited\(\)](#).

posix_trace_attr_setlogsize()*System Interfaces*47088 **NAME**

47089 posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize — retrieve and set trace stream size
 47090 attributes (**TRACING**)

47091 **SYNOPSIS**

```

47092 OB TRC #include <sys/types.h>
47093         #include <trace.h>
47094
47094 TRL     int posix_trace_attr_setlogsize(trace_attr_t *attr,
47095         size_t logsize);
47096 OB TRC  int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
47097         size_t maxdatasize);

```

47098 **DESCRIPTION**

47099 Refer to [posix_trace_attr_getlogsize\(\)](#).

47100 **NAME**
47101 `posix_trace_attr_setname` — retrieve and set information about a trace stream (**TRACING**)

47102 **SYNOPSIS**

```
47103 OB TRC #include <trace.h>  
47104 int posix_trace_attr_setname(trace_attr_t *attr,  
47105     const char *tracename);
```

47106 **DESCRIPTION**

47107 Refer to [posix_trace_attr_getclockres\(\)](#).

47108 **NAME**

47109 `posix_trace_attr_setstreamfullpolicy` — retrieve and set the behavior of a trace stream
47110 (**TRACING**)

47111 **SYNOPSIS**

```
47112 OB TRC #include <trace.h>  
47113  
47113 int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,  
47114 int streampolicy);
```

47115 **DESCRIPTION**

47116 Refer to [*posix_trace_attr_getinherited\(\)*](#).

47117 **NAME**

47118 posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (TRACING)

47119 **SYNOPSIS**

```
47120 OB TRC #include <sys/types.h>  
47121         #include <trace.h>  
  
47122         int posix_trace_attr_setstreamsize(trace_attr_t *attr,  
47123         size_t streamsize);
```

47124 **DESCRIPTION**47125 Refer to *posix_trace_attr_getlogsize()*.

47126 **NAME**47127 `posix_trace_clear` — clear trace stream and trace log (**TRACING**)47128 **SYNOPSIS**

```
47129 OB TRC #include <sys/types.h>
47130 #include <trace.h>
47131 int posix_trace_clear(trace_id_t trid);
```

47132 **DESCRIPTION**

47133 The `posix_trace_clear()` function shall reinitialize the trace stream identified by the argument *trid*
 47134 as if it were returning from the `posix_trace_create()` function, except that the same allocated
 47135 resources shall be reused, the mapping of trace event type identifiers to trace event names shall
 47136 be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it
 47137 remains running and if it was suspended, it remains suspended).

47138 All trace events in the trace stream recorded before the call to `posix_trace_clear()` shall be lost. The
 47139 `posix_stream_full_status` status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee
 47140 that all trace events that occurred during the `posix_trace_clear()` call are recorded; the behavior
 47141 with respect to trace points that may occur during this call is unspecified.

47142 OB TRL If the Trace Log option is supported and the trace stream has been created with a log, the
 47143 `posix_trace_clear()` function shall reinitialize the trace stream with the same behavior as if the
 47144 trace stream was created without the log, plus it shall reinitialize the trace log associated with
 47145 the trace stream identified by the argument *trid* as if it were returning from the
 47146 `posix_trace_create_withlog()` function, except that the same allocated resources, for the trace log,
 47147 may be reused and the associated trace stream status remains unchanged. The first trace event
 47148 recorded in the trace log after the call to `posix_trace_clear()` shall be the same as the first trace
 47149 event recorded in the active trace stream after the call to `posix_trace_clear()`. The
 47150 `posix_log_full_status` status shall be set to `POSIX_TRACE_NOT_FULL`. There is no guarantee that
 47151 all trace events that occurred during the `posix_trace_clear()` call are recorded in the trace log; the
 47152 behavior with respect to trace points that may occur during this call is unspecified. If the log full
 47153 policy is `POSIX_TRACE_APPEND`, the effect of a call to this function is unspecified for the trace
 47154 log associated with the trace stream identified by the *trid* argument.

47155 **RETURN VALUE**

47156 Upon successful completion, the `posix_trace_clear()` function shall return a value of zero.
 47157 Otherwise, it shall return the corresponding error number.

47158 **ERRORS**

47159 The `posix_trace_clear()` function shall fail if:

47160 [EINVAL] The value of the *trid* argument does not correspond to an active trace stream.

47161 **EXAMPLES**

47162 None.

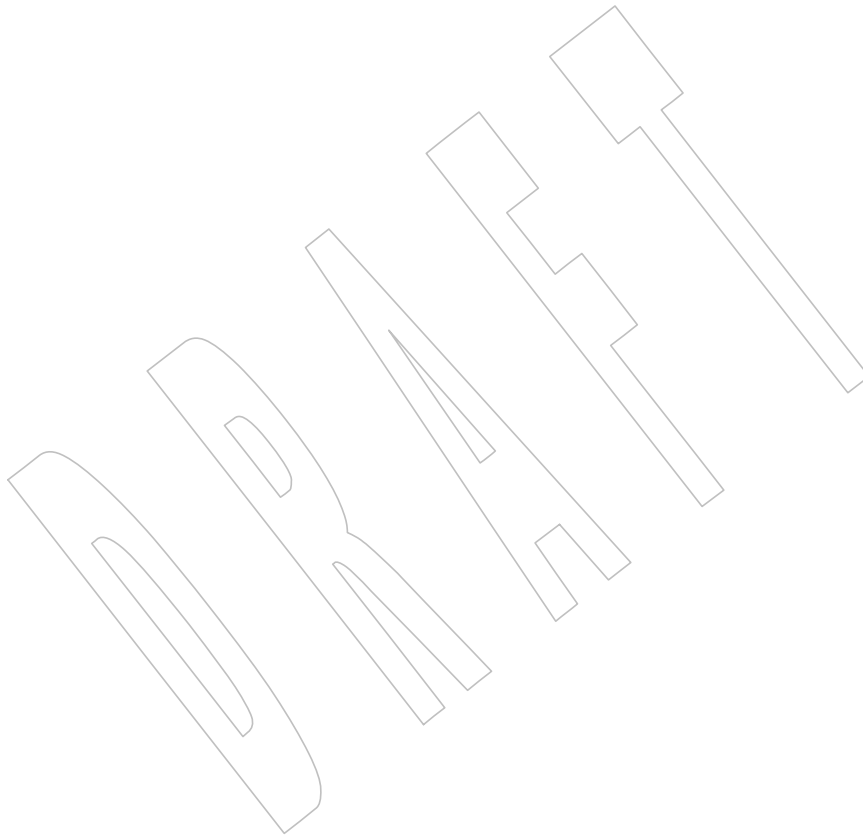
47163 **APPLICATION USAGE**

47164 None.

47165 **RATIONALE**

47166 None.

- 47167 **FUTURE DIRECTIONS**
- 47168 The *posix_trace_clear()* function may be removed in a future version. |
- 47169 **SEE ALSO**
- 47170 *posix_trace_attr_destroy()*, *posix_trace_create()*, *posix_trace_get_attr()*
- 47171 XBD [<sys/types.h>](#), [<trace.h>](#) +
- 47172 **CHANGE HISTORY**
- 47173 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
- 47174 IEEE PASC Interpretation 1003.1 #123 is applied.
- 47175 **Issue 7**
- 47176 The *posix_trace_clear()* function is marked obsolescent.



47177 **NAME**47178 `posix_trace_close`, `posix_trace_open`, `posix_trace_rewind` — trace log management (**TRACING**)47179 **SYNOPSIS**47180 OB TRC `#include <trace.h>`47181 TRL `int posix_trace_close(trace_id_t trid);`47182 `int posix_trace_open(int file_desc, trace_id_t *trid);`47183 `int posix_trace_rewind(trace_id_t trid);`47184 **DESCRIPTION**47185 The `posix_trace_close()` function shall deallocate the trace log identifier indicated by *trid*, and all
47186 of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall
47187 fail.47188 The `posix_trace_open()` function shall allocate the necessary resources and establish the
47189 connection between a trace log identified by the *file_desc* argument and a trace stream identifier
47190 identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid
47191 open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for
47192 reading. The current trace event timestamp, which specifies the timestamp of the trace event that
47193 will be read by the next call to `posix_trace_getnext_event()`, shall be set to the timestamp of the
47194 oldest trace event recorded in the trace log identified by *trid*.47195 The `posix_trace_open()` function shall return a trace stream identifier in the variable pointed to by
47196 the *trid* argument, that may only be used by the following functions:47197 `posix_trace_close()` `posix_trace_get_attr()`
47198 `posix_trace_eventid_equal()` `posix_trace_get_status()`
47199 `posix_trace_eventid_get_name()` `posix_trace_getnext_event()`
47200 `posix_trace_eventtypelist_getnext_id()` `posix_trace_rewind()`
47201 `posix_trace_eventtypelist_rewind()`47202 In particular, notice that the operations normally used by a trace controller process, such as
47203 `posix_trace_start()`, `posix_trace_stop()`, or `posix_trace_shutdown()`, cannot be invoked using the
47204 trace stream identifier returned by the `posix_trace_open()` function.47205 The `posix_trace_rewind()` function shall reset the current trace event timestamp, which specifies
47206 the timestamp of the trace event that will be read by the next call to `posix_trace_getnext_event()`,
47207 to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.47208 **RETURN VALUE**47209 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
47210 return the corresponding error number.47211 If successful, the `posix_trace_open()` function stores the trace stream identifier value in the object
47212 pointed to by *trid*.47213 **ERRORS**47214 The `posix_trace_open()` function shall fail if:

47215 [EINTR] The operation was interrupted by a signal and thus no trace log was opened.

47216 [EINVAL] The object pointed to by *file_desc* does not correspond to a valid trace log.47217 The `posix_trace_close()` and `posix_trace_rewind()` functions may fail if:

47218 [EINVAL] The object pointed to by *trid* does not correspond to a valid trace log.

EXAMPLES

47219 None.
47220

APPLICATION USAGE

47221 None.
47222

RATIONALE

47223 None.
47224

FUTURE DIRECTIONS

47225 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions may be removed in
47226 a future version.
47227

SEE ALSO

47228 *posix_trace_get_attr()*, *posix_trace_get_filter()*, *posix_trace_getnext_event()*
47229

47230 XBD <trace.h> +

CHANGE HISTORY

47231 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
47232

47233 IEEE PASC Interpretation 1003.1 #123 is applied.
47234

Issue 7

47235 The *posix_trace_close()*, *posix_trace_open()*, and *posix_trace_rewind()* functions are marked
47236 obsolescent.

47237 **NAME**

47238 `posix_trace_create`, `posix_trace_create_withlog`, `posix_trace_flush`, `posix_trace_shutdown` —
 47239 trace stream initialization, flush, and shutdown from a process (**TRACING**)

47240 **SYNOPSIS**

```
47241 OB TRC #include <sys/types.h>
47242 #include <trace.h>
47243
47243 int posix_trace_create(pid_t pid,
47244     const trace_attr_t *restrict attr,
47245     trace_id_t *restrict trid);
47246 TRL int posix_trace_create_withlog(pid_t pid,
47247     const trace_attr_t *restrict attr, int file_desc,
47248     trace_id_t *restrict trid);
47249 int posix_trace_flush(trace_id_t trid);
47250 int posix_trace_shutdown(trace_id_t trid);
```

47251 **DESCRIPTION**

47252 The `posix_trace_create()` function shall create an active trace stream. It allocates all the resources
 47253 needed by the trace stream being created for tracing the process specified by `pid` in accordance
 47254 with the `attr` argument. The `attr` argument represents the initial attributes of the trace stream and
 47255 shall have been initialized by the function `posix_trace_attr_init()` prior to the `posix_trace_create()`
 47256 call. If the argument `attr` is `NULL`, the default attributes shall be used. The `attr` attributes object
 47257 shall be manipulated through a set of functions described in the `posix_trace_attr` family of
 47258 functions. If the attributes of the object pointed to by `attr` are modified later, the attributes of the
 47259 trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream
 47260 shall be set to the value of the system clock, if the Timers option is not supported, or to the value
 47261 of the `CLOCK_REALTIME` clock, if the Timers option is supported.

47262 The `pid` argument represents the target process to be traced. If the process executing this function
 47263 does not have appropriate privileges to trace the process identified by `pid`, an error shall be
 47264 returned. If the `pid` argument is zero, the calling process shall be traced.

47265 The `posix_trace_create()` function shall store the trace stream identifier of the new trace stream in
 47266 the object pointed to by the `trid` argument. This trace stream identifier shall be used in
 47267 subsequent calls to control tracing. The `trid` argument may only be used by the following
 47268 functions:

47269	<code>posix_trace_clear()</code>	<code>posix_trace_getnext_event()</code>
47270	<code>posix_trace_eventid_equal()</code>	<code>posix_trace_shutdown()</code>
47271	<code>posix_trace_eventid_get_name()</code>	<code>posix_trace_start()</code>
47272	<code>posix_trace_eventtypelist_getnext_id()</code>	<code>posix_trace_stop()</code>
47273	<code>posix_trace_eventtypelist_rewind()</code>	<code>posix_trace_timedgetnext_event()</code>
47274	<code>posix_trace_get_attr()</code>	<code>posix_trace_trid_eventid_open()</code>
47275	<code>posix_trace_get_status()</code>	<code>posix_trace_trygetnext_event()</code>

47276 TEF If the Trace Event Filter option is supported, the following additional functions may use the `trid`
 47277 argument:

```
47278 posix_trace_get_filter()  posix_trace_set_filter()
```

47279 In particular, notice that the operations normally used by a trace analyzer process, such as
 47280 `posix_trace_rewind()` or `posix_trace_close()`, cannot be invoked using the trace stream identifier
 47281 returned by the `posix_trace_create()` function.

- 47282 TEF A trace stream shall be created in a suspended state. If the Trace Event Filter option is
47283 supported, its trace event type filter shall be empty.
- 47284 The *posix_trace_create()* function may be called multiple times from the same or different
47285 processes, with the system-wide limit indicated by the runtime invariant value
47286 {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.
- 47287 The trace stream identifier returned by the *posix_trace_create()* function in the argument pointed
47288 to by *trid* is valid only in the process that made the function call. If it is used from another
47289 process, that is a child process, in functions defined in POSIX.1-200x, these functions shall return
47290 with the error [EINVAL].
- 47291 TRL The *posix_trace_create_withlog()* function shall be equivalent to *posix_trace_create()*, except that it
47292 associates a trace log with this stream. The *file_desc* argument shall be the file descriptor
47293 designating the trace log destination. The function shall fail if this file descriptor refers to a file
47294 with a file type that is not compatible with the log policy associated with the trace log. The list of
47295 the appropriate file types that are compatible with each log policy is implementation-defined.
- 47296 The *posix_trace_create_withlog()* function shall return in the parameter pointed to by *trid* the trace
47297 stream identifier, which uniquely identifies the newly created trace stream, and shall be used in
47298 subsequent calls to control tracing. The *trid* argument may only be used by the following
47299 functions:
- | | | |
|-------|---|---|
| 47300 | <i>posix_trace_clear()</i> | <i>posix_trace_get_status()</i> |
| 47301 | <i>posix_trace_eventid_equal()</i> | <i>posix_trace_getnext_event()</i> |
| 47302 | <i>posix_trace_eventid_get_name()</i> | <i>posix_trace_shutdown()</i> |
| 47303 | <i>posix_trace_eventtypelist_getnext_id()</i> | <i>posix_trace_start()</i> |
| 47304 | <i>posix_trace_eventtypelist_rewind()</i> | <i>posix_trace_stop()</i> |
| 47305 | <i>posix_trace_flush()</i> | <i>posix_trace_timedgetnext_event()</i> |
| 47306 | <i>posix_trace_get_attr()</i> | <i>posix_trace_trid_eventid_open()</i> |
- 47307 TEF TRL If the Trace Event Filter option is supported, the following additional functions may use the *trid*
47308 argument:
- 47309 *posix_trace_get_filter()* *posix_trace_set_filter()*
- 47310 TRL In particular, notice that the operations normally used by a trace analyzer process, such as
47311 *posix_trace_rewind()* or *posix_trace_close()*, cannot be invoked using the trace stream identifier
47312 returned by the *posix_trace_create_withlog()* function.
- 47313 The *posix_trace_flush()* function shall initiate a flush operation which copies the contents of the
47314 trace stream identified by the argument *trid* into the trace log associated with the trace stream at
47315 the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this
47316 function shall return an error. The termination of the flush operation can be polled by the
47317 *posix_trace_get_status()* function. During the flush operation, it shall be possible to trace new
47318 trace events up to the point when the trace stream becomes full. After flushing is completed, the
47319 space used by the flushed trace events shall be available for tracing new trace events.
- 47320 If flushing the trace stream causes the resulting trace log to become full, the trace log full policy
47321 shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:
- 47322 POSIX_TRACE_UNTIL_FULL
47323 The trace events that have not yet been flushed shall be discarded.
- 47324 POSIX_TRACE_LOOP
47325 The trace events that have not yet been flushed shall be written to the beginning of the trace
47326 log, overwriting previous trace events stored there.

47327

POSIX_TRACE_APPEND

47328

The trace events that have not yet been flushed shall be appended to the trace log.

47329

The *posix_trace_shutdown()* function shall stop the tracing of trace events in the trace stream identified by *trid*, as if *posix_trace_stop()* had been invoked. The *posix_trace_shutdown()* function shall free all the resources associated with the trace stream.

47330

47331

47332

The *posix_trace_shutdown()* function shall not return until all the resources associated with the trace stream have been freed. When the *posix_trace_shutdown()* function returns, the *trid* argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the *trace_getnext_event()* functions (which specified this *trid*) before this call is unblocked with the error [EINVAL].

47333

47334

47335

47336

47337

47338

If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the *posix_trace_shutdown()* function.

47339

47340

47341 TRL

For an active trace stream with log, when the *posix_trace_shutdown()* function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the *posix_trace_flush()* function, and the trace log shall be closed.

47342

47343

47344

When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.

47345

47346

47347

47348

In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the *posix_trace_open()* operation.

47349

47350

The *posix_trace_shutdown()* function shall not return until all trace events have been flushed.

RETURN VALUE

47351

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

47352

47353

47354 TRL

The *posix_trace_create()* and *posix_trace_create_withlog()* functions store the trace stream identifier value in the object pointed to by *trid*, if successful.

47355

ERRORS

47356

47357 TRL

The *posix_trace_create()* and *posix_trace_create_withlog()* functions shall fail if:

47358

[EAGAIN] No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.

47359

47360

[EINTR] The operation was interrupted by a signal. No trace stream was created.

47361

[EINVAL] One or more of the trace parameters specified by the *attr* parameter is invalid.

47362

[ENOMEM] The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.

47363

47364

[EPERM] The caller does not have appropriate privilege to trace the process specified by *pid*.

47365

47366

[ESRCH] The *pid* argument does not refer to an existing process.

47367 TRL

The *posix_trace_create_withlog()* function shall fail if:

47368

[EBADF] The *file_desc* argument is not a valid file descriptor open for writing.

47369	[EINVAL]	The <i>file_desc</i> argument refers to a file with a file type that does not support the log policy associated with the trace log.
47370		
47371	[ENOSPC]	No space left on device. The device corresponding to the argument <i>file_desc</i> does not contain the space required to create this trace log.
47372		

47373 TRL The *posix_trace_flush()* and *posix_trace_shutdown()* functions shall fail if:

47374	[EINVAL]	The value of the <i>trid</i> argument does not correspond to an active trace stream with log.
47375		
47376	[EFBIG]	The trace log file has attempted to exceed an implementation-defined maximum file size.
47377		
47378	[ENOSPC]	No space left on device.

EXAMPLES

47379 None.

APPLICATION USAGE

47380 None.

RATIONALE

47381 None.

FUTURE DIRECTIONS

47382 The *posix_trace_create()*, *posix_trace_create_withlog()*, *posix_trace_flush()*, and *posix_trace_shutdown()* functions may be removed in a future version.

SEE ALSO

47383 *clock_getres()*, *exec*, *posix_trace_attr_destroy()*, *posix_trace_clear()*, *posix_trace_close()*,
 47384 *posix_trace_eventid_equal()*, *posix_trace_eventtypelist_getnext_id()*, *posix_trace_get_attr()*,
 47385 *posix_trace_get_filter()*, *posix_trace_getnext_event()*, *posix_trace_start()*, *posix_trace_start()*, *time*

47386 XBD <sys/types.h>, <trace.h>

CHANGE HISTORY

47387 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

Issue 7

47388 These functions are marked obsolescent.

47389 SD5-XSH-ERN-154 is applied, updating the DESCRIPTION to remove the
 47390 *posix_trace_trygetnext_event()* function from the list of functions that use the *trid* argument.

47399 **NAME**

47400 `posix_trace_event`, `posix_trace_eventid_open` — trace functions for instrumenting application
 47401 code (**TRACING**)

47402 **SYNOPSIS**

```
47403 OB TRC #include <sys/types.h>
47404 #include <trace.h>
47405
47406 void posix_trace_event(trace_event_id_t event_id,
47407                       const void *restrictdata_ptr, size_t data_len);
47407 int posix_trace_eventid_open(const char *restrict event_name,
47408                             trace_event_id_t *restrict event_id);
```

47409 **DESCRIPTION**

47410 The `posix_trace_event()` function shall record the `event_id` and the user data pointed to by `data_ptr`
 47411 in the trace stream into which the calling process is being traced and in which `event_id` is not
 47412 filtered out. If the total size of the user trace event data represented by `data_len` is not greater
 47413 than the declared maximum size for user trace event data, then the `truncation-status` attribute of
 47414 the trace event recorded is `POSIX_TRACE_NOT_TRUNCATED`. Otherwise, the user trace event
 47415 data is truncated to this declared maximum size and the `truncation-status` attribute of the trace
 47416 event recorded is `POSIX_TRACE_TRUNCATED_RECORD`.

47417 If there is no trace stream created for the process or if the created trace stream is not running, or
 47418 if the trace event specified by `event_id` is filtered out in the trace stream, the `posix_trace_event()`
 47419 function shall have no effect.

47420 The `posix_trace_eventid_open()` function shall associate a user trace event name with a trace event
 47421 type identifier for the calling process. The trace event name is the string pointed to by the
 47422 argument `event_name`. It shall have a maximum of `{TRACE_EVENT_NAME_MAX}` characters
 47423 (which has the minimum value `{_POSIX_TRACE_EVENT_NAME_MAX}`). The number of user
 47424 trace event type identifiers that can be defined for any given process is limited by the maximum
 47425 value `{TRACE_USER_EVENT_MAX}`, which has the minimum value
 47426 `{POSIX_TRACE_USER_EVENT_MAX}`.

47427 If the Trace Inherit option is not supported, the `posix_trace_eventid_open()` function shall associate
 47428 the user trace event name pointed to by the `event_name` argument with a trace event type
 47429 identifier that is unique for the traced process, and is returned in the variable pointed to by the
 47430 `event_id` argument. If the user trace event name has already been mapped for the traced process,
 47431 then the previously assigned trace event type identifier shall be returned. If the per-process user
 47432 trace event name limit represented by `{TRACE_USER_EVENT_MAX}` has been reached, the pre-
 47433 defined `POSIX_TRACE_UNNAMED_USEREVENT` (see [Table 2-7](#), on page 517) user trace event
 47434 shall be returned.

47435 TRI If the Trace Inherit option is supported, the `posix_trace_eventid_open()` function shall associate the
 47436 user trace event name pointed to by the `event_name` argument with a trace event type identifier
 47437 that is unique for all the processes being traced in this same trace stream, and is returned in the
 47438 variable pointed to by the `event_id` argument. If the user trace event name has already been
 47439 mapped for the traced processes, then the previously assigned trace event type identifier shall be
 47440 returned. If the per-process user trace event name limit represented by
 47441 `{TRACE_USER_EVENT_MAX}` has been reached, the pre-defined
 47442 `POSIX_TRACE_UNNAMED_USEREVENT` ([Table 2-7](#), on page 517) user trace event shall be
 47443 returned.

Note: The above procedure, together with the fact that multiple processes can only be traced into the same trace stream by inheritance, ensure that all the processes that are traced into a trace stream have the same mapping of trace event names to trace event type identifiers.

47444
4744547446
47447

If there is no trace stream created, the *posix_trace_eventid_open()* function shall store this information for future trace streams created for this process.

47448
47449**RETURN VALUE**

No return value is defined for the *posix_trace_event()* function.

47450
47451
47452
47453

Upon successful completion, the *posix_trace_eventid_open()* function shall return a value of zero. Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open()* function stores the trace event type identifier value in the object pointed to by *event_id*, if successful.

47454
47455**ERRORS**

The *posix_trace_eventid_open()* function shall fail if:

47456
47457
47458

[ENAMETOOLONG]

The size of the name pointed to by the *event_name* argument was longer than the implementation-defined value {TRACE_EVENT_NAME_MAX}.

47459
47460**EXAMPLES**

None.

47461
47462**APPLICATION USAGE**

None.

47463
47464**RATIONALE**

None.

47465
47466**FUTURE DIRECTIONS**

The *posix_trace_event()* and *posix_trace_eventid_open()* functions may be removed in a future version.

47468
47469**SEE ALSO**

Table 2-7 (on page 517), *exec*, *posix_trace_eventid_equal()*, *posix_trace_start()*

47470

XBD <sys/types.h>, <trace.h>

47471
47472**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47473

IEEE PASC Interpretation 1003.1 #123 is applied.

47474
47475

IEEE PASC Interpretation 1003.1 #127 is applied, correcting some editorial errors in the names of the *posix_trace_eventid_open()* function and the *event_id* argument.

47476
47477**Issue 7**

The *posix_trace_event()* and *posix_trace_eventid_open()* functions are marked obsolescent.

posix_trace_eventid_equal()

System Interfaces

47478 **NAME**

47479 `posix_trace_eventid_equal`, `posix_trace_eventid_get_name`, `posix_trace_trid_eventid_open` —
 47480 manipulate the trace event type identifier (**TRACING**)

47481 **SYNOPSIS**

```
47482 OB TRC #include <trace.h>
47483
47483 int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
47484 trace_event_id_t event2);
47485 int posix_trace_eventid_get_name(trace_id_t trid,
47486 trace_event_id_t event, char *event_name);
47487 TEF int posix_trace_trid_eventid_open(trace_id_t trid,
47488 const char *restrict event_name,
47489 trace_event_id_t *restrict event);
```

47490 **DESCRIPTION**

47491 The `posix_trace_eventid_equal()` function shall compare the trace event type identifiers `event1` and
 47492 `event2` from the same trace stream or the same trace log identified by the `trid` argument. If the
 47493 trace event type identifiers `event1` and `event2` are from different trace streams, the return value
 47494 shall be unspecified.

47495 The `posix_trace_eventid_get_name()` function shall return, in the argument pointed to by
 47496 `event_name`, the trace event name associated with the trace event type identifier identified by the
 47497 argument `event`, for the trace stream or for the trace log identified by the `trid` argument. The
 47498 name of the trace event shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 47499 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). Successive calls to
 47500 this function with the same trace event type identifier and the same trace stream identifier shall
 47501 return the same event name.

47502 TEF The `posix_trace_trid_eventid_open()` function shall associate a user trace event name with a trace
 47503 event type identifier for a given trace stream. The trace stream is identified by the `trid` argument,
 47504 and it shall be an active trace stream. The trace event name is the string pointed to by the
 47505 argument `event_name`. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters
 47506 (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user
 47507 trace event type identifiers that can be defined for any given process is limited by the maximum
 47508 value {TRACE_USER_EVENT_MAX}, which has the minimum value
 47509 {_POSIX_TRACE_USER_EVENT_MAX}.

47510 If the Trace Inherit option is not supported, the `posix_trace_trid_eventid_open()` function shall
 47511 associate the user trace event name pointed to by the `event_name` argument with a trace event
 47512 type identifier that is unique for the process being traced in the trace stream identified by the `trid`
 47513 argument, and is returned in the variable pointed to by the `event` argument. If the user trace
 47514 event name has already been mapped for the traced process, then the previously assigned trace
 47515 event type identifier shall be returned. If the per-process user trace event name limit represented
 47516 by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
 47517 POSIX_TRACE_UNNAMED_USEREVENT (see [Table 2-7](#), on page 517) user trace event shall be
 47518 returned.

47519 TEF TRI If the Trace Inherit option is supported, the `posix_trace_trid_eventid_open()` function shall
 47520 associate the user trace event name pointed to by the `event_name` argument with a trace event
 47521 type identifier that is unique for all the processes being traced in the trace stream identified by
 47522 the `trid` argument, and is returned in the variable pointed to by the `event` argument. If the user
 47523 trace event name has already been mapped for the traced processes, then the previously
 47524 assigned trace event type identifier shall be returned. If the per-process user trace event name

47525 limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined
 47526 POSIX_TRACE_UNNAMED_USEREVENT (see [Table 2-7](#), on page 517) user trace event shall be
 47527 returned.

47528 RETURN VALUE

47529 TEF Upon successful completion, the `posix_trace_eventid_get_name()` and
 47530 `posix_trace_trid_eventid_open()` functions shall return a value of zero. Otherwise, they shall return
 47531 the corresponding error number.

47532 The `posix_trace_eventid_equal()` function shall return a non-zero value if *event1* and *event2* are
 47533 equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or
 47534 *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid*
 47535 is invalid, the behavior shall be unspecified.

47536 The `posix_trace_eventid_get_name()` function stores the trace event name value in the object
 47537 pointed to by *event_name*, if successful.

47538 TEF The `posix_trace_trid_eventid_open()` function stores the trace event type identifier value in the
 47539 object pointed to by *event*, if successful.

47540 ERRORS

47541 TEF The `posix_trace_eventid_get_name()` and `posix_trace_trid_eventid_open()` functions shall fail if:
 47542 [EINVAL] The *trid* argument was not a valid trace stream identifier.

47543 TEF The `posix_trace_trid_eventid_open()` function shall fail if:

47544 TEF [ENAMETOOLONG]

47545 The size of the name pointed to by the *event_name* argument was longer than
 47546 the implementation-defined value {TRACE_EVENT_NAME_MAX}.

47547 The `posix_trace_eventid_get_name()` function shall fail if:

47548 [EINVAL] The trace event type identifier *event* was not associated with any name.

47549 EXAMPLES

47550 None.

47551 APPLICATION USAGE

47552 None.

47553 RATIONALE

47554 None.

47555 FUTURE DIRECTIONS

47556 The `posix_trace_eventid_equal()`, `posix_trace_eventid_get_name()`, and
 47557 `posix_trace_trid_eventid_open()` functions may be removed in a future version. |

47558 SEE ALSO

47559 [Table 2-7](#) (on page 517), `exec`, `posix_trace_event()`, `posix_trace_getnext_event()`

47560 XBD [<trace.h>](#) +

47561 CHANGE HISTORY

47562 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47563 IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

47564 Issue 7

47565 These functions are marked obsolescent.

posix_trace_eventid_open()*System Interfaces*47566 **NAME**47567 posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)47568 **SYNOPSIS**

```
47569 OB TRC #include <sys/types.h>  
47570 #include <trace.h>  
  
47571 int posix_trace_eventid_open(const char *restrict event_name,  
47572 trace_event_id_t *restrict event_id);
```

47573 **DESCRIPTION**47574 Refer to [posix_trace_event\(\)](#).

47575 **NAME**

47576 `posix_trace_eventset_add`, `posix_trace_eventset_del`, `posix_trace_eventset_empty`,
 47577 `posix_trace_eventset_fill`, `posix_trace_eventset_ismember` — manipulate trace event type sets
 47578 (**TRACING**)

47579 **SYNOPSIS**

```
47580 OB TRC #include <trace.h>
47581 TEF int posix_trace_eventset_add(trace_event_id_t event_id,
47582     trace_event_set_t *set);
47583 int posix_trace_eventset_del(trace_event_id_t event_id,
47584     trace_event_set_t *set);
47585 int posix_trace_eventset_empty(trace_event_set_t *set);
47586 int posix_trace_eventset_fill(trace_event_set_t *set, int what);
47587 int posix_trace_eventset_ismember(trace_event_id_t event_id,
47588     const trace_event_set_t *restrict set, int *restrict ismember);
```

47589 **DESCRIPTION**

47590 These primitives manipulate sets of trace event types. They operate on data objects addressable
 47591 by the application, not on the current trace event filter of any trace stream.

47592 The `posix_trace_eventset_add()` and `posix_trace_eventset_del()` functions, respectively, shall add or
 47593 delete the individual trace event type specified by the value of the argument `event_id` to or from
 47594 the trace event type set pointed to by the argument `set`. Adding a trace event type already in the
 47595 set or deleting a trace event type not in the set shall not be considered an error.

47596 The `posix_trace_eventset_empty()` function shall initialize the trace event type set pointed to by
 47597 the `set` argument such that all trace event types defined, both system and user, shall be excluded
 47598 from the set.

47599 The `posix_trace_eventset_fill()` function shall initialize the trace event type set pointed to by the
 47600 argument `set`, such that the set of trace event types defined by the argument `what` shall be
 47601 included in the set. The value of the argument `what` shall consist of one of the following values,
 47602 as defined in the `<trace.h>` header:

47603 **POSIX_TRACE_WOPID_EVENTS**

47604 All the process-independent implementation-defined system trace event types are included
 47605 in the set.

47606 **POSIX_TRACE_SYSTEM_EVENTS**

47607 All the implementation-defined system trace event types are included in the set, as are those
 47608 defined in POSIX.1-200x.

47609 **POSIX_TRACE_ALL_EVENTS**

47610 All trace event types defined, both system and user, are included in the set.

47611 Applications shall call either `posix_trace_eventset_empty()` or `posix_trace_eventset_fill()` at least
 47612 once for each object of type `trace_event_set_t` prior to any other use of that object. If such an
 47613 object is not initialized in this way, but is nonetheless supplied as an argument to any of the
 47614 `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, or `posix_trace_eventset_ismember()` functions,
 47615 the results are undefined.

47616 The `posix_trace_eventset_ismember()` function shall test whether the trace event type specified by
 47617 the value of the argument `event_id` is a member of the set pointed to by the argument `set`. The
 47618 value returned in the object pointed to by `ismember` argument is zero if the trace event type
 47619 identifier is not a member of the set and a value different from zero if it is a member of the set.

posix_trace_eventset_add()*System Interfaces*47620 **RETURN VALUE**

47621 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
47622 return the corresponding error number.

47623 **ERRORS**

47624 These functions may fail if:

47625 [EINVAL] The value of one of the arguments is invalid.

47626 **EXAMPLES**

47627 None.

47628 **APPLICATION USAGE**

47629 None.

47630 **RATIONALE**

47631 None.

47632 **FUTURE DIRECTIONS**

47633 The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,
47634 `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions may be removed in a
47635 future version.

47636 **SEE ALSO**

47637 [*posix_trace_eventid_equal\(\)*](#), [*posix_trace_get_filter\(\)*](#)

47638 XBD [**<trace.h>**](#)

47639 **CHANGE HISTORY**

47640 First released in Issue 6. Derived from IEEE Std. 1003.1q-2000.

47641 **Issue 7**

47642 The `posix_trace_eventset_add()`, `posix_trace_eventset_del()`, `posix_trace_eventset_empty()`,
47643 `posix_trace_eventset_fill()`, and `posix_trace_eventset_ismember()` functions are marked obsolescent.

47644 **NAME**

47645 `posix_trace_eventtypelist_getnext_id`, `posix_trace_eventtypelist_rewind` — iterate over a
 47646 mapping of trace event types (**TRACING**)

47647 **SYNOPSIS**

```
47648 OB TRC #include <trace.h>
47649
47649 int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
47650     trace_event_id_t *restrict event, int *restrict unavailable);
47651 int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

47652 **DESCRIPTION**

47653 The first time `posix_trace_eventtypelist_getnext_id()` is called, the function shall return in the
 47654 variable pointed to by `event` the first trace event type identifier of the list of trace events of the
 47655 trace stream identified by the `trid` argument. Successive calls to
 47656 `posix_trace_eventtypelist_getnext_id()` return in the variable pointed to by `event` the next trace
 47657 event type identifier in that same list. Each time a trace event type identifier is successfully
 47658 written into the variable pointed to by the `event` argument, the variable pointed to by the
 47659 `unavailable` argument shall be set to zero. When no more trace event type identifiers are available,
 47660 and so none is returned, the variable pointed to by the `unavailable` argument shall be set to a
 47661 value different from zero.

47662 The `posix_trace_eventtypelist_rewind()` function shall reset the next trace event type identifier to
 47663 be read to the first trace event type identifier from the list of trace events used in the trace stream
 47664 identified by `trid`.

47665 **RETURN VALUE**

47666 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47667 return the corresponding error number.

47668 The `posix_trace_eventtypelist_getnext_id()` function stores the trace event type identifier value in
 47669 the object pointed to by `event`, if successful.

47670 **ERRORS**

47671 These functions shall fail if:

47672 [EINVAL] The `trid` argument was not a valid trace stream identifier.

47673 **EXAMPLES**

47674 None.

47675 **APPLICATION USAGE**

47676 None.

47677 **RATIONALE**

47678 None.

47679 **FUTURE DIRECTIONS**

47680 The `posix_trace_eventtypelist_getnext_id()` and `posix_trace_eventtypelist_rewind()` functions may be
 47681 removed in a future version.

47682 **SEE ALSO**

47683 [*posix_trace_event\(\)*](#), [*posix_trace_eventid_equal\(\)*](#), [*posix_trace_getnext_event\(\)*](#)

47684 XBD [**<trace.h>**](#)

+

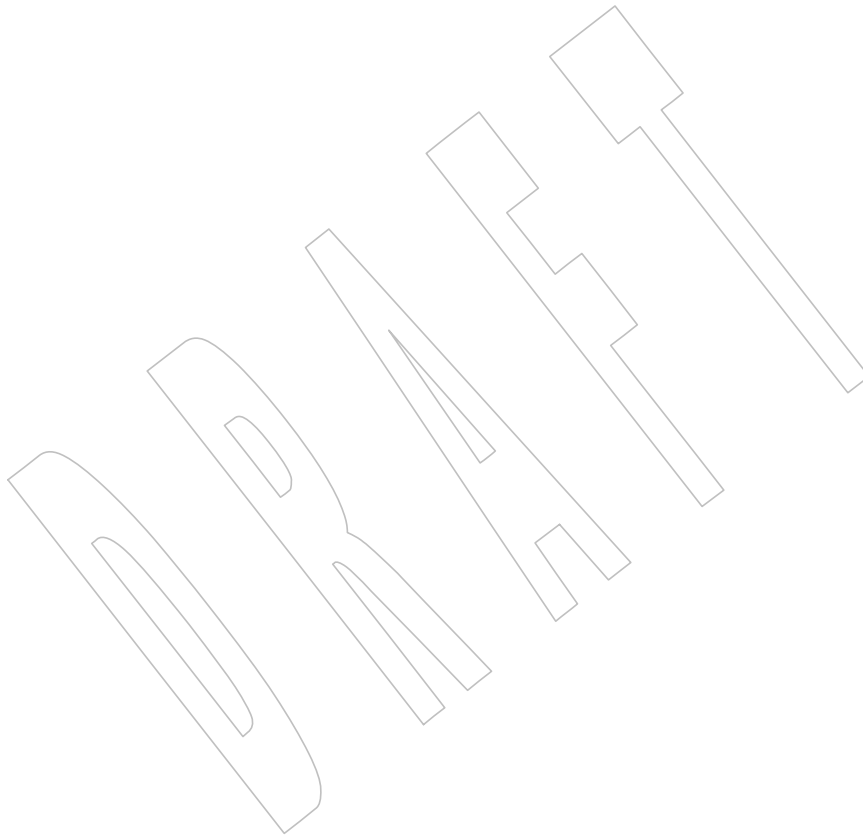
47685
47686
47687
47688
47689
47690**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

IEEE PASC Interpretations 1003.1 #123 and #129 are applied.

Issue 7

The *posix_trace_eventtypelist_getnext_id()* and *posix_trace_eventtypelist_rewind()* functions are marked obsolescent.



47691 **NAME**
47692 `posix_trace_flush` — trace stream flush from a process (**TRACING**)

47693 **SYNOPSIS**

```
47694 OB TRC #include <sys/types.h>  
47695 #include <trace.h>  
47696 TRL int posix_trace_flush(trace_id_t trid);
```

47697 **DESCRIPTION**

47698 Refer to [posix_trace_create\(\)](#).

47699 **NAME**

47700 posix_trace_get_attr, posix_trace_get_status — retrieve the trace attributes or trace status
 47701 (TRACING)

47702 **SYNOPSIS**

```
47703 OB TRC #include <trace.h>
47704
47704 int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
47705 int posix_trace_get_status(trace_id_t trid,
47706 struct posix_trace_status_info *statusinfo);
```

47707 **DESCRIPTION**

47708 The *posix_trace_get_attr()* function shall copy the attributes of the active trace stream identified
 47709 TRL by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid*
 47710 may represent a pre-recorded trace log.

47711 The *posix_trace_get_status()* function shall return, in the structure pointed to by the *statusinfo*
 47712 argument, the current trace status for the trace stream identified by the *trid* argument. These
 47713 status values returned in the structure pointed to by *statusinfo* shall have been appropriately
 47714 TRL read to ensure that the returned values are consistent. If the Trace Log option is supported and
 47715 the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the
 47716 completed trace stream.

47717 Each time the *posix_trace_get_status()* function is used, the overrun status of the trace stream
 47718 TRL shall be reset to POSIX_TRACE_NO_OVERRUN immediately after the call completes. If the
 47719 Trace Log option is supported, the *posix_trace_get_status()* function shall behave the same as
 47720 when the option is not supported except for the following differences:

- 47721 • If the *trid* argument refers to a trace stream with log, each time the *posix_trace_get_status()*
 47722 function is used, the log overrun status of the trace stream shall be reset to
 47723 POSIX_TRACE_NO_OVERRUN and the *flush_error* status shall be reset to zero
 47724 immediately after the call completes.
- 47725 • If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the
 47726 status of the completed trace stream and the status values of the trace stream shall not be
 47727 reset.

47728 **RETURN VALUE**

47729 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47730 return the corresponding error number.

47731 The *posix_trace_get_attr()* function stores the trace attributes in the object pointed to by *attr*, if
 47732 successful.

47733 The *posix_trace_get_status()* function stores the trace status in the object pointed to by *statusinfo*,
 47734 if successful.

47735 **ERRORS**

47736 These functions shall fail if:

- 47737 [EINVAL] The trace stream argument *trid* does not correspond to a valid active trace
 47738 stream or a valid trace log.

47739

EXAMPLES

47740

None.

47741

APPLICATION USAGE

47742

None.

47743

RATIONALE

47744

None.

47745

FUTURE DIRECTIONS

47746

The *posix_trace_get_attr()* and *posix_trace_get_status()* functions may be removed in a future version.

47747

47748

SEE ALSO

47749

posix_trace_attr_destroy(), *posix_trace_close()*, *posix_trace_create()*

47750

XBD <trace.h>

+

47751

CHANGE HISTORY

47752

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

47753

IEEE PASC Interpretation 1003.1 #123 is applied.

47754

Issue 7

47755

The *posix_trace_get_attr()* and *posix_trace_get_status()* functions are marked obsolescent.

DRAFT

47756 **NAME**

47757 `posix_trace_get_filter`, `posix_trace_set_filter` — retrieve and set the filter of an initialized trace
 47758 stream (**TRACING**)

47759 **SYNOPSIS**

```
47760 OB TRC #include <trace.h>
47761 TEF int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
47762 int posix_trace_set_filter(trace_id_t trid,
47763 const trace_event_set_t *set, int how);
```

47764 **DESCRIPTION**

47765 The `posix_trace_get_filter()` function shall retrieve, into the argument pointed to by *set*, the actual
 47766 trace event filter from the trace stream specified by *trid*.

47767 The `posix_trace_set_filter()` function shall change the set of filtered trace event types after a trace
 47768 stream identified by the *trid* argument is created. This function may be called prior to starting
 47769 the trace stream, or while the trace stream is active. By default, if no call is made to
 47770 `posix_trace_set_filter()`, all trace events shall be recorded (that is, none of the trace event types are
 47771 filtered out).

47772 If this function is called while the trace is in progress, a special system trace event,
 47773 `POSIX_TRACE_FILTER`, shall be recorded in the trace indicating both the old and the new sets
 47774 of filtered trace event types (see [Table 2-4](#) (on page 515) and [Table 2-6](#), on page 516).

47775 If the `posix_trace_set_filter()` function is interrupted by a signal, an error shall be returned and the
 47776 filter shall not be changed. In this case, the state of the trace stream shall not be changed.

47777 The value of the argument *how* indicates the manner in which the set is to be changed and shall
 47778 have one of the following values, as defined in the `<trace.h>` header:

47779 `POSIX_TRACE_SET_EVENTSET`

47780 The resulting set of trace event types to be filtered shall be the trace event type set pointed
 47781 to by the argument *set*.

47782 `POSIX_TRACE_ADD_EVENTSET`

47783 The resulting set of trace event types to be filtered shall be the union of the current set and
 47784 the trace event type set pointed to by the argument *set*.

47785 `POSIX_TRACE_SUB_EVENTSET`

47786 The resulting set of trace event types to be filtered shall be all trace event types in the
 47787 current set that are not in the set pointed to by the argument *set*; that is, remove each
 47788 element of the specified set from the current filter.

47789 **RETURN VALUE**

47790 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47791 return the corresponding error number.

47792 The `posix_trace_get_filter()` function stores the set of filtered trace event types in *set*, if successful.

47793 **ERRORS**

47794 These functions shall fail if:

47795 `[EINVAL]` The value of the *trid* argument does not correspond to an active trace stream
 47796 or the value of the argument pointed to by *set* is invalid.

47797 [EINTR] The operation was interrupted by a signal.

EXAMPLES

47798 None.
47799

APPLICATION USAGE

47800 None.
47801

RATIONALE

47802 None.
47803

FUTURE DIRECTIONS

47804 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions may be removed in a future
47805 version.
47806

SEE ALSO

47807 [Table 2-4](#) (on page 515), [Table 2-6](#) (on page 516), [posix_trace_eventset_add\(\)](#)
47808

47809 XBD [<trace.h>](#)

CHANGE HISTORY

47810 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
47811

47812 IEEE PASC Interpretation 1003.1 #123 is applied.

Issue 7

47813 The *posix_trace_get_filter()* and *posix_trace_set_filter()* functions are marked obsolescent.
47814

DRAFT

posix_trace_get_status()47815 **NAME**47816 `posix_trace_get_status` — retrieve the trace status (**TRACING**)47817 **SYNOPSIS**47818 OB TRC `#include <trace.h>`47819 `int posix_trace_get_status(trace_id_t trid,`
47820 `struct posix_trace_status_info *statusinfo);`47821 **DESCRIPTION**47822 Refer to [*posix_trace_get_attr\(\)*](#).

47823 **NAME**

47824 `posix_trace_getnext_event`, `posix_trace_timedgetnext_event`, `posix_trace_trygetnext_event` —
 47825 retrieve a trace event (**TRACING**)

47826 **SYNOPSIS**

```
47827 OB TRC #include <sys/types.h>
47828 #include <trace.h>

47829 int posix_trace_getnext_event(trace_id_t trid,
47830     struct posix_trace_event_info *restrict event,
47831     void *restrict data, size_t num_bytes,
47832     size_t *restrict data_len, int *restrict unavailable);
47833 int posix_trace_timedgetnext_event(trace_id_t trid,
47834     struct posix_trace_event_info *restrict event,
47835     void *restrict data, size_t num_bytes,
47836     size_t *restrict data_len, int *restrict unavailable,
47837     const struct timespec *restrict abstime);
47838 int posix_trace_trygetnext_event(trace_id_t trid,
47839     struct posix_trace_event_info *restrict event,
47840     void *restrict data, size_t num_bytes,
47841     size_t *restrict data_len, int *restrict unavailable);
```

47842 **DESCRIPTION**

47843 The `posix_trace_getnext_event()` function shall report a recorded trace event either from an active
 47844 TRL trace stream without log or a pre-recorded trace stream identified by the `trid` argument. The
 47845 `posix_trace_trygetnext_event()` function shall report a recorded trace event from an active trace
 47846 stream without log identified by the `trid` argument.

47847 The trace event information associated with the recorded trace event shall be copied by the
 47848 function into the structure pointed to by the argument `event` and the data associated with the
 47849 trace event shall be copied into the buffer pointed to by the `data` argument.

47850 The `posix_trace_getnext_event()` function shall block if the `trid` argument identifies an active trace
 47851 stream and there is currently no trace event ready to be retrieved. When returning, if a recorded
 47852 trace event was reported, the variable pointed to by the `unavailable` argument shall be set to zero.
 47853 Otherwise, the variable pointed to by the `unavailable` argument shall be set to a value different
 47854 from zero.

47855 The `posix_trace_timedgetnext_event()` function shall attempt to get another trace event from an
 47856 active trace stream without log, as in the `posix_trace_getnext_event()` function. However, if no
 47857 trace event is available from the trace stream, the implied wait shall be terminated when the
 47858 timeout specified by the argument `abstime` expires, and the function shall return the error
 47859 [ETIMEDOUT].

47860 The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the
 47861 clock upon which timeouts are based (that is, when the value of that clock equals or exceeds
 47862 `abstime`), or if the absolute time specified by `abstime` has already passed at the time of the call.

47863 The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall
 47864 be the resolution of the clock on which it is based. The `timespec` data type is defined in the
 47865 `<time.h>` header.

47866 Under no circumstance shall the function fail with a timeout if a trace event is immediately
 47867 available from the trace stream. The validity of the `abstime` argument need not be checked if a
 47868 trace event is immediately available from the trace stream.

47869 The behavior of this function for a pre-recorded trace stream is unspecified.

47870 TRL The *posix_trace_trygetnext_event()* function shall not block. This function shall return an error if
 47871 the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported,
 47872 the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace
 47873 event was reported, the variable pointed to by the *unavailable* argument shall be set to a value
 47874 different from zero.

47875 The argument *num_bytes* shall be the size of the buffer pointed to by the *data* argument. The
 47876 argument *data_len* reports to the application the length in bytes of the data record just
 47877 transferred. If *num_bytes* is greater than or equal to the size of the data associated with the trace
 47878 event pointed to by the *event* argument, all the recorded data shall be transferred. In this case,
 47879 the *truncation-status* member of the trace event structure shall be either
 47880 POSIX_TRACE_NOT_TRUNCATED, if the trace event data was recorded without truncation
 47881 while tracing, or POSIX_TRACE_TRUNCATED_RECORD, if the trace event data was truncated
 47882 when it was recorded. If the *num_bytes* argument is less than the length of recorded trace event
 47883 data, the data transferred shall be truncated to a length of *num_bytes*, the value stored in the
 47884 variable pointed to by *data_len* shall be equal to *num_bytes*, and the *truncation-status* member of
 47885 the *event* structure argument shall be set to POSIX_TRACE_TRUNCATED_READ (see the
 47886 **posix_trace_event_info** structure defined in <trace.h>).

47887 The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace
 47888 events shall be reported in the order in which they were generated, up to an implementation-
 47889 defined time resolution that causes the ordering of trace events occurring very close to each
 47890 other to be unknown. Once reported, a trace event cannot be reported again from an active trace
 47891 stream. Once a trace event is reported from an active trace stream without log, the trace stream
 47892 shall make the resources associated with that trace event available to record future generated
 47893 trace events.

47894 RETURN VALUE

47895 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47896 return the corresponding error number.

47897 If successful, these functions store:

- 47898 • The recorded trace event in the object pointed to by *event*
- 47899 • The trace event information associated with the recorded trace event in the object pointed
 47900 to by *data*
- 47901 • The length of this trace event information in the object pointed to by *data_len*
- 47902 • The value of zero in the object pointed to by *unavailable*

47903 ERRORS

47904 These functions shall fail if:

47905 [EINVAL] The trace stream identifier argument *trid* is invalid.

47906 The *posix_trace_getnext_event()* and *posix_trace_timedgetnext_event()* functions shall fail if:

47907 [EINTR] The operation was interrupted by a signal, and so the call had no effect.

47908 The *posix_trace_trygetnext_event()* function shall fail if:

47909 [EINVAL] The trace stream identifier argument *trid* does not correspond to an active
 47910 trace stream.

47911 The *posix_trace_timedgetnext_event()* function shall fail if:

47912 [EINVAL] There is no trace event immediately available from the trace stream, and the
 47913 *timeout* argument is invalid.

47914 [ETIMEDOUT] No trace event was available from the trace stream before the specified
47915 timeout *timeout* expired.

EXAMPLES

47916 None.
47917

APPLICATION USAGE

47918 None.
47919

RATIONALE

47920 None.
47921

FUTURE DIRECTIONS

47922 The `posix_trace_getnext_event()`, `posix_trace_timedgetnext_event()`, and
47923 `posix_trace_trygetnext_event()` functions may be removed in a future version.
47924

SEE ALSO

47925 [*posix_trace_close\(\)*](#), [*posix_trace_create\(\)*](#)
47926

47927 XBD [*<sys/types.h>*](#), [*<trace.h>*](#) |

CHANGE HISTORY

47928 First released in Issue 6. Derived from IEEE Std 1003.1q-2000.
47929

47930 IEEE PASC Interpretation 1003.1 #123 is applied.

Issue 7

47931 The `posix_trace_getnext_event()`, `posix_trace_timedgetnext_event()`, and
47932 `posix_trace_trygetnext_event()` functions are marked obsolescent.
47933

47934 Functionality relating to the Timers option is moved to the Base.

47935 **NAME**47936 posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)47937 **SYNOPSIS**47938 OB TRC `#include <trace.h>`47939 TRL `int posix_trace_open(int file_desc, trace_id_t *trid);`47940 `int posix_trace_rewind(trace_id_t trid);`47941 **DESCRIPTION**47942 Refer to [posix_trace_close\(\)](#).

47943 **NAME**47944 `posix_trace_set_filter` — set filter of an initialized trace stream (**TRACING**)47945 **SYNOPSIS**47946 OB TRC `#include <trace.h>`47947 TEF `int posix_trace_set_filter(trace_id_t trid,`
47948 `const trace_event_set_t *set, int how);`47949 **DESCRIPTION**47950 Refer to [posix_trace_get_filter\(\)](#).

posix_trace_shutdown()*System Interfaces*

47951 **NAME**
47952 `posix_trace_shutdown` — trace stream shutdown from a process (**TRACING**)

SYNOPSIS

```
47954 OB TRC #include <sys/types.h>  
47955 #include <trace.h>  
47956 int posix_trace_shutdown(trace_id_t trid);
```

DESCRIPTION

47957 Refer to [posix_trace_create\(\)](#).
47958

47959 **NAME**47960 `posix_trace_start`, `posix_trace_stop` — trace start and stop (**TRACING**)47961 **SYNOPSIS**

```
47962 OB TRC #include <trace.h>
47963
47963 int posix_trace_start(trace_id_t trid);
47964 int posix_trace_stop (trace_id_t trid);
```

47965 **DESCRIPTION**

47966 The `posix_trace_start()` and `posix_trace_stop()` functions, respectively, shall start and stop the trace
 47967 stream identified by the argument *trid*.

47968 The effect of calling the `posix_trace_start()` function shall be recorded in the trace stream as the
 47969 POSIX_TRACE_START system trace event and the status of the trace stream shall become
 47970 POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the
 47971 POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall
 47972 continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall
 47973 not be recorded and the status of the trace stream shall not be changed.

47974 The effect of calling the `posix_trace_stop()` function shall be recorded in the trace stream as the
 47975 POSIX_TRACE_STOP system trace event and the status of the trace stream shall become
 47976 POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the
 47977 POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain
 47978 suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be
 47979 recorded and the status of the trace stream shall not be changed.

47980 **RETURN VALUE**

47981 Upon successful completion, these functions shall return a value of zero. Otherwise, they shall
 47982 return the corresponding error number.

47983 **ERRORS**

47984 These functions shall fail if:

47985 [EINVAL] The value of the argument *trid* does not correspond to an active trace stream
 47986 and thus no trace stream was started or stopped.

47987 [EINTR] The operation was interrupted by a signal and thus the trace stream was not
 47988 necessarily started or stopped.

47989 **EXAMPLES**

47990 None.

47991 **APPLICATION USAGE**

47992 None.

47993 **RATIONALE**

47994 None.

47995 **FUTURE DIRECTIONS**47996 The `posix_trace_start()` and `posix_trace_stop()` functions may be removed in a future version.47997 **SEE ALSO**47998 [*posix_trace_create\(\)*](#)47999 XBD [**<trace.h>**](#)

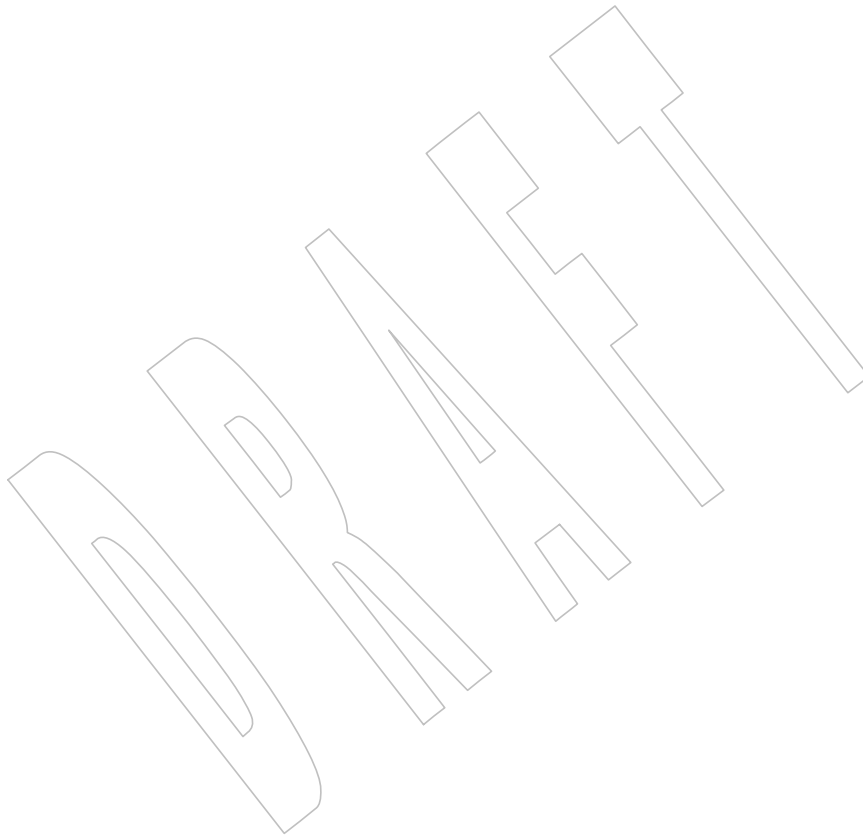
48000
48001
48002
48003
48004**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1q-2000.

IEEE PASC Interpretation 1003.1 #123 is applied.

Issue 7

The *posix_trace_start()* and *posix_trace_stop()* functions are marked obsolescent.



48005 **NAME**
48006 `posix_trace_timedgetnext_event` — retrieve a trace event (**TRACING**)

48007 **SYNOPSIS**

```
48008 OB TRC #include <sys/types.h>  
48009 #include <trace.h>  
  
48010 int posix_trace_timedgetnext_event(trace_id_t trid,  
48011 struct posix_trace_event_info *restrict event,  
48012 void *restrict data, size_t num_bytes,  
48013 size_t *restrict data_len, int *restrict unavailable,  
48014 const struct timespec *restrict abstime);
```

48015 **DESCRIPTION**

48016 Refer to [posix_trace_getnext_event\(\)](#).

48017 **NAME**48018 posix_trace_trid_eventid_open — open a trace event type identifier (**TRACING**)48019 **SYNOPSIS**

48020 OB TRC #include <trace.h>

48021 TEF int posix_trace_trid_eventid_open(trace_id_t trid,
48022 const char *restrict event_name,
48023 trace_event_id_t *restrict event);48024 **DESCRIPTION**48025 Refer to [posix_trace_eventid_equal\(\)](#).

48026 **NAME**
48027 `posix_trace_trygetnext_event` — retrieve a trace event (**TRACING**)

48028 **SYNOPSIS**

```
48029 OB TRC #include <sys/types.h>  
48030 #include <trace.h>  
  
48031 int posix_trace_trygetnext_event(trace_id_t trid,  
48032     struct posix_trace_event_info *restrict event,  
48033     void *restrict data, size_t num_bytes,  
48034     size_t *restrict data_len, int *restrict unavailable);
```

48035 **DESCRIPTION**

48036 Refer to [posix_trace_getnext_event\(\)](#).

48037 **NAME**48038 `posix_typed_mem_get_info` — query typed memory information (**ADVANCED REALTIME**)48039 **SYNOPSIS**

```
48040 TYM #include <sys/mman.h>
48041
48041 int posix_typed_mem_get_info(int fildev,
48042 struct posix_typed_mem_info *info);
```

48043 **DESCRIPTION**

48044 The `posix_typed_mem_get_info()` function shall return, in the `posix_tmi_length` field of the **posix_typed_mem_info** structure pointed to by `info`, the maximum length which may be successfully allocated by the typed memory object designated by `fildev`. This maximum length shall take into account the flag `POSIX_TYPED_MEM_ALLOCATE` or `POSIX_TYPED_MEM_ALLOCATE_CONTIG` specified when the typed memory object represented by `fildev` was opened. The maximum length is dynamic; therefore, the value returned is valid only while the current mapping of the corresponding typed memory pool remains unchanged.

48052 If `fildev` represents a typed memory object opened with neither the `POSIX_TYPED_MEM_ALLOCATE` flag nor the `POSIX_TYPED_MEM_ALLOCATE_CONTIG` flag specified, the returned value of `info->posix_tmi_length` is unspecified.

48055 The `posix_typed_mem_get_info()` function may return additional implementation-defined information in other fields of the **posix_typed_mem_info** structure pointed to by `info`.

48057 If the memory object specified by `fildev` is not a typed memory object, then the behavior of this function is undefined.

48059 **RETURN VALUE**

48060 Upon successful completion, the `posix_typed_mem_get_info()` function shall return zero; otherwise, the corresponding error status value shall be returned.

48062 **ERRORS**

48063 The `posix_typed_mem_get_info()` function shall fail if:

- | | | |
|-------|----------|--|
| 48064 | [EBADF] | The <code>fildev</code> argument is not a valid open file descriptor. |
| 48065 | [ENODEV] | The <code>fildev</code> argument is not connected to a memory object supported by this function. |

48067 This function shall not return an error code of [EINTR].

48068 **EXAMPLES**

48069 None.

48070 **APPLICATION USAGE**

48071 None.

48072 **RATIONALE**

48073 An application that needs to allocate a block of typed memory with length dependent upon the amount of memory currently available must either query the typed memory object to obtain the amount available, or repeatedly invoke `mmap()` attempting to guess an appropriate length. While the latter method is existing practice with `malloc()`, it is awkward and imprecise. The `posix_typed_mem_get_info()` function allows an application to immediately determine available memory. This is particularly important for typed memory objects that may in some cases be scarce resources. Note that when a typed memory pool is a shared resource, some form of mutual-exclusion or synchronization may be required while typed memory is being queried and

48081 allocated to prevent race conditions.

48082 The existing *fstat()* function is not suitable for this purpose. We realize that implementations
48083 may wish to provide other attributes of typed memory objects (for example, alignment
48084 requirements, page size, and so on). The *fstat()* function returns a structure which is not
48085 extensible and, furthermore, contains substantial information that is inappropriate for typed
48086 memory objects.

48087 FUTURE DIRECTIONS

48088 None.

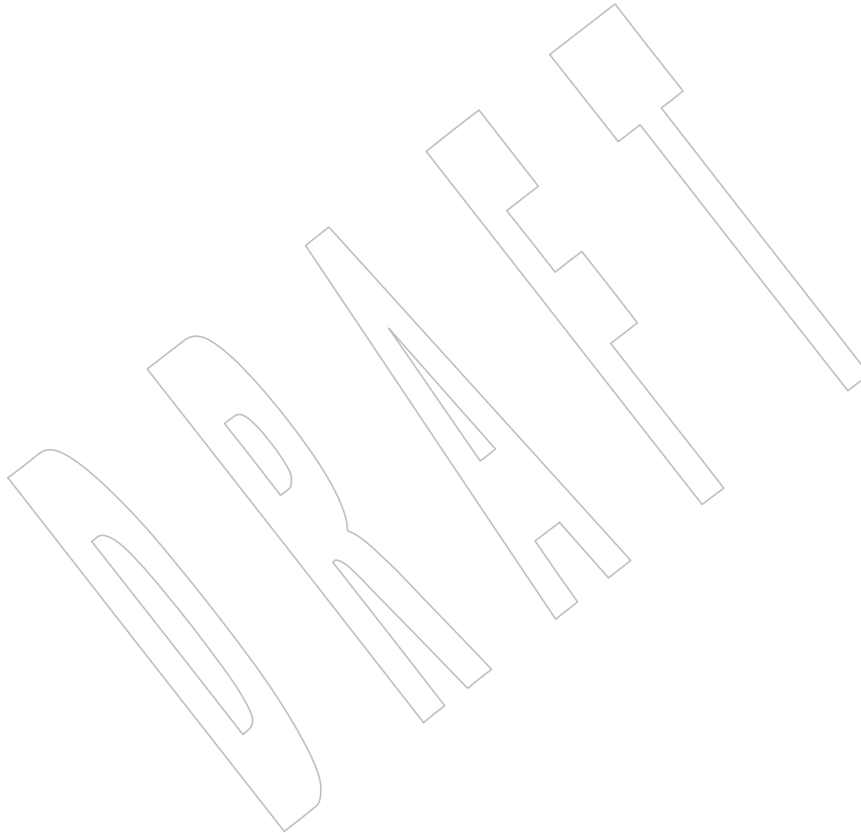
48089 SEE ALSO

48090 *fstat()*, *mmap()*, *posix_typed_mem_open()*

48091 XBD <[sys/mman.h](#)>

48092 CHANGE HISTORY

48093 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.



48094 **NAME**48095 posix_typed_mem_open — open a typed memory object (**ADVANCED REALTIME**)48096 **SYNOPSIS**

```
48097 TYM #include <sys/mman.h>
48098 int posix_typed_mem_open(const char *name, int oflag, int tflag);
```

48099 **DESCRIPTION**

48100 The *posix_typed_mem_open()* function shall establish a connection between the typed memory
 48101 object specified by the string pointed to by *name* and a file descriptor. It shall create an open file
 48102 description that refers to the typed memory object and a file descriptor that refers to that open
 48103 file description. The file descriptor is used by other functions to refer to that typed memory
 48104 object. It is unspecified whether the name appears in the file system and is visible to other
 48105 functions that take pathnames as arguments. The *name* argument shall conform to the
 48106 construction rules for a pathname. If *name* begins with the slash character, then processes calling
 48107 *posix_typed_mem_open()* with the same value of *name* shall refer to the same typed memory
 48108 object. If *name* does not begin with the slash character, the effect is implementation-defined. The
 48109 interpretation of slash characters other than the leading slash character in *name* is
 48110 implementation-defined.

48111 Each typed memory object supported in a system shall be identified by a name which specifies
 48112 not only its associated typed memory pool, but also the path or port by which it is accessed. That
 48113 is, the same typed memory pool accessed via several different ports shall have several different
 48114 corresponding names. The binding between names and typed memory objects is established in
 48115 an implementation-defined manner. Unlike shared memory objects, there is no way within
 48116 POSIX.1-200x for a program to create a typed memory object.

48117 The value of *tflag* shall determine how the typed memory object behaves when subsequently
 48118 mapped by calls to *mmap()*. At most, one of the following flags defined in **<sys/mman.h>** may
 48119 be specified:

48120 POSIX_TYPED_MEM_ALLOCATE
 48121 Allocate on *mmap()*.

48122 POSIX_TYPED_MEM_ALLOCATE_CONTIG
 48123 Allocate contiguously on *mmap()*.

48124 POSIX_TYPED_MEM_MAP_ALLOCATABLE
 48125 Map on *mmap()*, without affecting allocatability.

48126 If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap()*
 48127 using the returned file descriptor shall result in allocation and mapping of typed memory from
 48128 the specified typed memory pool. The allocated memory may be a contiguous previously
 48129 unallocated area of the typed memory pool or several non-contiguous previously unallocated
 48130 areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag
 48131 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 48132 returned file descriptor shall result in allocation and mapping of a single contiguous previously
 48133 unallocated area of the typed memory pool (also mapped to a contiguous portion of the process
 48134 address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or
 48135 POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap()* using the
 48136 returned file descriptor shall map an application-chosen area from the specified typed memory
 48137 pool such that this mapped area becomes unavailable for allocation until unmapped by all
 48138 processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any
 48139 subsequent call to *mmap()* using the returned file descriptor shall map an application-chosen

48140 area from the specified typed memory pool without an effect on the availability of that area for
 48141 allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it
 48142 was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. The
 48143 appropriate privilege to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag is
 48144 implementation-defined.

48145 If successful, *posix_typed_mem_open()* shall return a file descriptor for the typed memory object
 48146 that is the lowest numbered file descriptor not currently open for that process. The open file
 48147 description is new, and therefore the file descriptor shall not share it with any other processes. It
 48148 is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated
 48149 with the new file descriptor shall be cleared.

48150 The behavior of *msync()*, *ftruncate()*, and all file operations other than *mmap()*,
 48151 *posix_mem_offset()*, *posix_typed_mem_get_info()*, *fstat()*, *dup()*, *dup2()*, and *close()*, is unspecified
 48152 when passed a file descriptor connected to a typed memory object by this function.

48153 The file status flags of the open file description shall be set according to the value of *oflag*.
 48154 Applications shall specify exactly one of the three access mode values described below and
 48155 defined in the `<fcntl.h>` header, as the value of *oflag*.

48156 O_RDONLY Open for read access only.
 48157 O_WRONLY Open for write access only.
 48158 O_RDWR Open for read or write access.

48159 RETURN VALUE

48160 Upon successful completion, the *posix_typed_mem_open()* function shall return a non-negative
 48161 integer representing the lowest numbered unused file descriptor. Otherwise, it shall return `-1`
 48162 and set *errno* to indicate the error.

48163 ERRORS

48164 The *posix_typed_mem_open()* function shall fail if:

48165 [EACCES] The typed memory object exists and the permissions specified by *oflag* are
 48166 denied.

48167 [EINTR] The *posix_typed_mem_open()* operation was interrupted by a signal.

48168 [EINVAL] The flags specified in *tflag* are invalid (more than one of
 48169 POSIX_TYPED_MEM_ALLOCATE,
 48170 POSIX_TYPED_MEM_ALLOCATE_CONTIG, or
 48171 POSIX_TYPED_MEM_MAP_ALLOCATABLE is specified).

48172 [EMFILE] All file descriptors available to the process are currently open.

48173 [ENAMETOOLONG] The length of the *name* argument exceeds `{PATH_MAX}` or a pathname
 48174 component is longer than `{NAME_MAX}`.
 48175

48176 [ENFILE] Too many file descriptors are currently open in the system.

48177 [ENOENT] The named typed memory object does not exist.

48178 [EPERM] The caller lacks the appropriate privilege to specify the flag
 48179 POSIX_TYPED_MEM_MAP_ALLOCATABLE in argument *tflag*.

posix_typed_mem_open()*System Interfaces*48180
48181
48182
48183
48184
48185
48186
48187
48188
48189
48190
48191
48192
48193**EXAMPLES**

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*close(), dup(), exec, fcntl(), fstat(), ftruncate(), mmap(), msync(), posix_mem_offset(), posix_typed_mem_get_info(), umask*XBD [<fcntl.h>](#), [<sys/mman.h>](#)**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

DRAFT

48194 **NAME**

48195 pow, powf, powl — power function

48196 **SYNOPSIS**

48197 #include <math.h>

48198 double pow(double x, double y);

48199 float powf(float x, float y);

48200 long double powl(long double x, long double y);

48201 **DESCRIPTION**

48202 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 48203 conflict between the requirements described here and the ISO C standard is unintentional. This
 48204 volume of POSIX.1-200x defers to the ISO C standard.

48205 These functions shall compute the value of x raised to the power y , x^y . If x is negative, the
 48206 application shall ensure that y is an integer value.

48207 An application wishing to check for error situations should set *errno* to zero and call
 48208 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 48209 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 48210 zero, an error has occurred.

48211 **RETURN VALUE**48212 Upon successful completion, these functions shall return the value of x raised to the power y .

48213 MX For finite values of $x < 0$, and finite non-integer values of y , a domain error shall occur and
 48214 either a NaN (if representable), or an implementation-defined value shall be returned.

48215 If the correct value would cause overflow, a range error shall occur and *pow()*, *powf()*, and
 48216 *powl()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively, with the
 48217 same sign as the correct value of the function.

48218 If the correct value would cause underflow, and is not representable, a range error may occur,
 48219 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

48220 CX For $y < 0$, if x is zero, a pole error may occur and *pow()*, *powf()*, and *powl()* shall return
 48221 MX \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively. On systems that support the
 48222 IEC 60559 Floating-Point option, a pole error shall occur and *pow()*, *powf()*, and *powl()* shall
 48223 return \pm HUGE_VAL, \pm HUGE_VALF, and \pm HUGE_VALL, respectively if y is an odd integer, or
 48224 HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively if y is not an odd integer.

48225 MX If x or y is a NaN, a NaN shall be returned (unless specified elsewhere in this description).48226 For any value of y (including NaN), if x is +1, 1.0 shall be returned.48227 For any value of x (including NaN), if y is ± 0 , 1.0 shall be returned.48228 For any odd integer value of $y > 0$, if x is ± 0 , ± 0 shall be returned.48229 For $y > 0$ and not an odd integer, if x is ± 0 , +0 shall be returned.48230 If x is -1 , and y is \pm Inf, 1.0 shall be returned.48231 For $|x| < 1$, if y is $-\text{Inf}$, $+\text{Inf}$ shall be returned.48232 For $|x| > 1$, if y is $-\text{Inf}$, +0 shall be returned.48233 For $|x| < 1$, if y is $+\text{Inf}$, +0 shall be returned.48234 For $|x| > 1$, if y is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

pow()

48235 For y an odd integer < 0 , if x is $-\text{Inf}$, -0 shall be returned.

48236 For $y < 0$ and not an odd integer, if x is $-\text{Inf}$, $+0$ shall be returned.

48237 For y an odd integer > 0 , if x is $-\text{Inf}$, $-\text{Inf}$ shall be returned.

48238 For $y > 0$ and not an odd integer, if x is $-\text{Inf}$, $+\text{Inf}$ shall be returned.

48239 For $y < 0$, if x is $+\text{Inf}$, $+0$ shall be returned.

48240 For $y > 0$, if x is $+\text{Inf}$, $+\text{Inf}$ shall be returned.

48241 If the correct value would cause underflow, and is representable, a range error may occur and

48242 the correct value shall be returned.

ERRORS

48243 These functions shall fail if:

48245 Domain Error The value of x is negative and y is a finite non-integer.

48246 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

48247 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*

48248 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception

48249 shall be raised.

48250 MX Pole Error The value of x is zero and y is negative.

48251 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

48252 then *errno* shall be set to [ERANGE]. If the integer expression

48253 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero

48254 floating-point exception shall be raised.

48255 Range Error The result overflows.

48256 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

48257 then *errno* shall be set to [ERANGE]. If the integer expression

48258 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow

48259 floating-point exception shall be raised.

48260 These functions may fail if:

48261 Pole Error The value of x is zero and y is negative.

48262 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

48263 then *errno* shall be set to [ERANGE]. If the integer expression

48264 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero

48265 floating-point exception shall be raised.

48266 Range Error The result underflows.

48267 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,

48268 then *errno* shall be set to [ERANGE]. If the integer expression

48269 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow

48270 floating-point exception shall be raised.

48271
48272
48273
48274
48275
48276
48277
48278
48279
48280
48281
48282
48283
48284
48285
48286
48287
48288
48289
48290
48291
48292
48293
48294
48295
48296
48297
48298

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp(), *feclearexcept()*, *fetestexcept()*, *isnan()*

XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The normative text is updated to avoid use of the term “must” for application requirements.

The *powf()* and *powl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/42 is applied, correcting the third paragraph in the RETURN VALUE section.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #51 (SD5-XSH-ERN-81) is applied.

pread()48299 **NAME**48300

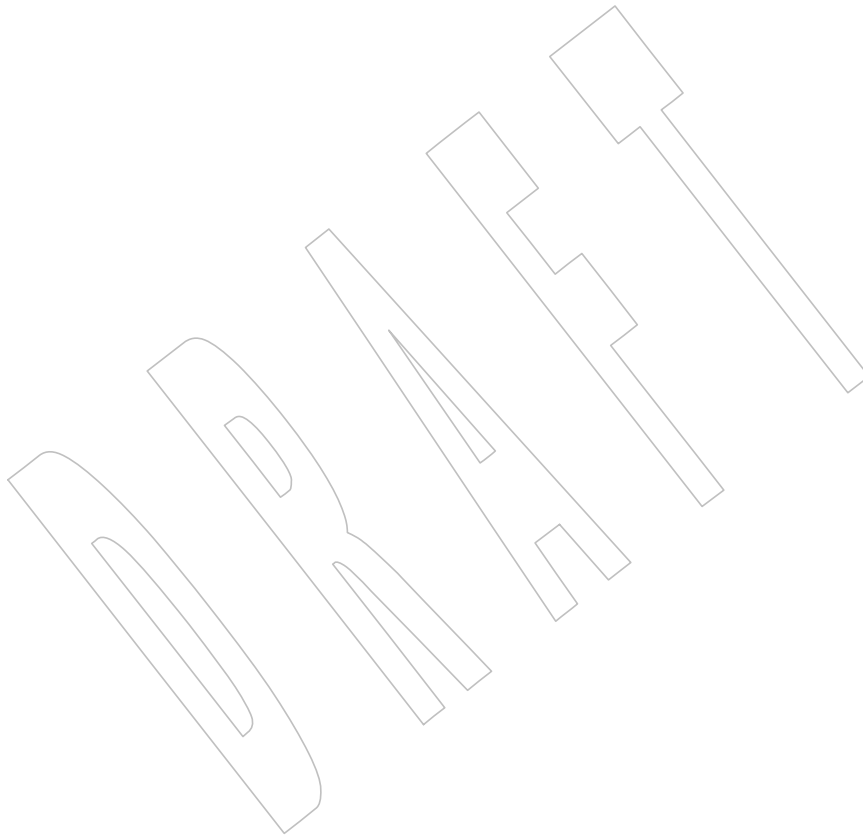
```
pread — read from a file
```

48301 **SYNOPSIS**48302

```
#include <unistd.h>
```

48303

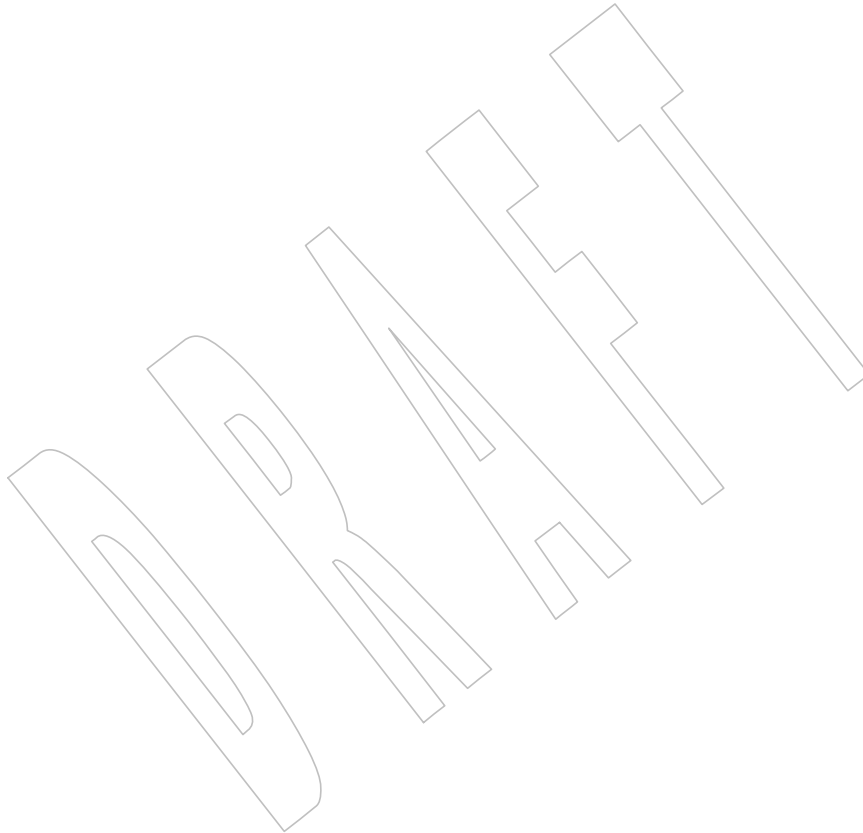
```
ssize_t pread(int fd, void *buf, size_t nbyte, off_t offset);
```

 |48304 **DESCRIPTION**48305 Refer to *read*.

48306 **NAME**
48307 printf — print formatted output

48308 **SYNOPSIS**
48309 #include <stdio.h>
48310 int printf(const char *restrict *format*, ...);

48311 **DESCRIPTION**
48312 Refer to *fprintf()*.



48313 NAME

48314 pselect, select — synchronous I/O multiplexing

48315 SYNOPSIS

```

48316 #include <sys/select.h>
48317
48317 int pselect(int nfds, fd_set *restrict readfds,
48318            fd_set *restrict writefds, fd_set *restrict errorfds,
48319            const struct timespec *restrict timeout,
48320            const sigset_t *restrict sigmask);
48321 int select(int nfds, fd_set *restrict readfds,
48322           fd_set *restrict writefds, fd_set *restrict errorfds,
48323           struct timeval *restrict timeout);
48324 void FD_CLR(int fd, fd_set *fdset);
48325 int FD_ISSET(int fd, fd_set *fdset);
48326 void FD_SET(int fd, fd_set *fdset);
48327 void FD_ZERO(fd_set *fdset);

```

48328 DESCRIPTION

48329 The *pselect()* function shall examine the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

48332 The *select()* function shall be equivalent to the *pselect()* function, except as follows:

- 48333 • For the *select()* function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect()* function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
- 48334 • The *select()* function has no *sigmask* argument; it shall behave as *pselect()* does when *sigmask* is a null pointer.
- 48336 • Upon successful completion, the *select()* function may modify the object pointed to by the *timeout* argument.

48340 The *pselect()* and *select()* functions shall support regular files, terminal and pseudo-terminal devices, **STREAMS-based files**, FIFOs, pipes, and sockets. The behavior of *pselect()* and *select()* on file descriptors that refer to other types of file is unspecified.

48343 The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall be checked in each set; that is, the descriptors from zero through *nfds*-1 in the descriptor sets shall be examined.

48346 If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

48349 If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

48352 If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

48355 Upon successful completion, the *pselect()* or *select()* function shall modify the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the

48358 total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the
 48359 corresponding bit shall be set on successful completion if it was set on input and the associated
 48360 condition is true for that file descriptor.

48361 If none of the selected descriptors are ready for the requested operation, the *pselect()* or *select()*
 48362 function shall block until at least one of the requested operations becomes ready, until the
 48363 *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the
 48364 *pselect()* or *select()* function shall take before timing out. If the *timeout* parameter is not a null
 48365 pointer, it specifies a maximum interval to wait for the selection to complete. If the specified
 48366 time interval expires without any requested operation becoming ready, the function shall return.
 48367 If the *timeout* parameter is a null pointer, then the call to *pselect()* or *select()* shall block
 48368 indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout*
 48369 parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

48370 The use of a timeout does not affect any pending timers set up by *alarm()* or *setitimer()*.

48371 Implementations may place limitations on the maximum timeout interval supported. All
 48372 implementations shall support a maximum timeout interval of at least 31 days. If the *timeout*
 48373 argument specifies a timeout interval greater than the implementation-defined maximum value,
 48374 the maximum value shall be used as the actual timeout value. Implementations may also place
 48375 limitations on the granularity of timeout intervals. If the requested timeout interval requires a
 48376 finer granularity than the implementation supports, the actual timeout interval shall be rounded
 48377 up to the next supported value.

48378 If *sigmask* is not a null pointer, then the *pselect()* function shall replace the signal mask of the
 48379 caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall
 48380 restore the signal mask of the calling thread before returning.

48381 A descriptor shall be considered ready for reading when a call to an input function with
 48382 O_NONBLOCK clear would not block, whether or not the function would transfer data
 48383 successfully. (The function might return data, an end-of-file indication, or an error other than
 48384 one indicating that it is blocked, and in each of these cases the descriptor shall be considered
 48385 ready for reading.)

48386 A descriptor shall be considered ready for writing when a call to an output function with
 48387 O_NONBLOCK clear would not block, whether or not the function would transfer data
 48388 successfully.

48389 If a socket has a pending error, it shall be considered to have an exceptional condition pending.
 48390 Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for
 48391 use with a socket, it is protocol-specific except as noted below. For other file types it is
 48392 implementation-defined. If the operation is meaningless for a particular file type, *pselect()* or
 48393 *select()* shall indicate that the descriptor is ready for read or write operations, and shall indicate
 48394 that the descriptor has no exceptional condition pending.

48395 If a descriptor refers to a socket, the implied input function is the *recvmsg()* function with
 48396 parameters requesting normal and ancillary data, such that the presence of either type shall
 48397 cause the socket to be marked as readable. The presence of out-of-band data shall be checked if
 48398 the socket option SO_OOINLINE has been enabled, as out-of-band data is enqueued with
 48399 normal data. If the socket is currently listening, then it shall be marked as readable if an
 48400 incoming connection request has been received, and a call to the *accept()* function shall complete
 48401 without blocking.

48402 If a descriptor refers to a socket, the implied output function is the *sendmsg()* function supplying
 48403 an amount of normal data equal to the current value of the SO_SNDLOWAT option for the
 48404 socket. If a non-blocking call to the *connect()* function has been made for a socket, and the
 48405 connection attempt has either succeeded or failed leaving a pending error, the socket shall be
 48406 marked as writable.

48407 A socket shall be considered to have an exceptional condition pending if a receive operation
 48408 with `O_NONBLOCK` clear for the open file description and with the `MSG_OOB` flag set would
 48409 return out-of-band data without blocking. (It is protocol-specific whether the `MSG_OOB` flag
 48410 would be used to read out-of-band data.) A socket shall also be considered to have an
 48411 exceptional condition pending if an out-of-band data mark is present in the receive queue. Other
 48412 circumstances under which a socket may be considered to have an exceptional condition
 48413 pending are protocol-specific and implementation-defined.

48414 If the `readfds`, `writfds`, and `errorfds` arguments are all null pointers and the `timeout` argument is
 48415 not a null pointer, the `pselect()` or `select()` function shall block for the time specified, or until
 48416 interrupted by a signal. If the `readfds`, `writfds`, and `errorfds` arguments are all null pointers and
 48417 the `timeout` argument is a null pointer, the `pselect()` or `select()` function shall block until
 48418 interrupted by a signal.

48419 File descriptors associated with regular files shall always select true for ready to read, ready to
 48420 write, and error conditions.

48421 On failure, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments shall not be
 48422 modified. If the timeout interval expires without the specified condition being true for any of the
 48423 specified file descriptors, the objects pointed to by the `readfds`, `writfds`, and `errorfds` arguments
 48424 shall have all bits set to 0.

48425 File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`,
 48426 `FD_SET()`, and `FD_ZERO()`. It is unspecified whether each of these is a macro or a function. If a
 48427 macro definition is suppressed in order to access an actual function, or a program defines an
 48428 external identifier with any of these names, the behavior is undefined.

48429 `FD_CLR(fd, fdsetp)` shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not
 48430 a member of this set, there shall be no effect on the set, nor will an error be returned.

48431 `FD_ISSET(fd, fdsetp)` shall evaluate to non-zero if the file descriptor *fd* is a member of the set
 48432 pointed to by *fdsetp*, and shall evaluate to zero otherwise.

48433 `FD_SET(fd, fdsetp)` shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file
 48434 descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be
 48435 returned.

48436 `FD_ZERO(fdsetp)` shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is
 48437 returned if the set is not empty at the time `FD_ZERO()` is invoked.

48438 The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or
 48439 equal to `FD_SETSIZE`, or if *fd* is not a valid file descriptor, or if any of the arguments are
 48440 expressions with side effects.

48441 If a thread gets canceled during a `pselect()` call, the signal mask in effect when executing the
 48442 registered cleanup functions is either the original signal mask or the signal mask installed as part
 48443 of the `pselect()` call.

48444 RETURN VALUE

48445 Upon successful completion, the `pselect()` and `select()` functions shall return the total number of
 48446 bits set in the bit masks. Otherwise, `-1` shall be returned, and `errno` shall be set to indicate the
 48447 error.

48448 `FD_CLR()`, `FD_SET()`, and `FD_ZERO()` do not return a value. `FD_ISSET()` shall return a non-
 48449 zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and
 48450 0 otherwise.

48451 **ERRORS**48452 Under the following conditions, *pselect()* and *select()* shall fail and set *errno* to:48453 [EBADF] One or more of the file descriptor sets specified a file descriptor that is not a
48454 valid open file descriptor.48455 [EINTR] The function was interrupted before any of the selected events occurred and
48456 before the timeout interval expired.48457 XSI If SA_RESTART has been set for the interrupting signal, it is implementation-
48458 defined whether the function restarts or returns with [EINTR].

48459 [EINVAL] An invalid timeout interval was specified.

48460 [EINVAL] The *nfds* argument is less than 0 or greater than FD_SETSIZE.48461 OB XSR [EINVAL] One of the specified file descriptors refers to a STREAM or multiplexer that is
48462 linked (directly or indirectly) downstream from a multiplexer.48463 **EXAMPLES**

48464 None.

48465 **APPLICATION USAGE**

48466 None.

48467 **RATIONALE**48468 In earlier versions of the Single UNIX Specification, the *select()* function was defined in the
48469 `<sys/time.h>` header. This is now changed to `<sys/select.h>`. The rationale for this change was
48470 as follows: the introduction of the *pselect()* function included the `<sys/select.h>` header and the
48471 `<sys/select.h>` header defines all the related definitions for the *pselect()* and *select()* functions.
48472 Backwards-compatibility to existing XSI implementations is handled by allowing `<sys/time.h>`
48473 to include `<sys/select.h>`.48474 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers
48475 can install an additional cancellation handler which resets the signal mask to the expected value.48476

```
void cleanup(void *arg)
48477 {
48478     sigset_t *ss = (sigset_t *) arg;
48479     pthread_sigmask(SIG_SETMASK, ss, NULL);
48480 }
48481 int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
48482 fd_set errorfds, const struct timespec *timeout,
48483 const sigset_t *sigmask)
48484 {
48485     sigset_t oldmask;
48486     int result;
48487     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
48488     pthread_cleanup_push(cleanup, &oldmask);
48489     result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
48490     pthread_cleanup_pop(0);
48491     return result;
48492 }
```

48493 **FUTURE DIRECTIONS**

48494 None.

48495 **SEE ALSO**48496 *accept(), alarm(), connect(), fcntl(), getitimer(), poll(), read, recvmsg(), sendmsg(), write*48497 XBD **<sys/select.h>**, **<sys/time.h>**48498 **CHANGE HISTORY**

48499 First released in Issue 4, Version 2.

48500 **Issue 5**

48501 Moved from X/OPEN UNIX extension to BASE.

48502 In the ERRORS section, the text has been changed to indicate that [EINVAL] is returned when
48503 *nfds* is less than 0 or greater than FD_SETSIZE. It previously stated less than 0, or greater than or
48504 equal to FD_SETSIZE.48505 Text about *timeout* is moved from the APPLICATION USAGE section to the DESCRIPTION.48506 **Issue 6**48507 The Open Group Corrigendum U026/6 is applied, changing the occurrences of *readfs* and *writefs*
48508 in the *select()* DESCRIPTION to be *readfds* and *writefds*.

48509 Text referring to sockets is added to the DESCRIPTION.

48510 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
48511 marked as part of the XSI STREAMS Option Group.48512 The following new requirements on POSIX implementations derive from alignment with the
48513 Single UNIX Specification:

- 48514
- These functions are now mandatory.

48515 The *pselect()* function is added for alignment with IEEE Std 1003.1g-2000 and additional detail
48516 related to sockets semantics is added to the DESCRIPTION.48517 The *select()* function now requires inclusion of **<sys/select.h>**.48518 The **restrict** keyword is added to the *select()* prototype for alignment with the
48519 ISO/IEC 9899:1999 standard.48520 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/70 is applied, updating the
48521 DESCRIPTION to reference the signal mask in terms of the calling thread rather than the
48522 process.48523 **Issue 7**48524 SD5-XSH-ERN-122 is applied, adding text to the DESCRIPTION for when a thread is canceled
48525 during a call to *pselect()*, and adding example code to the RATIONALE.

48526 Functionality relating to the XSI STREAMS option is marked obsolescent.

48527 Functionality relating to the Threads option is moved to the Base.

48528 **NAME**
 48529 `psiginfo, psignal` — print signal information to standard error

48530 **SYNOPSIS**

```
48531 CX #include <signal.h>
48532 void psiginfo(siginfo_t *pinfo, const char *message);
48533 void psignal(int signum, const char *message);
```

48534 **DESCRIPTION**

48535 The `psiginfo()` and `psignal()` functions shall print a message out on `stderr` associated with a signal
 48536 number. If `message` is not null and is not the empty string, then the string pointed to by the
 48537 `message` argument shall be printed first, followed by a colon, a space, and the signal description
 48538 string indicated by `signum`, or by the signal associated with `pinfo`. If the `message` argument is null
 48539 or points to an empty string, then only the signal description shall be printed. For `psiginfo()`, the
 48540 argument `pinfo` references a valid **`siginfo_t`** structure. For `psignal()`, if `signum` is not a valid signal
 48541 number, the behavior is implementation-defined.

48542 **RETURN VALUE**

48543 These functions shall not return a value.

48544 **ERRORS**

48545 No errors are defined.

48546 **EXAMPLES**

48547 None.

48548 **APPLICATION USAGE**

48549 None.

48550 **RATIONALE**

48551 System V historically has `psignal()` and `psiginfo()` in `<siginfo.h>`. However, the `<siginfo.h>`
 48552 header is not specified in the Base Definitions volume of POSIX.1-200x, and the type **`siginfo_t`** is
 48553 defined in `<signal.h>`.

48554 **FUTURE DIRECTIONS**

48555 None.

48556 **SEE ALSO**

48557 [*`perror\(\)`*](#), [*`strsignal\(\)`*](#)

48558 XBD `<signal.h>`

48559 **CHANGE HISTORY**

48560 First released in Issue 7.

psignal()

48561 **NAME**
48562 psignal — print signal information to standard error

48563 **SYNOPSIS**

```
48564 CX       #include <signal.h>  
48565       void psignal(int signum, const char *message);
```

48566 **DESCRIPTION**

48567 Refer to *psiginfo()*.

48568 **NAME**

48569 pthread_atfork — register fork handlers

48570 **SYNOPSIS**

48571 #include <pthread.h>

48572 int pthread_atfork(void (*prepare)(void), void (*parent)(void),
48573 void (*child)(void));48574 **DESCRIPTION**

48575 The *pthread_atfork()* function shall declare fork handlers to be called before and after *fork()*, in
48576 the context of the thread that called *fork()*. The *prepare* fork handler shall be called before *fork()*
48577 processing commences. The *parent* fork handle shall be called after *fork()* processing completes
48578 in the parent process. The *child* fork handler shall be called after *fork()* processing completes
48579 in the child process. If no handling is desired at one or more of these three points, the
48580 corresponding fork handler address(es) may be set to NULL.

48581 The order of calls to *pthread_atfork()* is significant. The *parent* and *child* fork handlers shall be
48582 called in the order in which they were established by calls to *pthread_atfork()*. The *prepare* fork
48583 handlers shall be called in the opposite order.

48584 **RETURN VALUE**

48585 Upon successful completion, *pthread_atfork()* shall return a value of zero; otherwise, an error
48586 number shall be returned to indicate the error.

48587 **ERRORS**48588 The *pthread_atfork()* function shall fail if:

48589 [ENOMEM] Insufficient table space exists to record the fork handler addresses.

48590 The *pthread_atfork()* function shall not return an error code of [EINTR].48591 **EXAMPLES**

48592 None.

48593 **APPLICATION USAGE**

48594 None.

48595 **RATIONALE**

48596 There are at least two serious problems with the semantics of *fork()* in a multi-threaded
48597 program. One problem has to do with state (for example, memory) covered by mutexes.
48598 Consider the case where one thread has a mutex locked and the state covered by that mutex is
48599 inconsistent while another thread calls *fork()*. In the child, the mutex is in the locked state
48600 (locked by a nonexistent thread and thus can never be unlocked). Having the child simply
48601 reinitialize the mutex is unsatisfactory since this approach does not resolve the question about
48602 how to correct or otherwise deal with the inconsistent state in the child.

48603 It is suggested that programs that use *fork()* call an *exec* function very soon afterwards in the
48604 child process, thus resetting all states. In the meantime, only a short list of async-signal-safe
48605 library routines are promised to be available.

48606 Unfortunately, this solution does not address the needs of multi-threaded libraries. Application
48607 programs may not be aware that a multi-threaded library is in use, and they feel free to call any
48608 number of library routines between the *fork()* and *exec* calls, just as they always have. Indeed,
48609 they may be extant single-threaded programs and cannot, therefore, be expected to obey new
48610 restrictions imposed by the threads library.

48611 On the other hand, the multi-threaded library needs a way to protect its internal state during
48612 *fork()* in case it is re-entered later in the child process. The problem arises especially in multi-

48613 threaded I/O libraries, which are almost sure to be invoked between the *fork()* and *exec* calls to
 48614 effect I/O redirection. The solution may require locking mutex variables during *fork()*, or it may
 48615 entail simply resetting the state in the child after the *fork()* processing completes.

48616 The *pthread_atfork()* function provides multi-threaded libraries with a means to protect
 48617 themselves from innocent application programs that call *fork()*, and it provides multi-threaded
 48618 application programs with a standard mechanism for protecting themselves from *fork()* calls in a
 48619 library routine or the application itself.

48620 The expected usage is that the *prepare* handler acquires all mutex locks and the other two fork
 48621 handlers release them.

48622 For example, an application can supply a *prepare* routine that acquires the necessary mutexes the
 48623 library maintains and supply *child* and *parent* routines that release those mutexes, thus ensuring
 48624 that the child gets a consistent snapshot of the state of the library (and that no mutexes are left
 48625 stranded). Alternatively, some libraries might be able to supply just a *child* routine that
 48626 reinitializes the mutexes in the library and all associated states to some known value (for
 48627 example, what it was when the image was originally executed).

48628 When *fork()* is called, only the calling thread is duplicated in the child process. Synchronization
 48629 variables remain in the same state in the child as they were in the parent at the time *fork()* was
 48630 called. Thus, for example, mutex locks may be held by threads that no longer exist in the child
 48631 process, and any associated states may be inconsistent. The parent process may avoid this by
 48632 explicit code that acquires and releases locks critical to the child via *pthread_atfork()*. In addition,
 48633 any critical threads need to be recreated and reinitialized to the proper state in the child (also via
 48634 *pthread_atfork()*).

48635 A higher-level package may acquire locks on its own data structures before invoking lower-level
 48636 packages. Under this scenario, the order specified for fork handler calls allows a simple rule of
 48637 initialization for avoiding package deadlock: a package initializes all packages on which it
 48638 depends before it calls the *pthread_atfork()* function for itself.

48639 FUTURE DIRECTIONS

48640 None.

48641 SEE ALSO

48642 [atexit\(\)](#), [exec](#), [fork\(\)](#)

48643 XBD [<sys/types.h>](#)

48644 CHANGE HISTORY

48645 First released in Issue 5. Derived from the POSIX Threads Extension.

48646 IEEE PASC Interpretation 1003.1c #4 is applied.

48647 Issue 6

48648 The *pthread_atfork()* function is marked as part of the Threads option.

48649 The [<pthread.h>](#) header is added to the SYNOPSIS.

48650 Issue 7

48651 The *pthread_atfork()* function is moved from the Threads option to the Base.

48652 **NAME**

48653 pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

48654 **SYNOPSIS**

48655 #include <pthread.h>

48656 int pthread_attr_destroy(pthread_attr_t *attr);

48657 int pthread_attr_init(pthread_attr_t *attr);

48658 **DESCRIPTION**

48659 The *pthread_attr_destroy()* function shall destroy a thread attributes object. An implementation
 48660 may cause *pthread_attr_destroy()* to set *attr* to an implementation-defined invalid value. A
 48661 destroyed *attr* attributes object can be reinitialized using *pthread_attr_init()*; the results of
 48662 otherwise referencing the object after it has been destroyed are undefined.

48663 The *pthread_attr_init()* function shall initialize a thread attributes object *attr* with the default
 48664 value for all of the individual attributes used by a given implementation.

48665 The resulting attributes object (possibly modified by setting individual attribute values) when
 48666 used by *pthread_create()* defines the attributes of the thread created. A single attributes object can
 48667 be used in multiple simultaneous calls to *pthread_create()*. Results are undefined if
 48668 *pthread_attr_init()* is called specifying an already initialized *attr* attributes object.

48669 **RETURN VALUE**

48670 Upon successful completion, *pthread_attr_destroy()* and *pthread_attr_init()* shall return a value of
 48671 0; otherwise, an error number shall be returned to indicate the error.

48672 **ERRORS**48673 The *pthread_attr_init()* function shall fail if:

48674 [ENOMEM] Insufficient memory exists to initialize the thread attributes object.

48675 The *pthread_attr_destroy()* function may fail if:48676 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
48677 object.48678 The *pthread_attr_init()* function may fail if:48679 [EBUSY] The implementation has detected an attempt to reinitialize the thread attribute
48680 referenced by *attr*, a previously initialized, but not yet destroyed, thread
48681 attribute.

48682 These functions shall not return an error code of [EINTR].

48683 **EXAMPLES**

48684 None.

48685 **APPLICATION USAGE**

48686 None.

48687 **RATIONALE**

48688 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to
 48689 support probable future standardization in these areas without requiring that the function itself
 48690 be changed.

48691 Attributes objects provide clean isolation of the configurable aspects of threads. For example,
 48692 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When
 48693 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects
 48694 can help by allowing the changes to be isolated in a single place, rather than being spread across

48695 every instance of thread creation.

48696 Attributes objects can be used to set up “classes’ of threads with similar attributes; for example,
48697 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes
48698 can be defined in a single place and then referenced wherever threads need to be created.
48699 Changes to “class” decisions become straightforward, and detailed analysis of each
48700 *pthread_create()* call is not required.

48701 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had
48702 been specified as structures, adding new attributes would force recompilation of all multi-
48703 threaded programs when the attributes objects are extended; this might not be possible if
48704 different program components were supplied by different vendors.

48705 Additionally, opaque attributes objects present opportunities for improving performance.
48706 Argument validity can be checked once when attributes are set, rather than each time a thread is
48707 created. Implementations often need to cache kernel objects that are expensive to create.
48708 Opaque attributes objects provide an efficient mechanism to detect when cached objects become
48709 invalid due to attribute changes.

48710 Since assignment is not necessarily defined on a given opaque type, implementation-defined
48711 default values cannot be defined in a portable way. The solution to this problem is to allow
48712 attributes objects to be initialized dynamically by attributes object initialization functions, so that
48713 default values can be supplied automatically by the implementation.

48714 The following proposal was provided as a suggested alternative to the supplied attributes:

- 48715 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to
48716 the initialization routines (*pthread_create()*, *pthread_mutex_init()*, *pthread_cond_init()*). The
48717 parameter containing the flags should be an opaque type for extensibility. If no flags are
48718 set in the parameter, then the objects are created with default characteristics. An
48719 implementation may specify implementation-defined flag values and associated
48720 behavior.
- 48721 2. If further specialization of mutexes and condition variables is necessary, implementations
48722 may specify additional procedures that operate on the **pthread_mutex_t** and
48723 **pthread_cond_t** objects (instead of on attributes objects).

48724 The difficulties with this solution are:

- 48725 1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using
48726 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,
48727 application programmers need to know the location of each bit. If bits are set or read by
48728 encapsulation (that is, get and set functions), then the bitmask is merely an
48729 implementation of attributes objects as currently defined and should not be exposed to
48730 the programmer.
- 48731 2. Many attributes are not Boolean or very small integral values. For example, scheduling
48732 policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking
48733 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,
48734 the bitmask can only reasonably control whether particular attributes are set or not, and it
48735 cannot serve as the repository of the value itself. The value needs to be specified as a
48736 function parameter (which is non-extensible), or by setting a structure field (which is non-
48737 opaque), or by get and set functions (making the bitmask a redundant addition to the
48738 attributes objects).

48739 Stack size is defined as an optional attribute because the very notion of a stack is inherently
48740 machine-dependent. Some implementations may not be able to change the size of the stack, for
48741 example, and others may not need to because stack pages may be discontinuous and can be
48742 allocated and released on demand.

48743 The attribute mechanism has been designed in large measure for extensibility. Future extensions
 48744 to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-200x
 48745 has to be done with care so as not to affect binary-compatibility.

48746 Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc()*,
 48747 may have their size fixed at compile time. This means, for example, a *pthread_create()* in an
 48748 implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary
 48749 application assumes is valid. This suggests that implementations should maintain a size field in
 48750 the attributes object, as well as possibly version information, if extensions in different directions
 48751 (possibly by different vendors) are to be accommodated.

FUTURE DIRECTIONS

48752 None.
 48753

SEE ALSO

48754 *pthread_attr_getstacksize()*, *pthread_attr_getdetachstate()*, *pthread_create()*
 48755

48756 XBD <pthread.h> +

CHANGE HISTORY

48757 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
 48758

Issue 6

48759 The *pthread_attr_destroy()* and *pthread_attr_init()* functions are marked as part of the Threads
 48760 option.
 48761

48762 IEEE PASC Interpretation 1003.1 #107 is applied, noting that the effect of initializing an already
 48763 initialized thread attributes object is undefined.

48764 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/71 is applied, updating the ERRORS
 48765 section to add the optional [EINVAL] error for the *pthread_attr_destroy()* function, and the
 48766 optional [EBUSY] error for the *pthread_attr_init()* function.

Issue 7

48767 The *pthread_attr_destroy()* and *pthread_attr_init()* functions are moved from the Threads option
 48768 to the Base.
 48769

48770 **NAME**

48771 pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

48772 **SYNOPSIS**

```
48773 #include <pthread.h>
48774
48775 int pthread_attr_getdetachstate(const pthread_attr_t *attr,
48776                               int *detachstate);
48777
48778 int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

48777 **DESCRIPTION**

48778 The *detachstate* attribute controls whether the thread is created in a detached state. If the thread
 48779 is created detached, then use of the ID of the newly created thread by the *pthread_detach()* or
 48780 *pthread_join()* function is an error.

48781 The *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* functions, respectively, shall get
 48782 and set the *detachstate* attribute in the *attr* object.

48783 For *pthread_attr_getdetachstate()*, *detachstate* shall be set to either
 48784 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

48785 For *pthread_attr_setdetachstate()*, the application shall set *detachstate* to either
 48786 PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

48787 A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in
 48788 the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all
 48789 threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute
 48790 shall be PTHREAD_CREATE_JOINABLE.

48791 **RETURN VALUE**

48792 Upon successful completion, *pthread_attr_getdetachstate()* and *pthread_attr_setdetachstate()* shall
 48793 return a value of 0; otherwise, an error number shall be returned to indicate the error.

48794 The *pthread_attr_getdetachstate()* function stores the value of the *detachstate* attribute in *detachstate*
 48795 if successful.

48796 **ERRORS**

48797 The *pthread_attr_setdetachstate()* function shall fail if:

48798 [EINVAL] The value of *detachstate* was not valid

48799 These functions may fail if:

48800 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 48801 object.

48802 These functions shall not return an error code of [EINTR].

48803 **EXAMPLES**48804 **Retrieving the detachstate Attribute**

48805 This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
48806 #include <pthread.h>
48807
48808 pthread_attr_t thread_attr;
48809 int detachstate;
48810 int rc;
48811
48812 /* code initializing thread_attr */
```

```

48811     ...
48812     rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
48813     if (rc!=0) {
48814         /* handle error */
48815         ...
48816     }
48817     else {
48818         /* legal values for detachstate are:
48819          * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
48820          */
48821         ...
48822     }

```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[pthread_attr_destroy\(\)](#), [pthread_attr_getstacksize\(\)](#), [pthread_create\(\)](#)

XBD [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The [pthread_attr_setdetachstate\(\)](#) and [pthread_attr_getdetachstate\(\)](#) functions are marked as part of the Threads option.

The normative text is updated to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/72 is applied, adding the example to the EXAMPLES section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/73 is applied, updating the ERRORS section to include the optional [EINVAL] error.

Issue 7

The [pthread_attr_setdetachstate\(\)](#) and [pthread_attr_getdetachstate\(\)](#) functions are moved from the Threads option to the Base.

48845 **NAME**

48846 pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize
 48847 attribute

48848 **SYNOPSIS**

```
48849 #include <pthread.h>

48850 int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
48851 size_t *restrict guardsize);
48852 int pthread_attr_setguardsize(pthread_attr_t *attr,
48853 size_t guardsize);
```

48854 **DESCRIPTION**

48855 The *pthread_attr_getguardsize()* function shall get the *guardsize* attribute in the *attr* object. This
 48856 attribute shall be returned in the *guardsize* parameter.

48857 The *pthread_attr_setguardsize()* function shall set the *guardsize* attribute in the *attr* object. The new
 48858 value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard
 48859 area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard
 48860 area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

48861 The *guardsize* attribute controls the size of the guard area for the created thread's stack. The
 48862 *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is
 48863 created with guard protection, the implementation allocates extra memory at the overflow end
 48864 of the stack as a buffer against stack overflow of the stack pointer. If an application overflows
 48865 into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

48866 A conforming implementation may round up the value contained in *guardsize* to a multiple of
 48867 the configurable system variable {PAGESIZE} (see <sys/mman.h>). If an implementation
 48868 rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize()*
 48869 specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous
 48870 *pthread_attr_setguardsize()* function call.

48871 The default value of the *guardsize* attribute is implementation-defined.

48872 If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread
 48873 stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the
 48874 implementation. It is the responsibility of the application to manage stack overflow along with
 48875 stack allocation and management in this case.

48876 **RETURN VALUE**

48877 If successful, the *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions shall return
 48878 zero; otherwise, an error number shall be returned to indicate the error.

48879 **ERRORS**

48880 These functions shall fail if:

48881 [EINVAL] The parameter *guardsize* is invalid.

48882 These functions may fail if:

48883 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 48884 object.

48885 These functions shall not return an error code of [EINTR].

EXAMPLES**Retrieving the guardsize Attribute**

This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```

48886 #include <pthread.h>
48887
48888 pthread_attr_t thread_attr;
48889 size_t guardsize;
48890 int rc;
48891
48892 /* code initializing thread_attr */
48893 ...
48894
48895 rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
48896 if (rc != 0) {
48897     /* handle error */
48898     ...
48899 }
48900 else {
48901     if (guardsize > 0) {
48902         /* a guard area of at least guardsize bytes is provided */
48903         ...
48904     }
48905     else {
48906         /* no guard area provided */
48907         ...
48908     }
48909 }

```

APPLICATION USAGE

None.

RATIONALE

The *guardsize* attribute is provided to the application for two reasons:

1. Overflow protection can potentially result in wasted system resources. An application that creates a large number of threads, and which knows its threads never overflow their stack, can save system resources by turning off guard areas.
2. When threads allocate large data structures on the stack, large guard areas may be needed to detect stack overflow.

The default size of the guard area is left implementation-defined since on systems supporting very large page sizes, the overhead might be substantial if at least one guard page is required by default.

FUTURE DIRECTIONS

None.

SEE ALSO

XBD [<pthread.h>](#), [<sys/mman.h>](#)

CHANGE HISTORY

First released in Issue 5.

48928

Issue 6

48929

In the ERRORS section, a third [EINVAL] error condition is removed as it is covered by the second error condition.

48930

48931

The **restrict** keyword is added to the *pthread_attr_getguardsize()* prototype for alignment with the ISO/IEC 9899:1999 standard.

48932

48933

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/74 is applied, updating the ERRORS section to remove the [EINVAL] error ("The attribute *attr* is invalid."), and replacing it with the optional [EINVAL] error.

48934

48935

48936

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/76 is applied, adding the example to the EXAMPLES section.

48937

48938

Issue 7

48939

SD5-XSH-ERN-111 is applied, removing the reference to the *stack* attribute in the DESCRIPTION.

48940

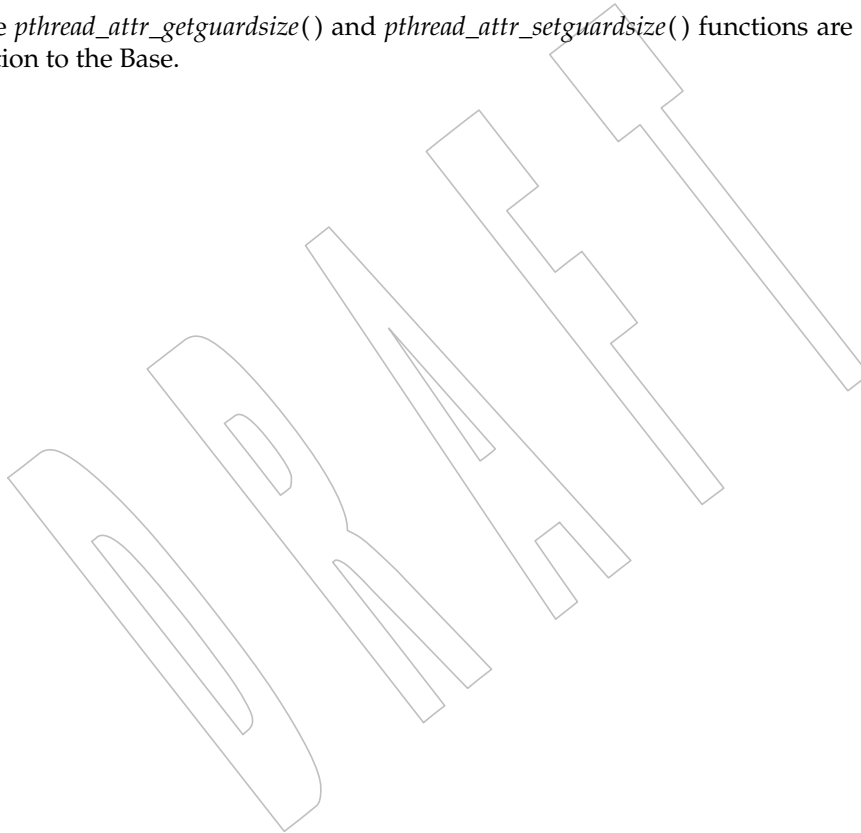
SD5-XSH-ERN-175 is applied, updating the DESCRIPTION to note that the default size of the guard area is implementation-defined. +

48941

48942

The *pthread_attr_getguardsize()* and *pthread_attr_setguardsize()* functions are moved from the XSI option to the Base. +

48943



48944 **NAME**

48945 pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched
 48946 attribute (**REALTIME THREADS**)

48947 **SYNOPSIS**

```
48948 TPS #include <pthread.h>
48949
48949 int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
48950 int *restrict inheritsched);
48951 int pthread_attr_setinheritsched(pthread_attr_t *attr,
48952 int inheritsched);
```

48953 **DESCRIPTION**

48954 The *pthread_attr_getinheritsched()*, and *pthread_attr_setinheritsched()* functions, respectively, shall
 48955 get and set the *inheritsched* attribute in the *attr* argument.

48956 When the attributes objects are used by *pthread_create()*, the *inheritsched* attribute determines
 48957 how the other scheduling attributes of the created thread shall be set.

48958 The supported values of *inheritsched* shall be:

48959 **PTHREAD_INHERIT_SCHED**

48960 Specifies that the thread scheduling attributes shall be inherited from the creating thread,
 48961 and the scheduling attributes in this *attr* argument shall be ignored.

48962 **PTHREAD_EXPLICIT_SCHED**

48963 Specifies that the thread scheduling attributes shall be set to the corresponding values from
 48964 this attributes object.

48965 The symbols **PTHREAD_INHERIT_SCHED** and **PTHREAD_EXPLICIT_SCHED** are defined in
 48966 the **<pthread.h>** header.

48967 The following thread scheduling attributes defined by POSIX.1-200x are affected by the
 48968 *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and
 48969 scheduling contention scope (*contentionscope*).

48970 **RETURN VALUE**

48971 If successful, the *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions shall
 48972 return zero; otherwise, an error number shall be returned to indicate the error.

48973 **ERRORS**

48974 The *pthread_attr_getinheritsched()* function may fail if:

48975 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 48976 object.

48977 The *pthread_attr_setinheritsched()* function may fail if:

48978 [EINVAL] The value of *inheritsched* is not valid.

48979 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 48980 object.

48981 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

48982 These functions shall not return an error code of [EINTR].

pthread_attr_getinheritsched()

System Interfaces

48983 **EXAMPLES**

48984 None.

48985 **APPLICATION USAGE**48986 After these attributes have been set, a thread can be created with the specified attributes using
48987 *pthread_create()*. Using these routines does not affect the current running thread.48988 See [Section 2.9.4](#) (on page 488) for further details on thread scheduling attributes and their
48989 default settings.48990 **RATIONALE**

48991 None.

48992 **FUTURE DIRECTIONS**

48993 None.

48994 **SEE ALSO**48995 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getschedpolicy()*,
48996 *pthread_attr_getschedparam()*, *pthread_create()*48997 XBD [<pthread.h>](#), [<sched.h>](#) +48998 **CHANGE HISTORY**

48999 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49000 Marked as part of the Realtime Threads Feature Group.

49001 **Issue 6**49002 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are marked as part
49003 of the Threads and Thread Execution Scheduling options.49004 The [ENOSYS] error condition has been removed as stubs need not be provided if an
49005 implementation does not support the Thread Execution Scheduling option.49006 The **restrict** keyword is added to the *pthread_attr_getinheritsched()* prototype for alignment with
49007 the ISO/IEC 9899:1999 standard.49008 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/75 is applied, clarifying the values of
49009 *inheritsched* in the DESCRIPTION and adding two optional [EINVAL] errors to the ERRORS
49010 section for checking when *attr* refers to an uninitialized thread attribute object.49011 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/77 is applied, adding a reference to
49012 [Section 2.9.4](#) (on page 488) in the APPLICATION USAGE section.49013 **Issue 7**49014 The *pthread_attr_getinheritsched()* and *pthread_attr_setinheritsched()* functions are moved from the
49015 Threads option.

49016 **NAME**

49017 pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam
49018 attribute

49019 **SYNOPSIS**

```
49020 #include <pthread.h>

49021 int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
49022 struct sched_param *restrict param);
49023 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
49024 const struct sched_param *restrict param);
```

49025 **DESCRIPTION**

49026 The *pthread_attr_getschedparam()*, and *pthread_attr_setschedparam()* functions, respectively, shall
49027 get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param*
49028 structure are defined in the **<sched.h>** header. For the SCHED_FIFO and SCHED_RR policies,
49029 the only required member of *param* is *sched_priority*.

49030 TSP For the SCHED_SPORADIC policy, the required members of the *param* structure are
49031 *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and
49032 *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the
49033 specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.
49034 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
49035 function to succeed; if not, the function shall fail. It is unspecified whether the
49036 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
49037 rounded to align with the resolution of the clock being used. +

49038 **RETURN VALUE**

49039 If successful, the *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions shall
49040 return zero; otherwise, an error number shall be returned to indicate the error.

49041 **ERRORS**

49042 The *pthread_attr_getschedparam()* function may fail if:

49043 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
49044 object.

49045 The *pthread_attr_setschedparam()* function may fail if:

49046 [EINVAL] The value of *param* is not valid, or the value specified by *attr* does not refer to
49047 an initialized thread attribute object.

49048 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49049 These functions shall not return an error code of [EINTR].

49050 **EXAMPLES**

49051 None.

49052 **APPLICATION USAGE**

49053 After these attributes have been set, a thread can be created with the specified attributes using
49054 *pthread_create()*. Using these routines does not affect the current running thread.

49055 **RATIONALE**

49056 None.

49057 **FUTURE DIRECTIONS**

49058 None.

49059 **SEE ALSO**49060 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
49061 *pthread_attr_getschedpolicy()*, *pthread_create()*

49062 XBD <pthread.h>, <sched.h> +

49063 **CHANGE HISTORY**

49064 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49065 **Issue 6**49066 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are marked as part
49067 of the Threads option.

49068 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

49069 The **restrict** keyword is added to the *pthread_attr_getschedparam()* and
49070 *pthread_attr_setschedparam()* prototypes for alignment with the ISO/IEC 9899:1999 standard.49071 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/78 is applied, updating the ERRORS
49072 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
49073 object.49074 **Issue 7**49075 The *pthread_attr_getschedparam()* and *pthread_attr_setschedparam()* functions are moved from the
49076 Threads option to the Base.

49077 Austin Group Interpretation 1003.1-2001 #119 is applied. +

49078 **NAME**

49079 pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy
 49080 attribute (**REALTIME THREADS**)

49081 **SYNOPSIS**

```
49082 TPS #include <pthread.h>
49083
49083 int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
49084 int *restrict policy);
49085 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

49086 **DESCRIPTION**

49087 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions, respectively, shall
 49088 get and set the *schedpolicy* attribute in the *attr* argument.

49089 The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER,
 49090 which are defined in the **<sched.h>** header. When threads executing with the scheduling policy
 49091 TSP SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire
 49092 the mutex in priority order when the mutex is unlocked.

49093 **RETURN VALUE**

49094 If successful, the *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions shall
 49095 return zero; otherwise, an error number shall be returned to indicate the error.

49096 **ERRORS**

49097 The *pthread_attr_getschedpolicy()* function may fail if:

49098 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 49099 object.

49100 The *pthread_attr_setschedpolicy()* function may fail if:

49101 [EINVAL] The value of *policy* is not valid, or the value specified by *attr* does not refer to
 49102 an initialized thread attribute object.

49103 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49104 These functions shall not return an error code of [EINTR].

49105 **EXAMPLES**

49106 None.

49107 **APPLICATION USAGE**

49108 After these attributes have been set, a thread can be created with the specified attributes using
 49109 *pthread_create()*. Using these routines does not affect the current running thread.

49110 See [Section 2.9.4](#) (on page 488) for further details on thread scheduling attributes and their
 49111 default settings.

49112 **RATIONALE**

49113 None.

49114 **FUTURE DIRECTIONS**

49115 None.

pthread_attr_getschedpolicy()49116 **SEE ALSO**

49117 *pthread_attr_destroy()*, *pthread_attr_getscope()*, *pthread_attr_getinheritsched()*,
 49118 *pthread_attr_getschedparam()*, *pthread_create()*

49119 XBD <pthread.h>, <sched.h> +

49120 **CHANGE HISTORY**

49121 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49122 Marked as part of the Realtime Threads Feature Group.

49123 **Issue 6**

49124 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are marked as part of
 49125 the Threads and Thread Execution Scheduling options.

49126 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 49127 implementation does not support the Thread Execution Scheduling option.

49128 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

49129 The **restrict** keyword is added to the *pthread_attr_getschedpolicy()* prototype for alignment with
 49130 the ISO/IEC 9899:1999 standard.

49131 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/79 is applied, adding a reference to
 49132 [Section 2.9.4](#) (on page 488) in the APPLICATION USAGE section.

49133 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/80 is applied, updating the ERRORS
 49134 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
 49135 object.

49136 **Issue 7**

49137 The *pthread_attr_getschedpolicy()* and *pthread_attr_setschedpolicy()* functions are moved from the
 49138 Threads option.

49139 **NAME**

49140 pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute
 49141 (**REALTIME THREADS**)

49142 **SYNOPSIS**

```
49143 TPS #include <pthread.h>
49144
49144 int pthread_attr_getscope(const pthread_attr_t *restrict attr,
49145 int *restrict contentionscope);
49146 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

49147 **DESCRIPTION**

49148 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions, respectively, shall get and set
 49149 the *contentionscope* attribute in the *attr* object.

49150 The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying
 49151 system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process
 49152 scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and
 49153 PTHREAD_SCOPE_PROCESS are defined in the **<pthread.h>** header.

49154 **RETURN VALUE**

49155 If successful, the *pthread_attr_getscope()* and *pthread_attr_setscope()* functions shall return zero;
 49156 otherwise, an error number shall be returned to indicate the error.

49157 **ERRORS**

49158 The *pthread_attr_getscope()* function may fail if:

49159 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 49160 object.

49161 The *pthread_attr_setscope()* function may fail if:

49162 [EINVAL] The value of *contentionscope* is not valid, or the value specified by *attr* does not
 49163 refer to an initialized thread attribute object.

49164 [ENOTSUP] An attempt was made to set the attribute to an unsupported value.

49165 These functions shall not return an error code of [EINTR].

49166 **EXAMPLES**

49167 None.

49168 **APPLICATION USAGE**

49169 After these attributes have been set, a thread can be created with the specified attributes using
 49170 *pthread_create()*. Using these routines does not affect the current running thread.

49171 See [Section 2.9.4](#) (on page 488) for further details on thread scheduling attributes and their
 49172 default settings.

49173 **RATIONALE**

49174 None.

49175 **FUTURE DIRECTIONS**

49176 None.

49177 **SEE ALSO**

49178 *pthread_attr_destroy()*, *pthread_attr_getinheritsched()*, *pthread_attr_getschedpolicy()*,
 49179 *pthread_attr_getschedparam()*, *pthread_create()*

49180 XBD <pthread.h>, <sched.h>

49181 **CHANGE HISTORY**

49182 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49183 Marked as part of the Realtime Threads Feature Group.

49184 **Issue 6**

49185 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are marked as part of the
 49186 Threads and Thread Execution Scheduling options.

49187 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 49188 implementation does not support the Thread Execution Scheduling option.

49189 The **restrict** keyword is added to the *pthread_attr_getscope()* prototype for alignment with the
 49190 ISO/IEC 9899:1999 standard.

49191 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/81 is applied, adding a reference to
 49192 [Section 2.9.4](#) (on page 488) in the APPLICATION USAGE section.

49193 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/82 is applied, updating the ERRORS
 49194 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
 49195 object.

49196 **Issue 7**

49197 The *pthread_attr_getscope()* and *pthread_attr_setscope()* functions are moved from the Threads
 49198 option.

49199 **NAME**

49200 pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

49201 **SYNOPSIS**

```

49202 TSA TSS #include <pthread.h>
49203
49204 int pthread_attr_getstack(const pthread_attr_t *restrict attr,
49205 void **restrict stackaddr, size_t *restrict stacksize);
49206 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
49207 size_t stacksize);

```

49207 **DESCRIPTION**

49208 The *pthread_attr_getstack()* and *pthread_attr_setstack()* functions, respectively, shall get and set the
 49209 thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

49210 The stack attributes specify the area of storage to be used for the created thread's stack. The base
 49211 (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be
 49212 *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The
 49213 *pthread_attr_setstack()* function may fail with [EINVAL] if *stackaddr* does not meet
 49214 implementation-defined alignment requirements. All pages within the stack described by
 49215 *stackaddr* and *stacksize* shall be both readable and writable by the thread.

49216 If the *pthread_attr_getstack()* function is called before the *stackaddr* attribute has been set, the
 49217 behavior is unspecified.

49218 **RETURN VALUE**

49219 Upon successful completion, these functions shall return a value of 0; otherwise, an error
 49220 number shall be returned to indicate the error.

49221 The *pthread_attr_getstack()* function shall store the stack attribute values in *stackaddr* and *stacksize*
 49222 if successful.

49223 **ERRORS**

49224 The *pthread_attr_setstack()* function shall fail if:

49225 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an
 49226 implementation-defined limit.

49227 The *pthread_attr_getstack()* function may fail if:

49228 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
 49229 object.

49230 The *pthread_attr_setstack()* function may fail if:

49231 [EINVAL] The value of *stackaddr* does not have proper alignment to be used as a stack, or
 49232 ((**char** *)*stackaddr* + *stacksize*) lacks proper alignment, or the value specified by
 49233 *attr* does not refer to an initialized thread attribute object.

49234 [EACCES] The stack page(s) described by *stackaddr* and *stacksize* are not both readable
 49235 and writable by the thread.

49236 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, "overflow" might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

After a successful call to *pthread_attr_setstack()*, the storage area specified by the *stackaddr* parameter is under the control of the implementation, as described in [Section 2.9.8](#) (on page 495).

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these functions impossible. For example, the standard allows implementations to impose arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer obtained from *malloc()* is suitably aligned. Note that although the *stacksize* value passed to *pthread_attr_setstack()* must satisfy alignment requirements, the same is not true for *pthread_attr_setstacksize()* where the implementation must increase the specified size if necessary to achieve the proper alignment.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_attr_destroy\(\)*](#), [*pthread_attr_getdetachstate\(\)*](#), [*pthread_attr_getstacksize\(\)*](#), [*pthread_create\(\)*](#)

XBD [`<limits.h>`](#), [`<pthread.h>`](#)

CHANGE HISTORY

First released in Issue 6. Developed as part of the XSI option and brought into the BASE by IEEE PASC Interpretation 1003.1 #101.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/83 is applied, updating the APPLICATION USAGE section to refer to [Section 2.9.8](#) (on page 495).

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC/D6/84 is applied, updating the ERRORS section to include optional errors for the case when *attr* refers to an uninitialized thread attribute object.

Issue 7

SD5-XSH-ERN-66 is applied, correcting the use of *attr* in the [EINVAL] error condition.

Austin Group Interpretation 1003.1-2001 #057 is applied, clarifying the behavior if the function is called before the *stackaddr* attribute is set.

SD5-XSH-ERN-157 is applied, updating the APPLICATION USAGE section.

The description of the *stackaddr* attribute is updated in the DESCRIPTION and APPLICATION USAGE sections.

49279 **NAME**

49280 pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

49281 **SYNOPSIS**

```
49282 TSS      #include <pthread.h>
49283
49283      int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
49284                                  size_t *restrict stacksize);
49285      int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

49286 **DESCRIPTION**49287 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions, respectively, shall get and
49288 set the thread creation *stacksize* attribute in the *attr* object.49289 The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created
49290 threads stack.49291 **RETURN VALUE**49292 Upon successful completion, *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* shall
49293 return a value of 0; otherwise, an error number shall be returned to indicate the error.49294 The *pthread_attr_getstacksize()* function stores the *stacksize* attribute value in *stacksize* if
49295 successful.49296 **ERRORS**49297 The *pthread_attr_setstacksize()* function shall fail if:49298 [EINVAL] The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a
49299 system-imposed limit.

49300 These functions may fail if:

49301 [EINVAL] The value specified by *attr* does not refer to an initialized thread attribute
49302 object.

49303 These functions shall not return an error code of [EINTR].

49304 **EXAMPLES**

49305 None.

49306 **APPLICATION USAGE**

49307 None.

49308 **RATIONALE**

49309 None.

49310 **FUTURE DIRECTIONS**

49311 None.

49312 **SEE ALSO**49313 *pthread_attr_destroy()*, *pthread_attr_getdetachstate()*, *pthread_create()*

49314 XBD <limits.h>, <pthread.h>

49315 **CHANGE HISTORY**

49316 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

49317 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are marked as part of the
49318 Threads and Thread Stack Size Attribute options.
49319

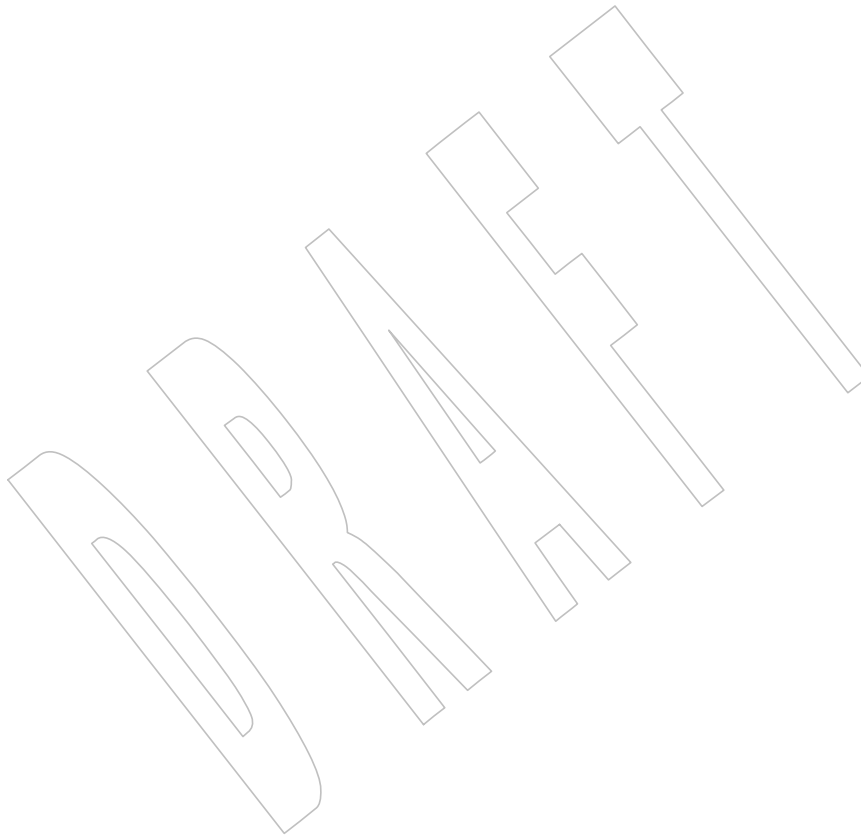
49320 The **restrict** keyword is added to the *pthread_attr_getstacksize()* prototype for alignment with the
49321 ISO/IEC 9899:1999 standard.

49322 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/43 is applied, correcting the margin code
49323 in the SYNOPSIS from TSA to TSS and updating the CHANGE HISTORY from “Thread Stack
49324 Address Attribute” option to “Thread Stack Size Attribute” option.

49325 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/87 is applied, updating the ERRORS
49326 section to include optional errors for the case when *attr* refers to an uninitialized thread attribute
49327 object.

Issue 7

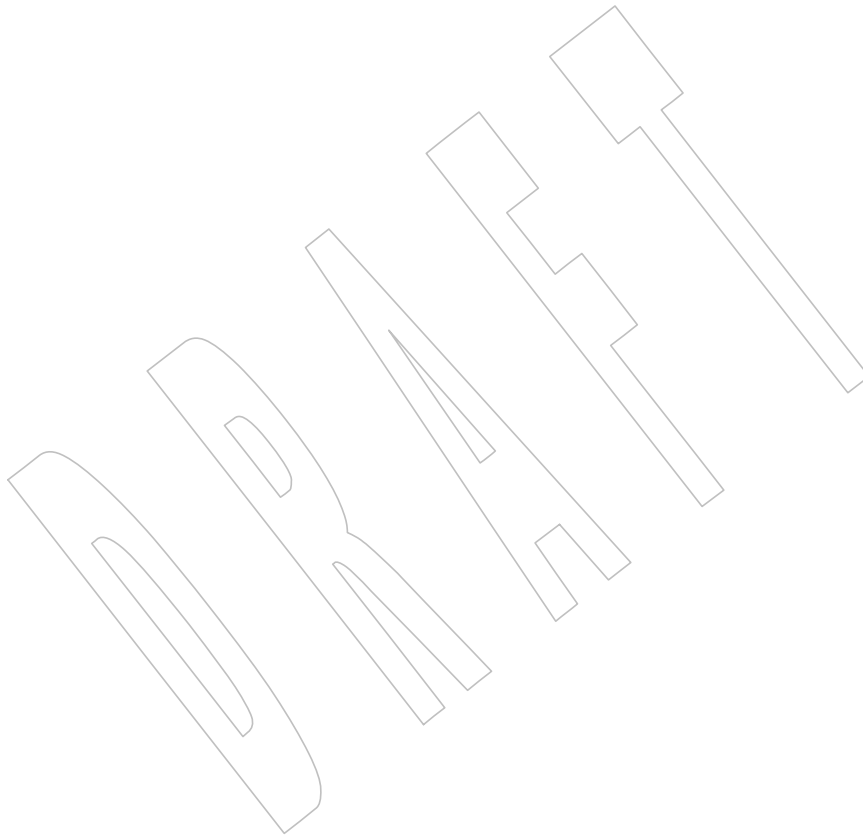
49328 The *pthread_attr_getstacksize()* and *pthread_attr_setstacksize()* functions are moved from the
49329 Threads option.
49330



49331 **NAME**
49332 pthread_attr_init — initialize the thread attributes object

49333 **SYNOPSIS**
49334 #include <pthread.h>
49335 int pthread_attr_init(pthread_attr_t *attr);

49336 **DESCRIPTION**
49337 Refer to *pthread_attr_destroy()*.



pthread_attr_setdetachstate()

49338

NAME

49339

pthread_attr_setdetachstate — set the detachstate attribute

49340

SYNOPSIS

49341

#include <pthread.h>

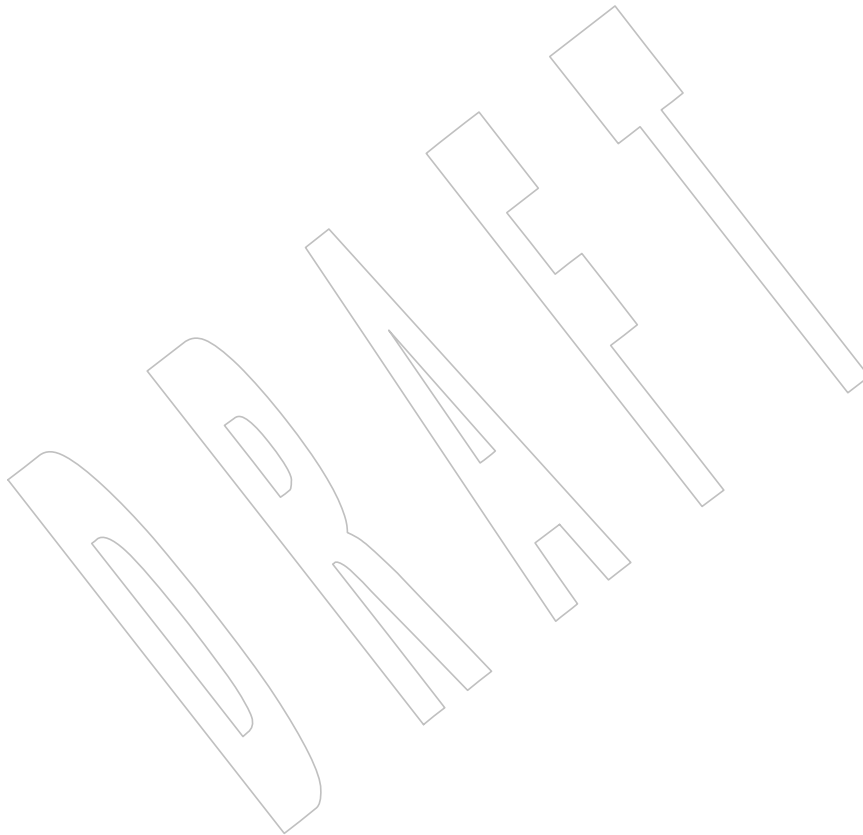
49342

int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);

49343

DESCRIPTION

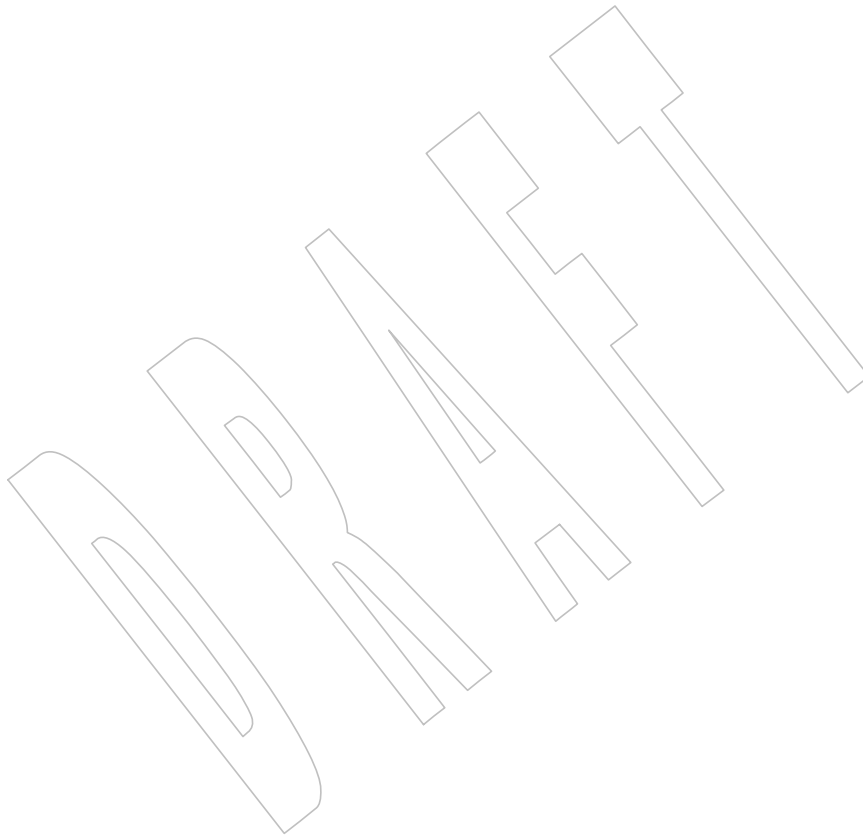
49344

Refer to *pthread_attr_getdetachstate()*.

49345 **NAME**
49346 pthread_attr_setguardsize — set the thread guardsize attribute

49347 **SYNOPSIS**
49348 #include <pthread.h>
49349 int pthread_attr_setguardsize(pthread_attr_t *attr,
49350 size_t guardsize);

49351 **DESCRIPTION**
49352 Refer to *pthread_attr_getguardsize()*.



pthread_attr_setinheritsched()*System Interfaces*49353 **NAME**49354 pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)49355 **SYNOPSIS**

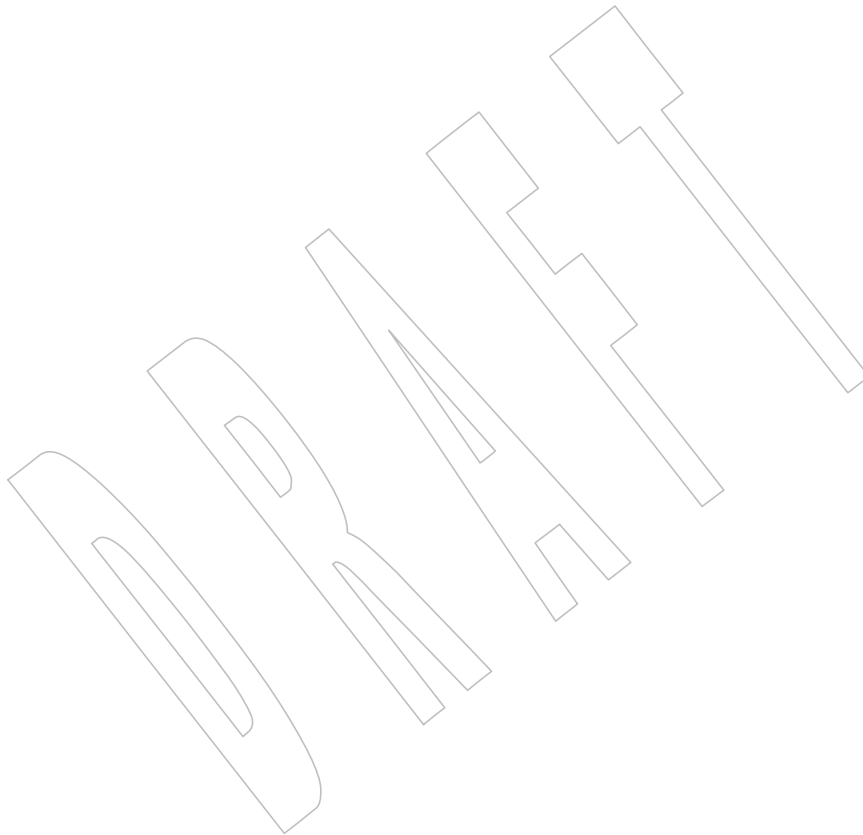
49356 TPS #include <pthread.h>

49357 int pthread_attr_setinheritsched(pthread_attr_t *attr,
49358 int inheritsched);49359 **DESCRIPTION**49360 Refer to *pthread_attr_getinheritsched()*.

49361 **NAME**
49362 pthread_attr_setschedparam — set the schedparam attribute

49363 **SYNOPSIS**
49364 #include <pthread.h>
49365 int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
49366 const struct sched_param *restrict param);

49367 **DESCRIPTION**
49368 Refer to *pthread_attr_getschedparam()*.



pthread_attr_setschedpolicy()*System Interfaces*49369 **NAME**49370 pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)49371 **SYNOPSIS**49372 TPS #include <pthread.h>
49373 int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);49374 **DESCRIPTION**49375 Refer to *pthread_attr_getschedpolicy()*.

49376 **NAME**
49377 pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

49378 **SYNOPSIS**

49379 TPS #include <pthread.h>
49380 int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);

49381 **DESCRIPTION**

49382 Refer to [pthread_attr_getscope\(\)](#).

pthread_attr_setstack()*System Interfaces*49383 **NAME**

49384 pthread_attr_setstack — set the stack attribute

49385 **SYNOPSIS**

49386 TSA TSS #include <pthread.h>

49387 int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
49388 size_t stacksize);49389 **DESCRIPTION**49390 Refer to *pthread_attr_getstack()*.

49391 **NAME**
49392 pthread_attr_setstacksize — set the stacksize attribute

49393 **SYNOPSIS**

49394 TSS #include <pthread.h>
49395 int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);

49396 **DESCRIPTION**

49397 Refer to [pthread_attr_getstacksize\(\)](#).

49398 **NAME**

49399 pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object

49400 **SYNOPSIS**

```
49401 #include <pthread.h>
49402
49402 int pthread_barrier_destroy(pthread_barrier_t *barrier);
49403 int pthread_barrier_init(pthread_barrier_t *restrict barrier,
49404     const pthread_barrierattr_t *restrict attr, unsigned count);
```

49405 **DESCRIPTION**

49406 The *pthread_barrier_destroy()* function shall destroy the barrier referenced by *barrier* and release
 49407 any resources used by the barrier. The effect of subsequent use of the barrier is undefined until
 49408 the barrier is reinitialized by another call to *pthread_barrier_init()*. An implementation may use
 49409 this function to set *barrier* to an invalid value. The results are undefined if
 49410 *pthread_barrier_destroy()* is called when any thread is blocked on the barrier, or if this function is
 49411 called with an uninitialized barrier.

49412 The *pthread_barrier_init()* function shall allocate any resources required to use the barrier
 49413 referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is
 49414 NULL, the default barrier attributes shall be used; the effect is the same as passing the address of
 49415 a default barrier attributes object. The results are undefined if *pthread_barrier_init()* is called
 49416 when any thread is blocked on the barrier (that is, has not returned from the
 49417 *pthread_barrier_wait()* call). The results are undefined if a barrier is used without first being
 49418 initialized. The results are undefined if *pthread_barrier_init()* is called specifying an already
 49419 initialized barrier.

49420 The *count* argument specifies the number of threads that must call *pthread_barrier_wait()* before
 49421 any of them successfully return from the call. The value specified by *count* must be greater than
 49422 zero.

49423 If the *pthread_barrier_init()* function fails, the barrier shall not be initialized and the contents of
 49424 *barrier* are undefined.

49425 Only the object referenced by *barrier* may be used for performing synchronization. The result of
 49426 referring to copies of that object in calls to *pthread_barrier_destroy()* or *pthread_barrier_wait()* is
 49427 undefined.

49428 **RETURN VALUE**

49429 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 49430 be returned to indicate the error.

49431 **ERRORS**

49432 The *pthread_barrier_destroy()* function may fail if:

49433 [EBUSY] The implementation has detected an attempt to destroy a barrier while it is in
 49434 use (for example, while being used in a *pthread_barrier_wait()* call) by another
 49435 thread.

49436 [EINVAL] The value specified by *barrier* is invalid.

49437 The *pthread_barrier_init()* function shall fail if:

49438 [EAGAIN] The system lacks the necessary resources to initialize another barrier.

49439 [EINVAL] The value specified by *count* is equal to zero.

49440 [ENOMEM] Insufficient memory exists to initialize the barrier.

49441 The *pthread_barrier_init()* function may fail if:

49442 [EBUSY] The implementation has detected an attempt to reinitialize a barrier while it is
49443 in use (for example, while being used in a *pthread_barrier_wait()* call) by
49444 another thread.

49445 [EINVAL] The value specified by *attr* is invalid.

49446 These functions shall not return an error code of [EINTR].

49447 **EXAMPLES**

49448 None.

49449 **APPLICATION USAGE**

49450 None.

49451 **RATIONALE**

49452 None.

49453 **FUTURE DIRECTIONS**

49454 None.

49455 **SEE ALSO**

49456 *pthread_barrier_wait()*

49457 XBD <pthread.h>

49458 **CHANGE HISTORY**

49459 First released in Issue 6. Derived from IEEE Std. 1003.1j-2000.

49460 **Issue 7**

49461 The *pthread_barrier_destroy()* and *pthread_barrier_init()* functions are moved from the Barriers
49462 option to the Base.

49463 **NAME**

49464 pthread_barrier_wait — synchronize at a barrier

49465 **SYNOPSIS**

49466 #include <pthread.h>

49467 int pthread_barrier_wait(pthread_barrier_t *barrier);

49468 **DESCRIPTION**

49469 The *pthread_barrier_wait()* function shall synchronize participating threads at the barrier
 49470 referenced by *barrier*. The calling thread shall block until the required number of threads have
 49471 called *pthread_barrier_wait()* specifying the barrier.

49472 When the required number of threads have called *pthread_barrier_wait()* specifying the barrier,
 49473 the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified
 49474 thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall
 49475 be reset to the state it had as a result of the most recent *pthread_barrier_init()* function that
 49476 referenced it.

49477 The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in <pthread.h> and its value
 49478 shall be distinct from any other value returned by *pthread_barrier_wait()*.

49479 The results are undefined if this function is called with an uninitialized barrier.

49480 If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the
 49481 thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the
 49482 required number of threads have not arrived at the barrier during the execution of the signal
 49483 handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until
 49484 the thread in the signal handler returns from it, it is unspecified whether other threads may
 49485 proceed past the barrier once they have all reached it.

49486 A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to
 49487 use the same processing resources from eventually making forward progress in its execution.
 49488 Eligibility for processing resources shall be determined by the scheduling policy.

49489 **RETURN VALUE**

49490 Upon successful completion, the *pthread_barrier_wait()* function shall return
 49491 PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the
 49492 barrier and zero for each of the other threads. Otherwise, an error number shall be returned to
 49493 indicate the error.

49494 **ERRORS**

49495 The *pthread_barrier_wait()* function may fail if:

49496 [EINVAL] The value specified by *barrier* does not refer to an initialized barrier object.

49497 This function shall not return an error code of [EINTR].

49498 **EXAMPLES**

49499 None.

49500 **APPLICATION USAGE**

49501 Applications using this function may be subject to priority inversion, as discussed in XBD |
 49502 [Section 3.284](#) (on page 72).

49503
49504
49505
49506
49507
49508
49509
49510
49511
49512
49513
49514**RATIONALE**

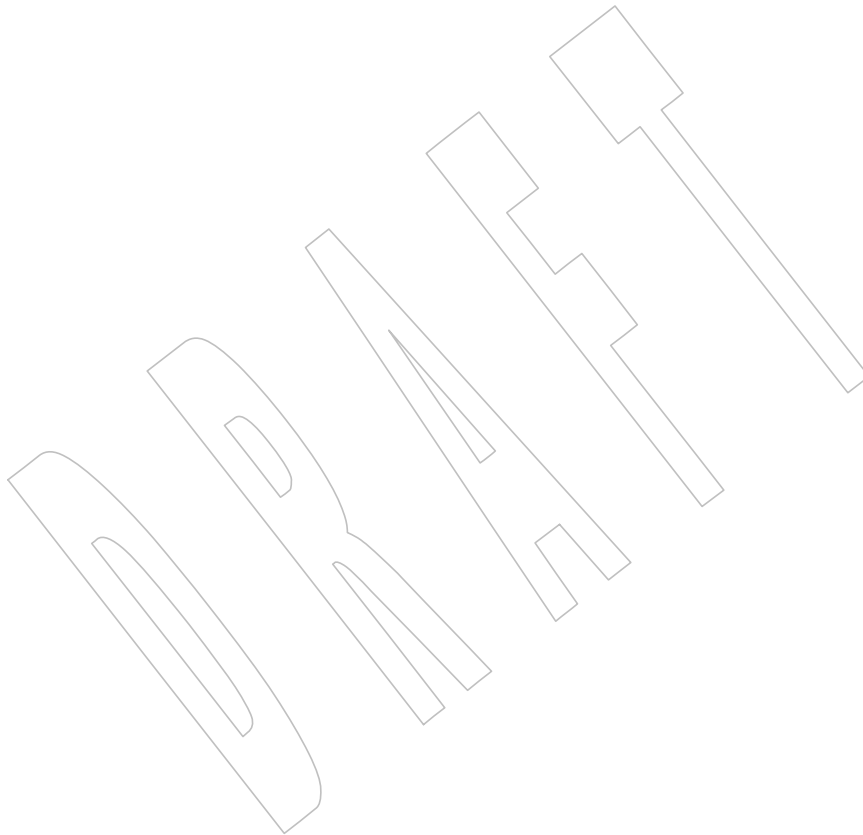
None.

FUTURE DIRECTIONS

None.

SEE ALSO*pthread_barrier_destroy()*XBD [Section 3.284](#) (on page 72), [Section 4.11](#) (on page 98), [<pthread.h>](#)**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.**Issue 7**The *pthread_barrier_wait()* function is moved from the Barriers option to the Base.

49515 **NAME**

49516 pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier
 49517 attributes object

49518 **SYNOPSIS**

49519 #include <pthread.h>

49520 int pthread_barrierattr_destroy(pthread_barrierattr_t *attr);

49521 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

49522 **DESCRIPTION**

49523 The *pthread_barrierattr_destroy()* function shall destroy a barrier attributes object. A destroyed
 49524 *attr* attributes object can be reinitialized using *pthread_barrierattr_init()*; the results of otherwise
 49525 referencing the object after it has been destroyed are undefined. An implementation may cause
 49526 *pthread_barrierattr_destroy()* to set the object referenced by *attr* to an invalid value.

49527 The *pthread_barrierattr_init()* function shall initialize a barrier attributes object *attr* with the
 49528 default value for all of the attributes defined by the implementation.

49529 Results are undefined if *pthread_barrierattr_init()* is called specifying an already initialized *attr*
 49530 attributes object.

49531 After a barrier attributes object has been used to initialize one or more barriers, any function
 49532 affecting the attributes object (including destruction) shall not affect any previously initialized
 49533 barrier.

49534 **RETURN VALUE**

49535 If successful, the *pthread_barrierattr_destroy()* and *pthread_barrierattr_init()* functions shall return
 49536 zero; otherwise, an error number shall be returned to indicate the error.

49537 **ERRORS**

49538 The *pthread_barrierattr_destroy()* function may fail if:

49539 [EINVAL] The value specified by *attr* is invalid.

49540 The *pthread_barrierattr_init()* function shall fail if:

49541 [ENOMEM] Insufficient memory exists to initialize the barrier attributes object.

49542 These functions shall not return an error code of [EINTR].

49543 **EXAMPLES**

49544 None.

49545 **APPLICATION USAGE**

49546 None.

49547 **RATIONALE**

49548 None.

49549 **FUTURE DIRECTIONS**

49550 None.

49551 **SEE ALSO**

49552 [*pthread_barrierattr_getpshared\(\)*](#)

49553 XBD [*<pthread.h>*](#)

49554

CHANGE HISTORY

49555

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

49556

In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

49557

Issue 7

49558

The `pthread_barrierattr_destroy()` and `pthread_barrierattr_init()` functions are moved from the Barriers option to the Base.

49559



49560 **NAME**

49561 pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-
 49562 shared attribute of the barrier attributes object

49563 **SYNOPSIS**

```
49564 TSH #include <pthread.h>
49565
49565 int pthread_barrierattr_getpshared(const pthread_barrierattr_t
49566     *restrict attr, int *restrict pshared);
49567 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
49568     int pshared);
```

49569 **DESCRIPTION**

49570 The *pthread_barrierattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 49571 from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared()* function shall
 49572 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

49573 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be
 49574 operated upon by any thread that has access to the memory where the barrier is allocated. If the
 49575 *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the barrier shall only be operated
 49576 upon by threads created within the same process as the thread that initialized the barrier; if
 49577 threads of different processes attempt to operate on such a barrier, the behavior is undefined.
 49578 The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE. Both constants
 49579 PTHREAD_PROCESS_SHARED and PTHREAD_PROCESS_PRIVATE are defined in
 49580 **<pthread.h>**.

49581 Additional attributes, their default values, and the names of the associated functions to get and
 49582 set those attribute values are implementation-defined.

49583 **RETURN VALUE**

49584 If successful, the *pthread_barrierattr_getpshared()* function shall return zero and store the value of
 49585 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 49586 an error number shall be returned to indicate the error.

49587 If successful, the *pthread_barrierattr_setpshared()* function shall return zero; otherwise, an error
 49588 number shall be returned to indicate the error.

49589 **ERRORS**

49590 These functions may fail if:

49591 [EINVAL] The value specified by *attr* is invalid.

49592 The *pthread_barrierattr_setpshared()* function may fail if:

49593 [EINVAL] The new value specified for the *process-shared* attribute is not one of the legal
 49594 values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

49595 These functions shall not return an error code of [EINTR].

49596
49597
49598
49599
49600
49601
49602
49603
49604
49605
49606
49607
49608
49609
49610
49611
49612
49613

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are part of the Thread Process-Shared Synchronization option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_destroy(), *pthread_barrierattr_destroy()*

XBD <pthread.h>

+

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1j-2000

Issue 7

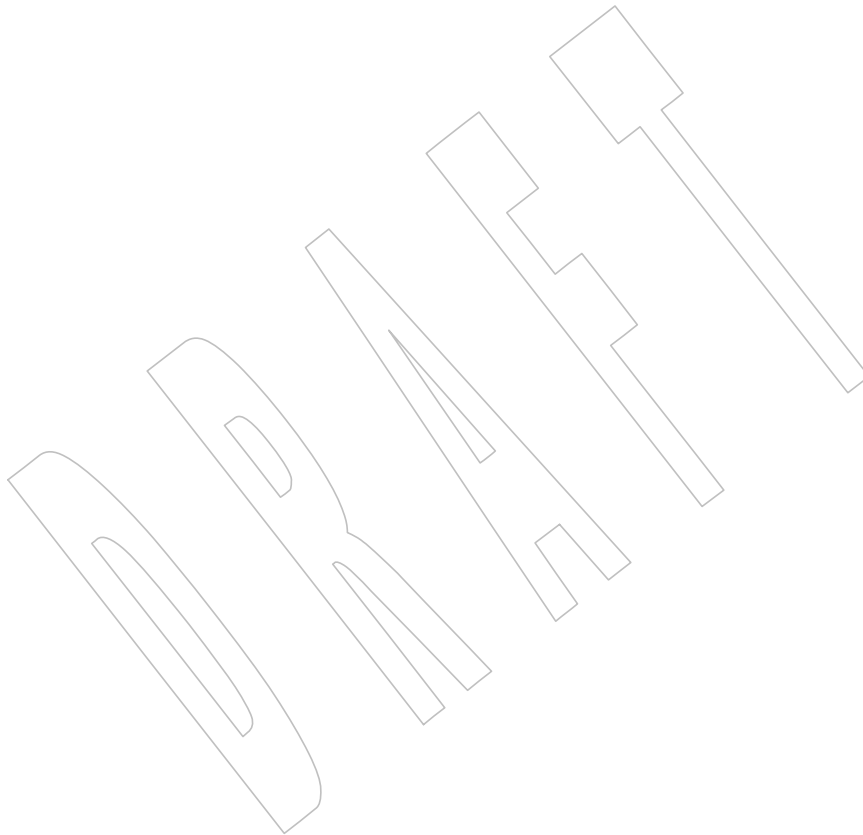
The *pthread_barrierattr_getpshared()* and *pthread_barrierattr_setpshared()* functions are moved from the Barriers option.

DRAFT

49614 **NAME**
49615 pthread_barrierattr_init — initialize the barrier attributes object

49616 **SYNOPSIS**
49617 #include <pthread.h>
49618 int pthread_barrierattr_init(pthread_barrierattr_t *attr);

49619 **DESCRIPTION**
49620 Refer to *pthread_barrierattr_destroy()*.



49621 **NAME**
49622 `pthread_barrierattr_setpshared` — set the process-shared attribute of the barrier attributes object

49623 **SYNOPSIS**

```
49624 TSH #include <pthread.h>  
49625 int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,  
49626 int pshared);
```

49627 **DESCRIPTION**

49628 Refer to [pthread_barrierattr_getpshared\(\)](#).

49629 **NAME**

49630 pthread_cancel — cancel execution of a thread

49631 **SYNOPSIS**

49632 #include <pthread.h>

49633 int pthread_cancel(pthread_t thread);

49634 **DESCRIPTION**

49635 The *pthread_cancel()* function shall request that *thread* be canceled. The target thread's
 49636 cancelability state and type determines when the cancellation takes effect. When the cancellation
 49637 is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last
 49638 cancellation cleanup handler returns, the thread-specific data destructor functions shall be called
 49639 for *thread*. When the last destructor function returns, *thread* shall be terminated.

49640 The cancellation processing in the target thread shall run asynchronously with respect to the
 49641 calling thread returning from *pthread_cancel()*.

49642 **RETURN VALUE**

49643 If successful, the *pthread_cancel()* function shall return zero; otherwise, an error number shall be
 49644 returned to indicate the error.

49645 **ERRORS**49646 The *pthread_cancel()* function may fail if:

49647 [ESRCH] No thread could be found corresponding to that specified by the given thread
 49648 ID.

49649 The *pthread_cancel()* function shall not return an error code of [EINTR].49650 **EXAMPLES**

49651 None.

49652 **APPLICATION USAGE**

49653 None.

49654 **RATIONALE**

49655 Two alternative functions were considered for sending the cancellation notification to a thread.
 49656 One would be to define a new SIGCANCEL signal that had the cancellation semantics when
 49657 delivered; the other was to define the new *pthread_cancel()* function, which would trigger the
 49658 cancellation semantics.

49659 The advantage of a new signal was that so much of the delivery criteria were identical to that
 49660 used when trying to deliver a signal that making cancellation notification a signal was seen as
 49661 consistent. Indeed, many implementations implement cancellation using a special signal. On the
 49662 other hand, there would be no signal functions that could be used with this signal except
 49663 *pthread_kill()*, and the behavior of the delivered cancellation signal would be unlike any
 49664 previously existing defined signal.

49665 The benefits of a special function include the recognition that this signal would be defined
 49666 because of the similar delivery criteria and that this is the only common behavior between a
 49667 cancellation request and a signal. In addition, the cancellation delivery mechanism does not
 49668 have to be implemented as a signal. There are also strong, if not stronger, parallels with
 49669 language exception mechanisms than with signals that are potentially obscured if the delivery
 49670 mechanism is visibly closer to signals.

49671 In the end, it was considered that as there were so many exceptions to the use of the new signal
 49672 with existing signals functions it would be misleading. A special function has resolved this
 49673 problem. This function was carefully defined so that an implementation wishing to provide the

49674 cancellation functions on top of signals could do so. The special function also means that
49675 implementations are not obliged to implement cancellation with signals.

FUTURE DIRECTIONS

49676 None.
49677

SEE ALSO

49678 *pthread_exit()*, *pthread_cond_timedwait()*, *pthread_join()*, *pthread_setcancelstate()*
49679

49680 XBD <pthread.h> +

CHANGE HISTORY

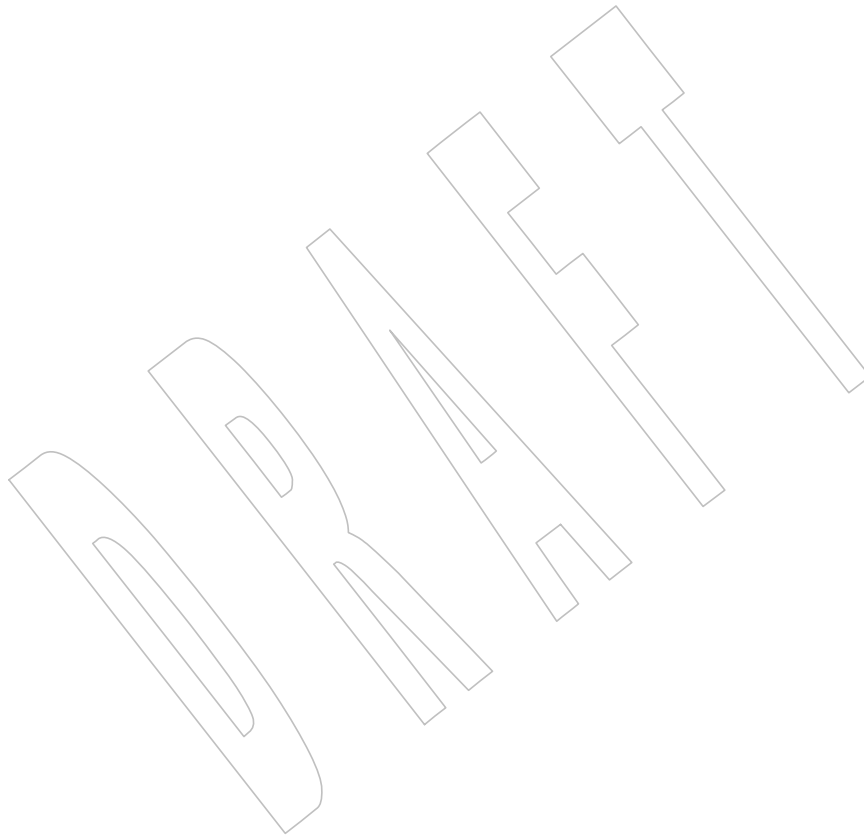
49681 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
49682

Issue 6

49683 The *pthread_cancel()* function is marked as part of the Threads option.
49684

Issue 7

49685 The *pthread_cancel()* function is moved from the Threads option to the Base.
49686



49687 **NAME**

49688 pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

49689 **SYNOPSIS**

49690 #include <pthread.h>

49691 void pthread_cleanup_pop(int *execute*);49692 void pthread_cleanup_push(void (**routine*)(void*), void **arg*);49693 **DESCRIPTION**49694 The *pthread_cleanup_pop()* function shall remove the routine at the top of the calling thread's
49695 cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).49696 The *pthread_cleanup_push()* function shall push the specified cancellation cleanup handler *routine*
49697 onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be
49698 popped from the cancellation cleanup stack and invoked with the argument *arg* when:

- 49699
- The thread exits (that is, calls *pthread_exit()*).
 - The thread acts upon a cancellation request.
 - The thread calls *pthread_cleanup_pop()* with a non-zero *execute* argument.

49702 These functions may be implemented as macros. The application shall ensure that they appear
49703 as statements, and in pairs within the same lexical scope (that is, the *pthread_cleanup_push()*
49704 macro may be thought to expand to a token list whose first token is '{' with
49705 *pthread_cleanup_pop()* expanding to a token list whose last token is the corresponding '}').49706 The effect of calling *longjmp()* or *siglongjmp()* is undefined if there have been any calls to
49707 *pthread_cleanup_push()* or *pthread_cleanup_pop()* made without the matching call since the jump
49708 buffer was filled. The effect of calling *longjmp()* or *siglongjmp()* from inside a cancellation
49709 cleanup handler is also undefined unless the jump buffer was also filled in the cancellation
49710 cleanup handler.49711 The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block
49712 described by a pair of *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions calls is
49713 undefined.49714 **RETURN VALUE**49715 The *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions shall not return a value.49716 **ERRORS**

49717 No errors are defined.

49718 These functions shall not return an error code of [EINTR].

49719 **EXAMPLES**49720 The following is an example using thread primitives to implement a cancelable, writers-priority
49721 read-write lock:49722 typedef struct {
49723 pthread_mutex_t lock;
49724 pthread_cond_t rcond,
49725 wcond;
49726 int lock_count; /* < 0 .. Held by writer. */
49727 /* > 0 .. Held by lock_count readers. */
49728 /* = 0 .. Held by nobody. */
49729 int waiting_writers; /* Count of waiting writers. */
49730 } rwlock;


```

49731     void
49732     waiting_reader_cleanup(void *arg)
49733     {
49734         rwlock *l;
49735
49736         l = (rwlock *) arg;
49737         pthread_mutex_unlock(&l->lock);
49738     }
49739
49740     void
49741     lock_for_read(rwlock *l)
49742     {
49743         pthread_mutex_lock(&l->lock);
49744         pthread_cleanup_push(waiting_reader_cleanup, l);
49745         while ((l->lock_count < 0) && (l->waiting_writers != 0))
49746             pthread_cond_wait(&l->rcond, &l->lock);
49747         l->lock_count++;
49748         /*
49749          * Note the pthread_cleanup_pop executes
49750          * waiting_reader_cleanup.
49751          */
49752         pthread_cleanup_pop(1);
49753     }
49754
49755     void
49756     release_read_lock(rwlock *l)
49757     {
49758         pthread_mutex_lock(&l->lock);
49759         if (--l->lock_count == 0)
49760             pthread_cond_signal(&l->wcond);
49761         pthread_mutex_unlock(l);
49762     }
49763
49764     void
49765     waiting_writer_cleanup(void *arg)
49766     {
49767         rwlock *l;
49768
49769         l = (rwlock *) arg;
49770         if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
49771             /*
49772              * This only happens if we have been canceled.
49773              */
49774             pthread_cond_broadcast(&l->wcond);
49775         }
49776         pthread_mutex_unlock(&l->lock);
49777     }
49778
49779     void
49780     lock_for_write(rwlock *l)
49781     {
49782         pthread_mutex_lock(&l->lock);
49783         l->waiting_writers++;
49784         pthread_cleanup_push(waiting_writer_cleanup, l);
49785         while (l->lock_count != 0)
49786             pthread_cond_wait(&l->wcond, &l->lock);
49787         l->lock_count = -1;

```

```

49782     /*
49783     * Note the pthread_cleanup_pop executes
49784     * waiting_writer_cleanup.
49785     */
49786     pthread_cleanup_pop(1);
49787 }
49788
49789 void
49790 release_write_lock(rwlock *l)
49791 {
49792     pthread_mutex_lock(&l->lock);
49793     l->lock_count = 0;
49794     if (l->waiting_writers == 0)
49795         pthread_cond_broadcast(&l->rcond)
49796     else
49797         pthread_cond_signal(&l->wcond);
49798     pthread_mutex_unlock(&l->lock);
49799 }
49800 /*
49801 * This function is called to initialize the read/write lock.
49802 */
49803 void
49804 initialize_rwlock(rwlock *l)
49805 {
49806     pthread_mutex_init(&l->lock, pthread_mutexattr_default);
49807     pthread_cond_init(&l->wcond, pthread_condattr_default);
49808     pthread_cond_init(&l->rcond, pthread_condattr_default);
49809     l->lock_count = 0;
49810     l->waiting_writers = 0;
49811 }
49812
49813 reader_thread()
49814 {
49815     lock_for_read(&lock);
49816     pthread_cleanup_push(release_read_lock, &lock);
49817     /*
49818     * Thread has read lock.
49819     */
49820     pthread_cleanup_pop(1);
49821 }
49822
49823 writer_thread()
49824 {
49825     lock_for_write(&lock);
49826     pthread_cleanup_push(release_write_lock, &lock);
49827     /*
49828     * Thread has write lock.
49829     */
49830     pthread_cleanup_pop(1);
49831 }

```

APPLICATION USAGE

The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and *pthread_cleanup_pop()*, can be thought of as left and right parentheses. They always need to be matched.

RATIONALE

The restriction that the two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push()* and *pthread_cleanup_pop()*, have to appear in the same lexical scope allows for efficient macro or compiler implementations and efficient storage management. A sample implementation of these routines as macros might look like this:

```

49838 #define pthread_cleanup_push(rtn,arg) { \
49839     struct _pthread_handler_rec __cleanup_handler, **__head; \
49840     __cleanup_handler.rtn = rtn; \
49841     __cleanup_handler.arg = arg; \
49842     (void) pthread_getspecific(_pthread_handler_key, &__head); \
49843     __cleanup_handler.next = *__head; \
49844     *__head = &__cleanup_handler;
49845
49846 #define pthread_cleanup_pop(ex) \
49847     *__head = __cleanup_handler.next; \
49848     if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
49849 }

```

A more ambitious implementation of these routines might do even better by allowing the compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

This volume of POSIX.1-200x currently leaves unspecified the effect of calling *longjmp()* from a signal handler executing in a POSIX System Interfaces function. If an implementation wants to allow this and give the programmer reasonable behavior, the *longjmp()* function has to call all cancellation cleanup handlers that have been pushed but not popped since the time *setjmp()* was called.

Consider a multi-threaded function called by a thread that uses signals. If a signal were delivered to a signal handler during the operation of *qsort()* and that handler were to call *longjmp()* (which, in turn, did *not* call the cancellation cleanup handlers), the helper threads created by the *qsort()* function would not be canceled. Instead, they would continue to execute and write into the argument array even though the array might have been popped off the stack.

Note that the specified cleanup handling mechanism is especially tied to the C language and, while the requirement for a uniform mechanism for expressing cleanup is language-independent, the mechanism used in other languages may be quite different. In addition, this mechanism is really only necessary due to the lack of a real exception mechanism in the C language, which would be the ideal solution.

There is no notion of a cancellation cleanup-safe function. If an application has no cancellation points in its signal handlers, blocks any signal whose handler may have cancellation points while calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all functions may be safely called from cancellation cleanup routines.

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_cancel\(\)*](#), [*pthread_setcancelstate\(\)*](#)

XBD [**<pthread.h>**](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

pthread_cleanup_pop()49877
49878
49879
49880
49881
49882
49883
49884
49885
49886
49887**Issue 6**

The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are marked as part of the Threads option.

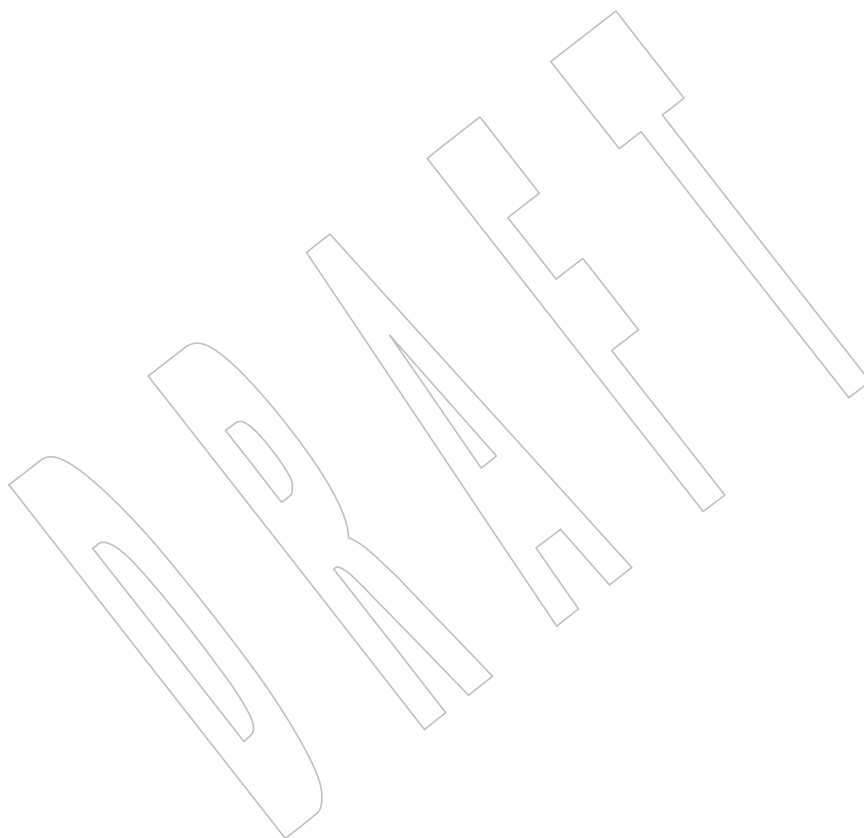
The APPLICATION USAGE section is added.

The normative text is updated to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/88 is applied, updating the DESCRIPTION to describe the consequences of prematurely leaving a code block defined by the *pthread_cleanup_push()* and *pthread_cleanup_pop()* functions.

Issue 7

The *pthread_cleanup_pop()* and *pthread_cleanup_push()* functions are moved from the Threads option to the Base.



49888 **NAME**

49889 pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

49890 **SYNOPSIS**

49891 #include <pthread.h>

49892 int pthread_cond_broadcast(pthread_cond_t *cond);

49893 int pthread_cond_signal(pthread_cond_t *cond);

49894 **DESCRIPTION**

49895 These functions shall unblock threads blocked on a condition variable.

49896 The *pthread_cond_broadcast()* function shall unblock all threads currently blocked on the
49897 specified condition variable *cond*.49898 The *pthread_cond_signal()* function shall unblock at least one of the threads that are blocked on
49899 the specified condition variable *cond* (if any threads are blocked on *cond*).49900 If more than one thread is blocked on a condition variable, the scheduling policy shall determine
49901 the order in which threads are unblocked. When each thread unblocked as a result of a
49902 *pthread_cond_broadcast()* or *pthread_cond_signal()* returns from its call to *pthread_cond_wait()* or
49903 *pthread_cond_timedwait()*, the thread shall own the mutex with which it called
49904 *pthread_cond_wait()* or *pthread_cond_timedwait()*. The thread(s) that are unblocked shall contend
49905 for the mutex according to the scheduling policy (if applicable), and as if each had called
49906 *pthread_mutex_lock()*.49907 The *pthread_cond_broadcast()* or *pthread_cond_signal()* functions may be called by a thread
49908 whether or not it currently owns the mutex that threads calling *pthread_cond_wait()* or
49909 *pthread_cond_timedwait()* have associated with the condition variable during their waits;
49910 however, if predictable scheduling behavior is required, then that mutex shall be locked by the
49911 thread calling *pthread_cond_broadcast()* or *pthread_cond_signal()*.49912 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall have no effect if there are
49913 no threads currently blocked on *cond*.49914 **RETURN VALUE**49915 If successful, the *pthread_cond_broadcast()* and *pthread_cond_signal()* functions shall return zero;
49916 otherwise, an error number shall be returned to indicate the error.49917 **ERRORS**49918 The *pthread_cond_broadcast()* and *pthread_cond_signal()* function may fail if:49919 [EINVAL] The value *cond* does not refer to an initialized condition variable.

49920 These functions shall not return an error code of [EINTR].

49921 **EXAMPLES**

49922 None.

49923 **APPLICATION USAGE**49924 The *pthread_cond_broadcast()* function is used whenever the shared-variable state has been
49925 changed in a way that more than one thread can proceed with its task. Consider a single
49926 producer/multiple consumer problem, where the producer can insert multiple items on a list
49927 that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast()*
49928 function, the producer would notify all consumers that might be waiting, and thereby the
49929 application would receive more throughput on a multi-processor. In addition,
49930 *pthread_cond_broadcast()* makes it easier to implement a read-write lock. The
49931 *pthread_cond_broadcast()* function is needed in order to wake up all waiting readers when a
49932 writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function

49933 to notify all clients of an impending transaction commit.

49934 It is not safe to use the *pthread_cond_signal()* function in a signal handler that is invoked
49935 asynchronously. Even if it were safe, there would still be a race between the test of the Boolean
49936 *pthread_cond_wait()* that could not be efficiently eliminated.

49937 Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling
49938 from code running in a signal handler.

49939 RATIONALE

49940 Multiple Awakenings by Condition Signal

49941 On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal()* to
49942 avoid the unblocking of more than one thread blocked on a condition variable. For example,
49943 consider the following partial implementation of *pthread_cond_wait()* and *pthread_cond_signal()*,
49944 executed by two threads in the order given. One thread is trying to wait on the condition
49945 variable, another is concurrently executing *pthread_cond_signal()*, while a third thread is already
49946 waiting.

```

49947 pthread_cond_wait(mutex, cond):
49948     value = cond->value; /* 1 */
49949     pthread_mutex_unlock(mutex); /* 2 */
49950     pthread_mutex_lock(cond->mutex); /* 10 */
49951     if (value == cond->value) { /* 11 */
49952         me->next_cond = cond->waiter;
49953         cond->waiter = me;
49954         pthread_mutex_unlock(cond->mutex);
49955         unable_to_run(me);
49956     } else
49957         pthread_mutex_unlock(cond->mutex); /* 12 */
49958     pthread_mutex_lock(mutex); /* 13 */

49959 pthread_cond_signal(cond):
49960     pthread_mutex_lock(cond->mutex); /* 3 */
49961     cond->value++; /* 4 */
49962     if (cond->waiter) { /* 5 */
49963         sleeper = cond->waiter; /* 6 */
49964         cond->waiter = sleeper->next_cond; /* 7 */
49965         able_to_run(sleeper); /* 8 */
49966     }
49967     pthread_mutex_unlock(cond->mutex); /* 9 */

```

49968 The effect is that more than one thread can return from its call to *pthread_cond_wait()* or
49969 *pthread_cond_timedwait()* as a result of one call to *pthread_cond_signal()*. This effect is called
49970 “spurious wakeup”. Note that the situation is self-correcting in that the number of threads that
49971 are so awakened is finite; for example, the next thread to call *pthread_cond_wait()* after the
49972 sequence of events above blocks.

49973 While this problem could be resolved, the loss of efficiency for a fringe condition that occurs
49974 only rarely is unacceptable, especially given that one has to check the predicate associated with a
49975 condition variable anyway. Correcting this problem would unnecessarily reduce the degree of
49976 concurrency in this basic building block for all higher-level synchronization operations.

49977 An added benefit of allowing spurious wakeups is that applications are forced to code a
49978 predicate-testing-loop around the condition wait. This also makes the application tolerate
49979 superfluous condition broadcasts or signals on the same condition variable that may be coded in
49980 some other part of the application. The resulting applications are thus more robust. Therefore,
49981 POSIX.1-200x explicitly documents that spurious wakeups may occur.

49982 **FUTURE DIRECTIONS**

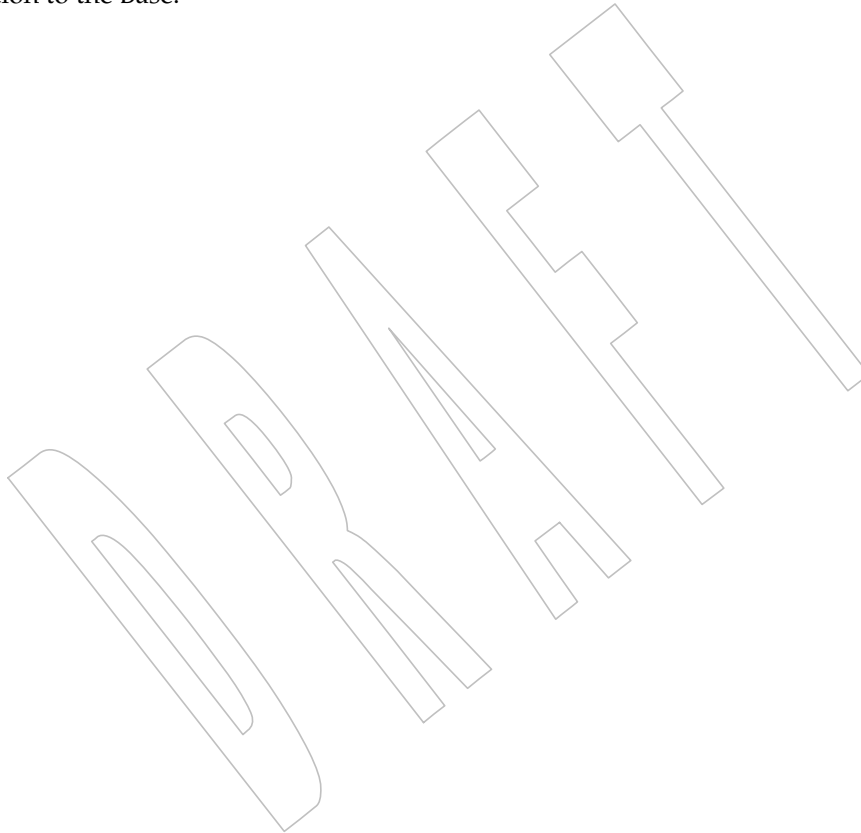
49983 None.

49984 **SEE ALSO**49985 *pthread_cond_destroy()*, *pthread_cond_timedwait()*49986 XBD Section 4.11 (on page 98), **<pthread.h>**49987 **CHANGE HISTORY**

49988 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

49989 **Issue 6**49990 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions are marked as part of the
49991 Threads option.

49992 The APPLICATION USAGE section is added.

49993 **Issue 7**49994 The *pthread_cond_broadcast()* and *pthread_cond_signal()* functions are moved from the Threads
49995 option to the Base.

49996 **NAME**

49997 pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

49998 **SYNOPSIS**

```
49999 #include <pthread.h>
50000
50000 int pthread_cond_destroy(pthread_cond_t *cond);
50001 int pthread_cond_init(pthread_cond_t *restrict cond,
50002     const pthread_condattr_t *restrict attr);
50003 pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

50004 **DESCRIPTION**

50005 The *pthread_cond_destroy()* function shall destroy the given condition variable specified by *cond*;
 50006 the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy()*
 50007 to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can
 50008 be reinitialized using *pthread_cond_init()*; the results of otherwise referencing the object after it
 50009 has been destroyed are undefined.

50010 It shall be safe to destroy an initialized condition variable upon which no threads are currently
 50011 blocked. Attempting to destroy a condition variable upon which other threads are currently
 50012 blocked results in undefined behavior.

50013 The *pthread_cond_init()* function shall initialize the condition variable referenced by *cond* with
 50014 attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be
 50015 used; the effect is the same as passing the address of a default condition variable attributes
 50016 object. Upon successful initialization, the state of the condition variable shall become initialized.

50017 Only *cond* itself may be used for performing synchronization. The result of referring to copies of
 50018 *cond* in calls to *pthread_cond_wait()*, *pthread_cond_timedwait()*, *pthread_cond_signal()*,
 50019 *pthread_cond_broadcast()*, and *pthread_cond_destroy()* is undefined.

50020 Attempting to initialize an already initialized condition variable results in undefined behavior.

50021 In cases where default condition variable attributes are appropriate, the macro
 50022 PTHREAD_COND_INITIALIZER can be used to initialize condition variables that are statically
 50023 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init()*
 50024 with parameter *attr* specified as NULL, except that no error checks are performed.

50025 **RETURN VALUE**

50026 If successful, the *pthread_cond_destroy()* and *pthread_cond_init()* functions shall return zero;
 50027 otherwise, an error number shall be returned to indicate the error.

50028 The [EBUSY] and [EINVAL] error checks, if implemented, shall act as if they were performed
 50029 immediately at the beginning of processing for the function and caused an error return prior to
 50030 modifying the state of the condition variable specified by *cond*.

50031 **ERRORS**

50032 The *pthread_cond_destroy()* function may fail if:

50033 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 50034 by *cond* while it is referenced (for example, while being used in a
 50035 *pthread_cond_wait()* or *pthread_cond_timedwait()*) by another thread.

50036 [EINVAL] The value specified by *cond* is invalid.

50037 The *pthread_cond_init()* function shall fail if:

- 50038 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
50039 another condition variable.
- 50040 [ENOMEM] Insufficient memory exists to initialize the condition variable.
- 50041 The *pthread_cond_init()* function may fail if:
- 50042 [EBUSY] The implementation has detected an attempt to reinitialize the object
50043 referenced by *cond*, a previously initialized, but not yet destroyed, condition
50044 variable.
- 50045 [EINVAL] The value specified by *attr* is invalid.
- 50046 These functions shall not return an error code of [EINTR].

EXAMPLES

50047 A condition variable can be destroyed immediately after all the threads that are blocked on it are
50048 awakened. For example, consider the following code:

```

50050 struct list {
50051     pthread_mutex_t lm;
50052     ...
50053 }
50054 struct elt {
50055     key k;
50056     int busy;
50057     pthread_cond_t notbusy;
50058     ...
50059 }
50060 /* Find a list element and reserve it. */
50061 struct elt *
50062 list_find(struct list *lp, key k)
50063 {
50064     struct elt *ep;
50065     pthread_mutex_lock(&lp->lm);
50066     while ((ep = find_elt(l, k) != NULL) && ep->busy)
50067         pthread_cond_wait(&ep->notbusy, &lp->lm);
50068     if (ep != NULL)
50069         ep->busy = 1;
50070     pthread_mutex_unlock(&lp->lm);
50071     return(ep);
50072 }
50073 delete_elt(struct list *lp, struct elt *ep)
50074 {
50075     pthread_mutex_lock(&lp->lm);
50076     assert(ep->busy);
50077     ... remove ep from list ...
50078     ep->busy = 0; /* Paranoid. */
50079     (A) pthread_cond_broadcast(&ep->notbusy);
50080     pthread_mutex_unlock(&lp->lm);
50081     (B) pthread_cond_destroy(&rp->notbusy);
50082     free(ep);
50083 }

```

50084 In this example, the condition variable and its list element may be freed (line B) immediately
50085 after all threads waiting for it are awakened (line A), since the mutex and the code ensure that

50086 no other thread can touch the element to be deleted.

50087 **APPLICATION USAGE**

50088 None.

50089 **RATIONALE**

50090 See *pthread_mutex_destroy()*; a similar rationale applies to condition variables.

50091 **FUTURE DIRECTIONS**

50092 None.

50093 **SEE ALSO**

50094 *pthread_cond_broadcast()*, *pthread_cond_timedwait()*, *pthread_mutex_destroy()* -

50095 XBD <pthread.h> |

50096 **CHANGE HISTORY**

50097 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50098 **Issue 6**

50099 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are marked as part of the Threads
50100 option.

50101 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

50102 The **restrict** keyword is added to the *pthread_cond_init()* prototype for alignment with the
50103 ISO/IEC 9899:1999 standard.

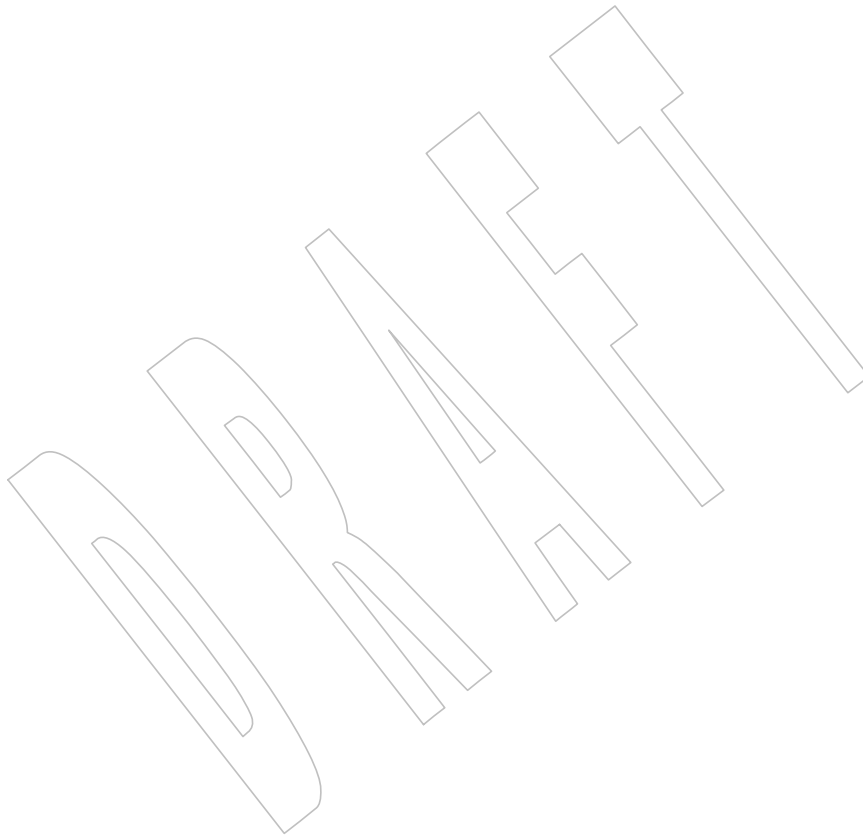
50104 **Issue 7**

50105 The *pthread_cond_destroy()* and *pthread_cond_init()* functions are moved from the Threads option
50106 to the Base.

50107 **NAME**
50108 pthread_cond_signal — signal a condition

50109 **SYNOPSIS**
50110 #include <pthread.h>
50111 int pthread_cond_signal(pthread_cond_t *cond);

50112 **DESCRIPTION**
50113 Refer to *pthread_cond_broadcast()*.



50114 **NAME**

50115 pthread_cond_timedwait, pthread_cond_wait — wait on a condition

50116 **SYNOPSIS**

50117 #include <pthread.h>

```

50118 int pthread_cond_timedwait(pthread_cond_t *restrict cond,
50119 pthread_mutex_t *restrict mutex,
50120 const struct timespec *restrict abstime);
50121 int pthread_cond_wait(pthread_cond_t *restrict cond,
50122 pthread_mutex_t *restrict mutex);

```

50123 **DESCRIPTION**

50124 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions shall block on a condition
50125 variable. They shall be called with *mutex* locked by the calling thread or undefined behavior
50126 results.

50127 These functions atomically release *mutex* and cause the calling thread to block on the condition
50128 variable *cond*; atomically here means “atomically with respect to access by another thread to the
50129 mutex and then the condition variable”. That is, if another thread is able to acquire the mutex
50130 after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast()*
50131 or *pthread_cond_signal()* in that thread shall behave as if it were issued after the about-to-block
50132 thread has blocked.

50133 Upon successful return, the mutex shall have been locked and shall be owned by the calling
50134 thread. If *mutex* is a robust mutex where an owner terminated while holding the lock and the
50135 state is recoverable, the mutex shall be acquired even though the function returns an error code.

50136 When using condition variables there is always a Boolean predicate involving shared variables
50137 associated with each condition wait that is true if the thread should proceed. Spurious wakeups
50138 from the *pthread_cond_timedwait()* or *pthread_cond_wait()* functions may occur. Since the return
50139 from *pthread_cond_timedwait()* or *pthread_cond_wait()* does not imply anything about the value of
50140 this predicate, the predicate should be re-evaluated upon such return.

50141 When a thread waits on a condition variable, having specified a particular mutex to either the
50142 *pthread_cond_timedwait()* or the *pthread_cond_wait()* operation, a dynamic binding is formed
50143 between that mutex and condition variable that remains in effect as long as at least one thread is
50144 blocked on the condition variable. During this time, the effect of an attempt by any thread to
50145 wait on that condition variable using a different mutex is undefined. Once all waiting threads
50146 have been unblocked (as by the *pthread_cond_broadcast()* operation), the next wait operation on
50147 that condition variable shall form a new dynamic binding with the mutex specified by that wait
50148 operation. Even though the dynamic binding between condition variable and mutex may be
50149 removed or replaced between the time a thread is unblocked from a wait on the condition
50150 variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked
50151 thread shall always re-acquire the mutex specified in the condition wait operation call from
50152 which it is returning.

50153 A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a
50154 thread is set to *PTHREAD_CANCEL_DEFERRED*, a side effect of acting upon a cancellation
50155 request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first
50156 cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up
50157 to the point of returning from the call to *pthread_cond_timedwait()* or *pthread_cond_wait()*, but at
50158 that point notices the cancellation request and instead of returning to the caller of
50159 *pthread_cond_timedwait()* or *pthread_cond_wait()*, starts the thread cancellation activities, which
50160 includes calling cancellation cleanup handlers.

50161 A thread that has been unblocked because it has been canceled while blocked in a call to
 50162 *pthread_cond_timedwait()* or *pthread_cond_wait()* shall not consume any condition signal that may
 50163 be directed concurrently at the condition variable if there are other threads blocked on the
 50164 condition variable.

50165 The *pthread_cond_timedwait()* function shall be equivalent to *pthread_cond_wait()*, except that an
 50166 error is returned if the absolute time specified by *abstime* passes (that is, system time equals or
 50167 exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time
 50168 specified by *abstime* has already been passed at the time of the call.

50169 The condition variable shall have a clock attribute which specifies the clock that shall be used to
 50170 measure the time specified by the *abstime* argument. When such timeouts occur,
 50171 *pthread_cond_timedwait()* shall nonetheless release and re-acquire the mutex referenced by *mutex*.
 50172 The *pthread_cond_timedwait()* function is also a cancellation point.

50173 If a signal is delivered to a thread waiting for a condition variable, upon return from the signal
 50174 handler the thread resumes waiting for the condition variable as if it was not interrupted, or it
 50175 shall return zero due to spurious wakeup.

50176 RETURN VALUE

50177 Except in the case of [ETIMEDOUT], all these error checks shall act as if they were performed
 50178 immediately at the beginning of processing for the function and shall cause an error return, in
 50179 effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable
 50180 specified by *cond*.

50181 Upon successful completion, a value of zero shall be returned; otherwise, an error number shall
 50182 be returned to indicate the error.

50183 ERRORS

50184 The *pthread_cond_timedwait()* function shall fail if:

50185 [ETIMEDOUT] The time specified by *abstime* to *pthread_cond_timedwait()* has passed.

50186 [EINVAL] The *abstime* argument specified a nanosecond value less than zero or greater
 50187 than or equal to 1000 million.

50188 [ENOTRECOVERABLE]
 50189 The state protected by the mutex is not recoverable. The mutex is not locked.

50190 [EOWNERDEAD]
 50191 The mutex is a robust mutex and the process containing the previous owner
 50192 thread terminated while holding the mutex lock. The mutex lock has been
 50193 acquired and it is up to the new owner to make the state consistent.

50194 These functions may fail if:

50195 [EINVAL] The value specified by *cond* or *mutex* is invalid.

50196 [EOWNERDEAD]
 50197 The mutex is a robust mutex and the previous owning thread terminated
 50198 while holding the mutex lock. The mutex lock has been acquired and it is up
 50199 to the new owner to make the state consistent.

50200 [EPERM] The mutex was not owned by the current thread at the time of the call.

50201 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE**Condition Wait Semantics**

It is important to note that when *pthread_cond_wait()* and *pthread_cond_timedwait()* return without error, the associated predicate may still be false. Similarly, when *pthread_cond_timedwait()* returns with the timeout error, the associated predicate may be true due to an unavoidable race between the expiration of the timeout and the predicate state change.

The application needs to recheck the predicate on any return because it cannot be sure there is another thread waiting on the thread to handle the signal, and if there is not then the signal is lost. The burden is on the application to check the predicate.

Some implementations, particularly on a multi-processor, may sometimes cause multiple threads to wake up when the condition variable is signaled simultaneously on different processors.

In general, whenever a condition wait returns, the thread has to re-evaluate the predicate associated with the condition wait to determine whether it can safely proceed, should wait again, or should declare a timeout. A return from the wait does not imply that the associated predicate is either true or false.

It is thus recommended that a condition wait be enclosed in the equivalent of a “while loop” that checks the predicate.

Timed Wait Semantics

An absolute time measure was chosen for specifying the timeout parameter for two reasons. First, a relative time measure can be easily implemented on top of a function that specifies absolute time, but there is a race condition associated with specifying an absolute timeout on top of a function that specifies relative timeouts. For example, assume that *clock_gettime()* returns the current time and *cond_relative_timed_wait()* uses relative timeouts:

```
clock_gettime(CLOCK_REALTIME, &now)
reltime = sleep_til_this_absolute_time -now;
cond_relative_timed_wait(c, m, &reltime);
```

If the thread is preempted between the first statement and the last statement, the thread blocks for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

50245

Cancellation and Condition Wait50246
50247
50248
50249
50250
50251
50252

A condition wait, whether timed or not, is a cancellation point. That is, the functions `pthread_cond_wait()` or `pthread_cond_timedwait()` are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

50253
50254
50255
50256
50257
50258
50259
50260
50261
50262

A side effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

50263
50264
50265

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

50266
50267
50268
50269
50270
50271

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

50272

Performance of Mutexes and Condition Variables50273
50274
50275

Mutexes are expected to be locked only for a few instructions. This practice is almost automatically enforced by the desire of programmers to avoid long serial regions of execution (which would reduce total effective parallelism).

50276
50277
50278
50279
50280
50281
50282
50283

When using mutexes and condition variables, one tries to ensure that the usual case is to lock the mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a relatively rare situation. For example, when implementing a read-write lock, code that acquires a read-lock typically needs only to increment the count of readers (under mutual-exclusion) and return. The calling thread would actually wait on the condition variable only when there is already an active writer. So the efficiency of a synchronization operation is bounded by the cost of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context switch.

50284
50285
50286
50287

This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable is important. The cost of waiting on a condition variable should be little more than the minimal cost for a context switch plus the time to unlock and lock the mutex.

50288

Features of Mutexes and Condition Variables50289
50290
50291
50292
50293
50294
50295
50296

It had been suggested that the mutex acquisition and release be decoupled from condition wait. This was rejected because it is the combined nature of the operation that, in fact, facilitates realtime implementations. Those implementations can atomically move a high-priority thread between the condition variable and the mutex in a manner that is transparent to the caller. This can prevent extra context switches and provide more deterministic acquisition of a mutex when the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the scheduling discipline. Furthermore, the current condition wait operation matches existing practice.

50297

Scheduling Behavior of Mutexes and Condition Variables50298
50299
50300
50301
50302

Synchronization primitives that attempt to interfere with scheduling policy by specifying an ordering rule are considered undesirable. Threads waiting on mutexes and condition variables are selected to proceed in an order dependent upon the scheduling policy rather than in some fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which thread(s) are awakened and allowed to proceed.

50303

Timed Condition Wait50304
50305

The `pthread_cond_timedwait()` function allows an application to give up waiting for a particular condition after a given amount of time. An example of its use follows:

50306
50307
50308
50309
50310
50311
50312
50313
50314
50315

```
(void) pthread_mutex_lock(&t.mn);
    t.waiters++;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += 5;
    rc = 0;
    while (! mypredicate(&t) && rc == 0)
        rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
    t.waiters--;
    if (rc == 0) setmystate(&t);
(void) pthread_mutex_unlock(&t.mn);
```

50316
50317
50318
50319
50320

By making the timeout parameter absolute, it does not need to be recomputed each time the program checks its blocking predicate. If the timeout was relative, it would have to be recomputed before each call. This would be especially difficult since such code would need to take into account the possibility of extra wakeups that result from extra broadcasts or signals on the condition variable that occur before either the predicate is true or the timeout is due.

50321

FUTURE DIRECTIONS

50322

None.

50323

SEE ALSO

50324

[pthread_cond_broadcast\(\)](#)

50325

XBD Section 4.11 (on page 98), [<pthread.h>](#)

50326

CHANGE HISTORY

50327

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50328

Issue 650329
50330

The `pthread_cond_timedwait()` and `pthread_cond_wait()` functions are marked as part of the Threads option.

50331
50332

The Open Group Corrigendum U021/9 is applied, correcting the prototype for the `pthread_cond_wait()` function.

50333

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding semantics

- 50334 for the Clock Selection option.
- 50335 The ERRORS section has an additional case for [EPERM] in response to IEEE PASC
50336 Interpretation 1003.1c #28.
- 50337 The **restrict** keyword is added to the *pthread_cond_timedwait()* and *pthread_cond_wait()*
50338 prototypes for alignment with the ISO/IEC 9899:1999 standard.
- 50339 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/89 is applied, updating the
50340 DESCRIPTION for consistency with the *pthread_cond_destroy()* function that states it is safe to
50341 destroy an initialized condition variable upon which no threads are currently blocked.
- 50342 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/90 is applied, updating words in the
50343 DESCRIPTION from “the cancelability enable state” to “the cancelability type”.
- 50344 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/91 is applied, updating the ERRORS
50345 section to remove the error case related to *abstime* from the *pthread_cond_wait()* function, and to
50346 make the error case related to *abstime* mandatory for *pthread_cond_timedwait()* for consistency
50347 with other functions.
- 50348 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/92 is applied, adding a new paragraph to
50349 the RATIONALE section stating that an application should check the predicate on any return
50350 from this function.
- 50351 **Issue 7**
- 50352 SD5-XSH-ERN-44 is applied, changing the definition of the “shall fail” case of the [EINVAL]
50353 error.
- 50354 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.
- 50355 The *pthread_cond_timedwait()* and *pthread_cond_wait()* functions are moved from the Threads
50356 option to the Base.

50357 **NAME**

50358 pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable
 50359 attributes object

50360 **SYNOPSIS**

50361 #include <pthread.h>

50362 int pthread_condattr_destroy(pthread_condattr_t *attr);
 50363 int pthread_condattr_init(pthread_condattr_t *attr);

50364 **DESCRIPTION**

50365 The *pthread_condattr_destroy()* function shall destroy a condition variable attributes object; the
 50366 object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy()*
 50367 to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
 50368 reinitialized using *pthread_condattr_init()*; the results of otherwise referencing the object after it
 50369 has been destroyed are undefined.

50370 The *pthread_condattr_init()* function shall initialize a condition variable attributes object *attr* with
 50371 the default value for all of the attributes defined by the implementation.

50372 Results are undefined if *pthread_condattr_init()* is called specifying an already initialized *attr*
 50373 attributes object.

50374 After a condition variable attributes object has been used to initialize one or more condition
 50375 variables, any function affecting the attributes object (including destruction) shall not affect any
 50376 previously initialized condition variables.

50377 This volume of POSIX.1-200x requires two attributes, the *clock* attribute and the *process-shared*
 50378 attribute.

50379 Additional attributes, their default values, and the names of the associated functions to get and
 50380 set those attribute values are implementation-defined.

50381 **RETURN VALUE**

50382 If successful, the *pthread_condattr_destroy()* and *pthread_condattr_init()* functions shall return
 50383 zero; otherwise, an error number shall be returned to indicate the error.

50384 **ERRORS**

50385 The *pthread_condattr_destroy()* function may fail if:

50386 [EINVAL] The value specified by *attr* is invalid.

50387 The *pthread_condattr_init()* function shall fail if:

50388 [ENOMEM] Insufficient memory exists to initialize the condition variable attributes object.

50389 These functions shall not return an error code of [EINTR].

50390 **EXAMPLES**

50391 None.

50392 **APPLICATION USAGE**

50393 None.

50394 **RATIONALE**

50395 See *pthread_attr_destroy()* and *pthread_mutex_destroy()*.

50396 A *process-shared* attribute has been defined for condition variables for the same reason it has been
 50397 defined for mutexes.

50398 **FUTURE DIRECTIONS**

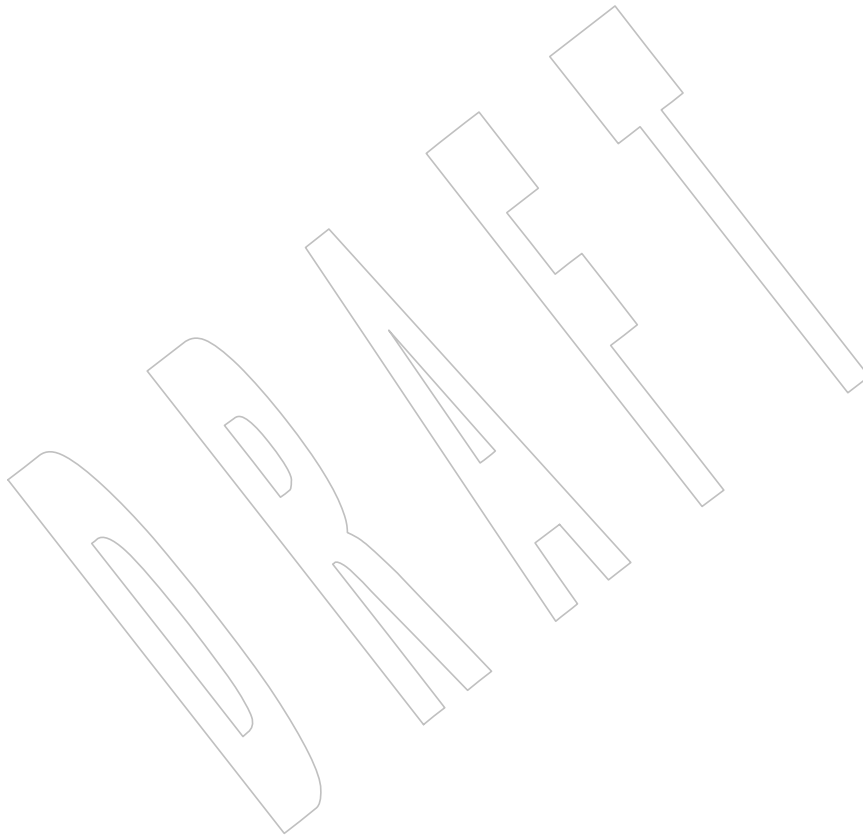
50399 None.

50400 **SEE ALSO**50401 *pthread_attr_destroy()*, *pthread_cond_destroy()*, *pthread_condattr_getpshared()*, *pthread_create()*,
50402 *pthread_mutex_destroy()*

50403 XBD <pthread.h>

50404 **CHANGE HISTORY**

50405 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50406 **Issue 6**50407 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are marked as part of the
50408 Threads option.50409 **Issue 7**50410 The *pthread_condattr_destroy()* and *pthread_condattr_init()* functions are moved from the Threads
50411 option to the Base.

50412 **NAME**

50413 pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection
 50414 condition variable attribute

50415 **SYNOPSIS**

```
50416 #include <pthread.h>

50417 int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
50418 clockid_t *restrict clock_id);
50419 int pthread_condattr_setclock(pthread_condattr_t *attr,
50420 clockid_t clock_id);
```

50421 **DESCRIPTION**

50422 The *pthread_condattr_getclock()* function shall obtain the value of the *clock* attribute from the
 50423 attributes object referenced by *attr*. The *pthread_condattr_setclock()* function shall set the *clock*
 50424 attribute in an initialized attributes object referenced by *attr*. If *pthread_condattr_setclock()* is
 50425 called with a *clock_id* argument that refers to a CPU-time clock, the call shall fail.

50426 The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of
 50427 *pthread_cond_timedwait()*. The default value of the *clock* attribute shall refer to the system clock.

50428 **RETURN VALUE**

50429 If successful, the *pthread_condattr_getclock()* function shall return zero and store the value of the
 50430 clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error
 50431 number shall be returned to indicate the error.

50432 If successful, the *pthread_condattr_setclock()* function shall return zero; otherwise, an error
 50433 number shall be returned to indicate the error.

50434 **ERRORS**

50435 These functions may fail if:

50436 [EINVAL] The value specified by *attr* is invalid.

50437 The *pthread_condattr_setclock()* function may fail if:

50438 [EINVAL] The value specified by *clock_id* does not refer to a known clock, or is a CPU-
 50439 time clock.

50440 These functions shall not return an error code of [EINTR].

50441 **EXAMPLES**

50442 None.

50443 **APPLICATION USAGE**

50444 None.

50445 **RATIONALE**

50446 None.

50447 **FUTURE DIRECTIONS**

50448 None.

50449 **SEE ALSO**

50450 [pthread_cond_destroy\(\)](#), [pthread_cond_timedwait\(\)](#), [pthread_condattr_destroy\(\)](#),
 50451 [pthread_condattr_getpshared\(\)](#) (on page 1558), [pthread_create\(\)](#), [pthread_mutex_destroy\(\)](#) -

50452 XBD [<pthread.h>](#) +

50453

CHANGE HISTORY

50454

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

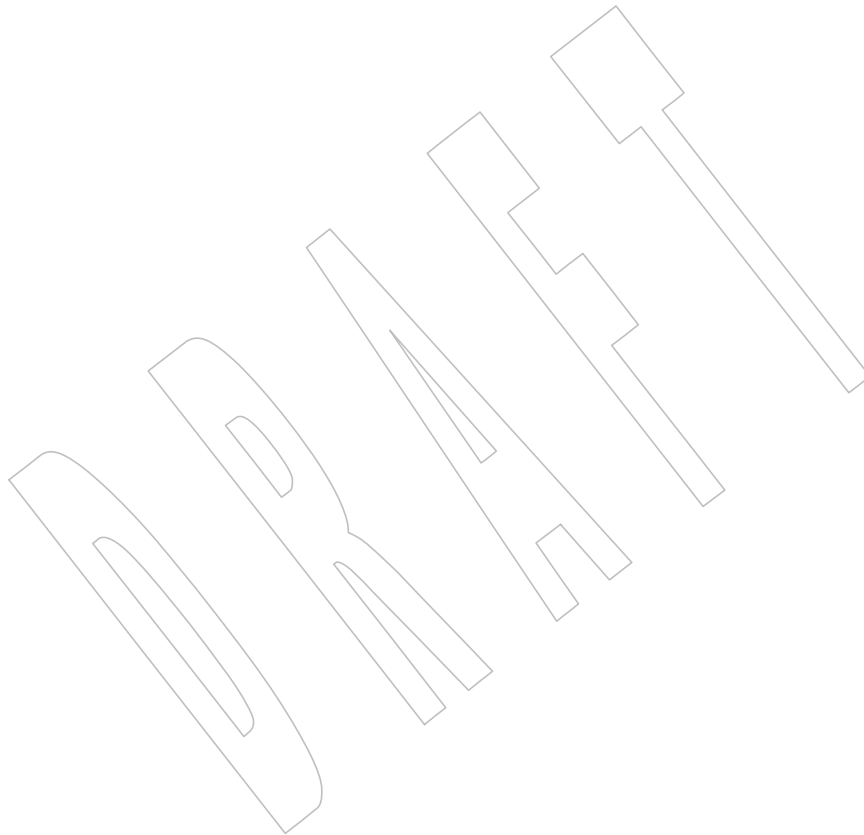
50455

Issue 7

50456

The *pthread_condattr_getclock()* and *pthread_condattr_setclock()* functions are moved from the Clock Selection option to the Base.

50457



50458 **NAME**

50459 pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared
 50460 condition variable attributes

50461 **SYNOPSIS**

```
50462 TSH #include <pthread.h>
50463
50463 int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
50464 int *restrict pshared);
50465 int pthread_condattr_setpshared(pthread_condattr_t *attr,
50466 int pshared);
```

50467 **DESCRIPTION**

50468 The *pthread_condattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 50469 from the attributes object referenced by *attr*. The *pthread_condattr_setpshared()* function shall set
 50470 the *process-shared* attribute in an initialized attributes object referenced by *attr*.

50471 The *process-shared* attribute is set to `PTHREAD_PROCESS_SHARED` to permit a condition
 50472 variable to be operated upon by any thread that has access to the memory where the condition
 50473 variable is allocated, even if the condition variable is allocated in memory that is shared by
 50474 multiple processes. If the *process-shared* attribute is `PTHREAD_PROCESS_PRIVATE`, the
 50475 condition variable shall only be operated upon by threads created within the same process as the
 50476 thread that initialized the condition variable; if threads of differing processes attempt to operate
 50477 on such a condition variable, the behavior is undefined. The default value of the attribute is
 50478 `PTHREAD_PROCESS_PRIVATE`.

50479 **RETURN VALUE**

50480 If successful, the *pthread_condattr_setpshared()* function shall return zero; otherwise, an error
 50481 number shall be returned to indicate the error.

50482 If successful, the *pthread_condattr_getpshared()* function shall return zero and store the value of
 50483 the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise,
 50484 an error number shall be returned to indicate the error.

50485 **ERRORS**

50486 The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions may fail if:

50487 [EINVAL] The value specified by *attr* is invalid.

50488 The *pthread_condattr_setpshared()* function may fail if:

50489 [EINVAL] The new value specified for the attribute is outside the range of legal values
 50490 for that attribute.

50491 These functions shall not return an error code of [EINTR].

50492 **EXAMPLES**

50493 None.

50494 **APPLICATION USAGE**

50495 None.

50496 **RATIONALE**

50497 None.

50498
50499
50500
50501
50502
50503
50504
50505
50506
50507
50508
50509
50510
50511
50512

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create(), *pthread_cond_destroy()*, *pthread_condattr_destroy()*, *pthread_mutex_destroy()*

XBD <pthread.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are marked as part of the Threads and Thread Process-Shared Synchronization options.

The **restrict** keyword is added to the *pthread_condattr_getpshared()* prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

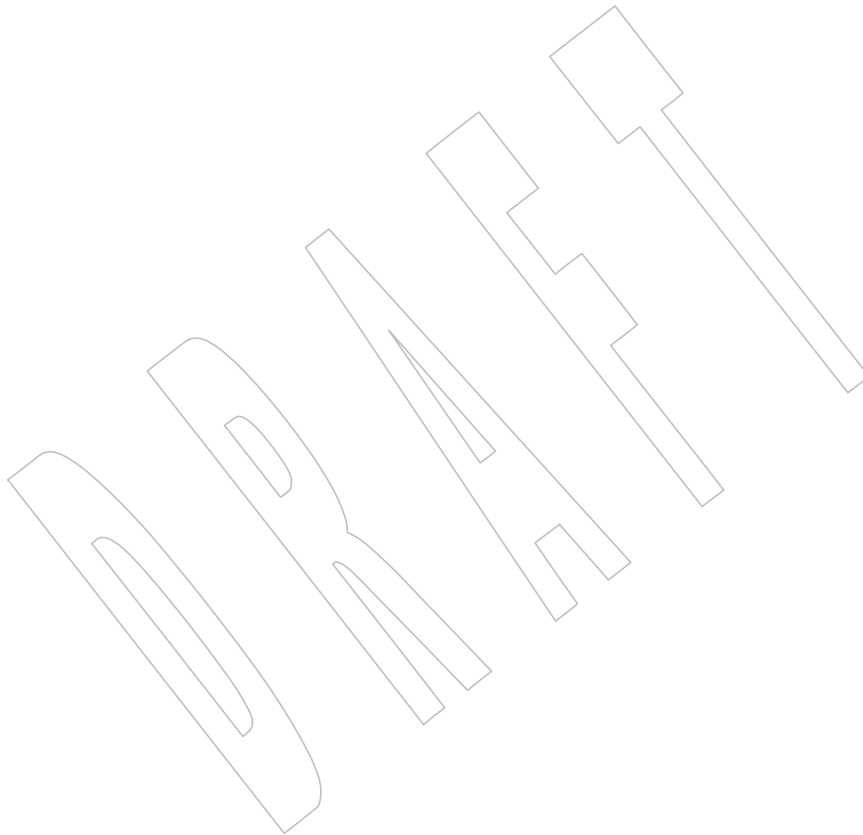
The *pthread_condattr_getpshared()* and *pthread_condattr_setpshared()* functions are moved from the Threads option.

DRAFT

50513 **NAME**
50514 pthread_condattr_init — initialize the condition variable attributes object

50515 **SYNOPSIS**
50516 #include <pthread.h>
50517 int pthread_condattr_init(pthread_condattr_t *attr);

50518 **DESCRIPTION**
50519 Refer to *pthread_condattr_destroy()*.

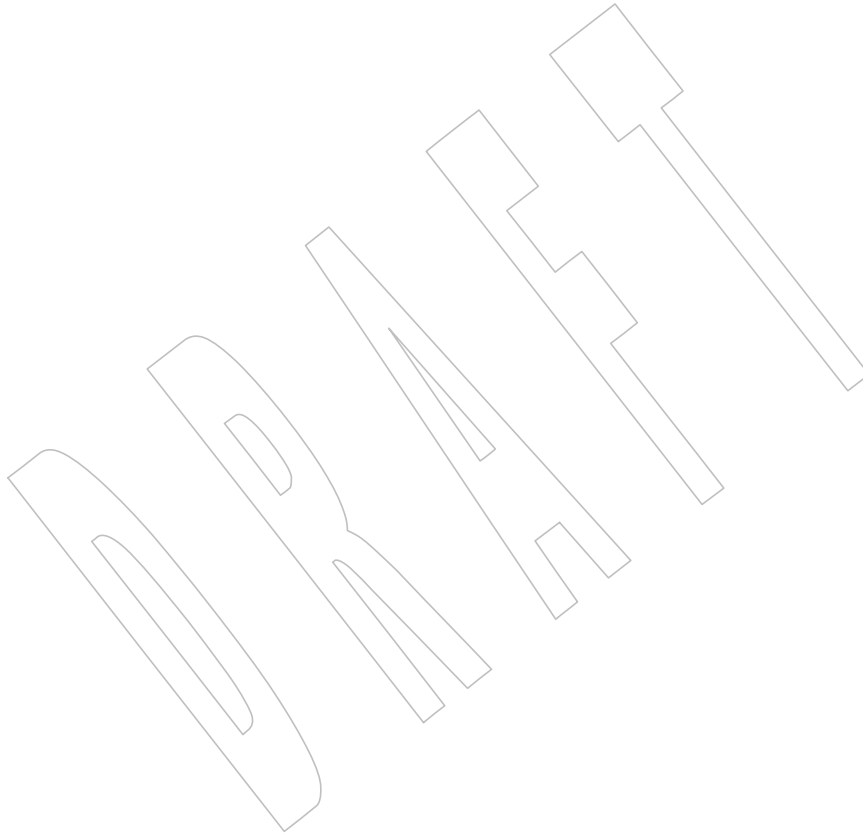


50520 **NAME**
50521 pthread_condattr_setclock — set the clock selection condition variable attribute

50522 **SYNOPSIS**
50523 #include <pthread.h>

50524 int pthread_condattr_setclock(pthread_condattr_t *attr,
50525 clockid_t clock_id);

50526 **DESCRIPTION**
50527 Refer to *pthread_condattr_getclock()*.



50528 **NAME**

50529 pthread_condattr_setpshared — set the process-shared condition variable attribute

50530 **SYNOPSIS**50531 TSH

```
#include <pthread.h>
```

```
int pthread_condattr_setpshared(pthread_condattr_t *attr,
```

```
int pshared);
```

50534 **DESCRIPTION**50535 Refer to *pthread_condattr_getpshared()*.

50536 **NAME**

50537 pthread_create — thread creation

50538 **SYNOPSIS**

50539 #include <pthread.h>

```
50540 int pthread_create(pthread_t *restrict thread,
50541                  const pthread_attr_t *restrict attr,
50542                  void *(*start_routine)(void*), void *restrict arg);
```

50543 **DESCRIPTION**

50544 The *pthread_create()* function shall create a new thread, with attributes specified by *attr*, within a
 50545 process. If *attr* is *NULL*, the default attributes shall be used. If the attributes specified by *attr* are
 50546 modified later, the thread's attributes shall not be affected. Upon successful completion,
 50547 *pthread_create()* shall store the ID of the created thread in the location referenced by *thread*.

50548 The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine*
 50549 returns, the effect shall be as if there was an implicit call to *pthread_exit()* using the return value
 50550 of *start_routine* as the exit status. Note that the thread in which *main()* was originally invoked
 50551 differs from this. When it returns from *main()*, the effect shall be as if there was an implicit call to
 50552 *exit()* using the return value of *main()* as the exit status.

50553 The signal state of the new thread shall be initialized as follows:

- 50554 • The signal mask shall be inherited from the creating thread.
- 50555 • The set of signals pending for the new thread shall be empty.

50556 XSI The alternate stack shall not be inherited.

50557 The floating-point environment shall be inherited from the creating thread.

50558 If *pthread_create()* fails, no new thread is created and the contents of the location referenced by
 50559 *thread* are undefined.

50560 TCT If *_POSIX_THREAD_CPUTIME* is defined, the new thread shall have a CPU-time clock
 50561 accessible, and the initial value of this clock shall be set to zero.

50562 **RETURN VALUE**

50563 If successful, the *pthread_create()* function shall return zero; otherwise, an error number shall be
 50564 returned to indicate the error.

50565 **ERRORS**

50566 The *pthread_create()* function shall fail if:

50567 [EAGAIN] The system lacked the necessary resources to create another thread, or the
 50568 system-imposed limit on the total number of threads in a process
 50569 {*PTHREAD_THREADS_MAX*} would be exceeded.

50570 [EPERM] The caller does not have appropriate permission to set the required scheduling
 50571 parameters or scheduling policy.

50572 The *pthread_create()* function may fail if:

50573 [EINVAL] The attributes specified by *attr* are invalid.

50574 The *pthread_create()* function shall not return an error code of [EINTR].

EXAMPLES

50575
50576 None.

APPLICATION USAGE

50577
50578 There is no requirement on the implementation that the ID of the created thread be available
50579 before the newly created thread starts executing. The calling thread can obtain the ID of the
50580 created thread through the return value of the *pthread_create()* function, and the newly created
50581 thread can obtain its ID by a call to *pthread_self()*.

RATIONALE

50582
50583 A suggested alternative to *pthread_create()* would be to define two separate operations: create
50584 and start. Some applications would find such behavior more natural. Ada, in particular,
50585 separates the “creation” of a task from its “activation”.

50586 Splitting the operation was rejected by the standard developers for many reasons:

- 50587 • The number of calls required to start a thread would increase from one to two and thus
50588 place an additional burden on applications that do not require the additional
50589 synchronization. The second call, however, could be avoided by the additional
50590 complication of a start-up state attribute.
- 50591 • An extra state would be introduced: “created but not started”. This would require the
50592 standard to specify the behavior of the thread operations when the target has not yet
50593 started executing.
- 50594 • For those applications that require such behavior, it is possible to simulate the two separate
50595 steps with the facilities that are currently provided. The *start_routine()* can synchronize by
50596 waiting on a condition variable that is signaled by the start operation.

50597 An Ada implementor can choose to create the thread at either of two points in the Ada program:
50598 when the task object is created, or when the task is activated (generally at a “begin”). If the first
50599 approach is adopted, the *start_routine()* needs to wait on a condition variable to receive the order
50600 to begin “activation”. The second approach requires no such condition variable or extra
50601 synchronization. In either approach, a separate Ada task control block would need to be created
50602 when the task object is created to hold rendezvous queues, and so on.

50603 An extension of the preceding model would be to allow the state of the thread to be modified
50604 between the create and start. This would allow the thread attributes object to be eliminated. This
50605 has been rejected because:

- 50606 • All state in the thread attributes object has to be able to be set for the thread. This would
50607 require the definition of functions to modify thread attributes. There would be no
50608 reduction in the number of function calls required to set up the thread. In fact, for an
50609 application that creates all threads using identical attributes, the number of function calls
50610 required to set up the threads would be dramatically increased. Use of a thread attributes
50611 object permits the application to make one set of attribute setting function calls.
50612 Otherwise, the set of attribute setting function calls needs to be made for each thread
50613 creation.
- 50614 • Depending on the implementation architecture, functions to set thread state would require
50615 kernel calls, or for other implementation reasons would not be able to be implemented as
50616 macros, thereby increasing the cost of thread creation.
- 50617 • The ability for applications to segregate threads by class would be lost.

50618 Another suggested alternative uses a model similar to that for process creation, such as “thread
50619 fork”. The fork semantics would provide more flexibility and the “create” function can be
50620 implemented simply by doing a thread fork followed immediately by a call to the desired “start
50621 routine” for the thread. This alternative has these problems:

- 50622 • For many implementations, the entire stack of the calling thread would need to be
- 50623 duplicated, since in many architectures there is no way to determine the size of the calling
- 50624 frame.
- 50625 • Efficiency is reduced since at least some part of the stack has to be copied, even though in
- 50626 most cases the thread never needs the copied context, since it merely calls the desired start
- 50627 routine.

FUTURE DIRECTIONS

50628 None.

SEE ALSO

50630 *fork()*, *pthread_exit()*, *pthread_join()*

50631 XBD Section 4.11 (on page 98), [<pthread.h>](#)

CHANGE HISTORY

50633 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

50635 The *pthread_create()* function is marked as part of the Threads option.

50637 The following new requirements on POSIX implementations derive from alignment with the

50638 Single UNIX Specification:

- 50639 • The [EPERM] mandatory error condition is added.

50640 The thread CPU-time clock semantics are added for alignment with IEEE Std 1003.1d-1999.

50641 The **restrict** keyword is added to the *pthread_create()* prototype for alignment with the

50642 ISO/IEC 9899:1999 standard.

50643 The DESCRIPTION is updated to make it explicit that the floating-point environment is

50644 inherited from the creating thread.

50645 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/44 is applied, adding text that the

50646 alternate stack is not inherited.

50647 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/93 is applied, updating the ERRORS

50648 section to remove the mandatory [EINVAL] error (“The value specified by *attr* is invalid”), and

50649 adding the optional [EINVAL] error (“The attributes specified by *attr* are invalid”).

50650 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/94 is applied, adding the APPLICATION

50651 USAGE section.

Issue 7

50652 The *pthread_create()* function is moved from the Threads option to the Base.

50653

50654 **NAME**

50655 pthread_detach — detach a thread

50656 **SYNOPSIS**

50657 #include <pthread.h>

50658 int pthread_detach(pthread_t thread);

50659 **DESCRIPTION**

50660 The *pthread_detach()* function shall indicate to the implementation that storage for the thread
 50661 *thread* can be reclaimed when that thread terminates. If *thread* has not terminated,
 50662 *pthread_detach()* shall not cause it to terminate. The effect of multiple *pthread_detach()* calls on the
 50663 same target thread is unspecified.

50664 **RETURN VALUE**

50665 If the call succeeds, *pthread_detach()* shall return 0; otherwise, an error number shall be returned
 50666 to indicate the error.

50667 **ERRORS**50668 The *pthread_detach()* function may fail if:

- 50669 [EINVAL] The implementation has detected that the value specified by *thread* does not
 50670 refer to a joinable thread.
- 50671 [ESRCH] No thread could be found corresponding to that specified by the given thread
 50672 ID.

50673 The *pthread_detach()* function shall not return an error code of [EINTR].50674 **EXAMPLES**

50675 None.

50676 **APPLICATION USAGE**

50677 None.

50678 **RATIONALE**

50679 The *pthread_join()* or *pthread_detach()* functions should eventually be called for every thread that
 50680 is created so that storage associated with the thread may be reclaimed.

50681 It has been suggested that a “detach” function is not necessary; the *detachstate* thread creation
 50682 attribute is sufficient, since a thread need never be dynamically detached. However, need arises
 50683 in at least two cases:

- 50684 1. In a cancellation handler for a *pthread_join()* it is nearly essential to have a
 50685 *pthread_detach()* function in order to detach the thread on which *pthread_join()* was
 50686 waiting. Without it, it would be necessary to have the handler do another *pthread_join()*
 50687 to attempt to detach the thread, which would both delay the cancellation processing for an
 50688 unbounded period and introduce a new call to *pthread_join()*, which might itself need a
 50689 cancellation handler. A dynamic detach is nearly essential in this case.
- 50690 2. In order to detach the “initial thread” (as may be desirable in processes that set up server
 50691 threads).

50692 **FUTURE DIRECTIONS**

50693 None.

50694

SEE ALSO

50695

pthread_join()

50696

XBD <pthread.h>

50697

CHANGE HISTORY

50698

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50699

Issue 6

50700

The *pthread_detach()* function is marked as part of the Threads option.

50701

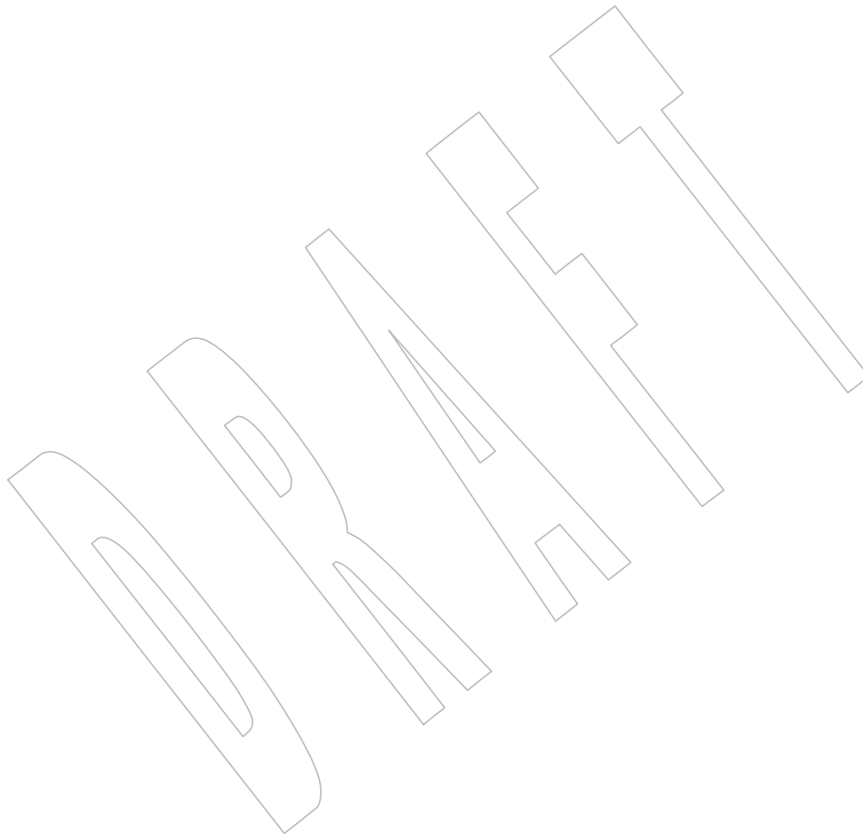
IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/95 is applied, updating the ERRORS section so that the [EINVAL] and [ESRCH] error cases become optional.

50702

50703

Issue 7

50704

The *pthread_detach()* function is moved from the Threads option to the Base.

50705 **NAME**

50706 pthread_equal — compare thread IDs

50707 **SYNOPSIS**

50708 #include <pthread.h>

50709 int pthread_equal(pthread_t t1, pthread_t t2);

50710 **DESCRIPTION**50711 This function shall compare the thread IDs *t1* and *t2*.50712 **RETURN VALUE**50713 The *pthread_equal()* function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero
50714 shall be returned.50715 If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.50716 **ERRORS**

50717 No errors are defined.

50718 The *pthread_equal()* function shall not return an error code of [EINTR].50719 **EXAMPLES**

50720 None.

50721 **APPLICATION USAGE**

50722 None.

50723 **RATIONALE**50724 Implementations may choose to define a thread ID as a structure. This allows additional
50725 flexibility and robustness over using an **int**. For example, a thread ID could include a sequence
50726 number that allows detection of “dangling IDs” (copies of a thread ID that has been detached).
50727 Since the C language does not support comparison on structure types, the *pthread_equal()*
50728 function is provided to compare thread IDs.50729 **FUTURE DIRECTIONS**

50730 None.

50731 **SEE ALSO**50732 *pthread_create()*, *pthread_self()*

50733 XBD <pthread.h>

50734 **CHANGE HISTORY**

50735 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50736 **Issue 6**50737 The *pthread_equal()* function is marked as part of the Threads option.50738 **Issue 7**50739 The *pthread_equal()* function is moved from the Threads option to the Base.

50740 **NAME**

50741 pthread_exit — thread termination

50742 **SYNOPSIS**

50743 #include <pthread.h>

50744 void pthread_exit(void *value_ptr);

50745 **DESCRIPTION**

50746 The *pthread_exit()* function shall terminate the calling thread and make the value *value_ptr*
 50747 available to any successful join with the terminating thread. Any cancellation cleanup handlers
 50748 that have been pushed and not yet popped shall be popped in the reverse order that they were
 50749 pushed and then executed. After all cancellation cleanup handlers have been executed, if the
 50750 thread has any thread-specific data, appropriate destructor functions shall be called in an
 50751 unspecified order. Thread termination does not release any application visible process resources,
 50752 including, but not limited to, mutexes and file descriptors, nor does it perform any process-level
 50753 cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

50754 An implicit call to *pthread_exit()* is made when a thread other than the thread in which *main()*
 50755 was first invoked returns from the start routine that was used to create it. The function's return
 50756 value shall serve as the thread's exit status.

50757 The behavior of *pthread_exit()* is undefined if called from a cancellation cleanup handler or
 50758 destructor function that was invoked as a result of either an implicit or explicit call to
 50759 *pthread_exit()*.

50760 After a thread has terminated, the result of access to local (auto) variables of the thread is
 50761 undefined. Thus, references to local variables of the exiting thread should not be used for the
 50762 *pthread_exit()* *value_ptr* parameter value.

50763 The process shall exit with an exit status of 0 after the last thread has been terminated. The
 50764 behavior shall be as if the implementation called *exit()* with a zero argument at thread
 50765 termination time.

50766 **RETURN VALUE**50767 The *pthread_exit()* function cannot return to its caller.50768 **ERRORS**

50769 No errors are defined.

50770 **EXAMPLES**

50771 None.

50772 **APPLICATION USAGE**

50773 None.

50774 **RATIONALE**

50775 The normal mechanism by which a thread terminates is to return from the routine that was
 50776 specified in the *pthread_create()* call that started it. The *pthread_exit()* function provides the
 50777 capability for a thread to terminate without requiring a return from the start routine of that
 50778 thread, thereby providing a function analogous to *exit()*.

50779 Regardless of the method of thread termination, any cancellation cleanup handlers that have
 50780 been pushed and not yet popped are executed, and the destructors for any existing thread-
 50781 specific data are executed. This volume of POSIX.1-200x requires that cancellation cleanup
 50782 handlers be popped and called in order. After all cancellation cleanup handlers have been
 50783 executed, thread-specific data destructors are called, in an unspecified order, for each item of
 50784 thread-specific data that exists in the thread. This ordering is necessary because cancellation

50785 cleanup handlers may rely on thread-specific data.

50786 As the meaning of the status is determined by the application (except when the thread has been
50787 canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an
50788 illegal status value is, which is why no address error checking is done.

FUTURE DIRECTIONS

50789 None.

SEE ALSO

50791 *exit()*, *pthread_create()*, *pthread_join()*

50792 XBD <pthread.h>

CHANGE HISTORY

50794 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

50796 The *pthread_exit()* function is marked as part of the Threads option.

Issue 7

50798 The *pthread_exit()* function is moved from the Threads option to the Base.
50799

DRAFT

50800 **NAME**

50801 pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

50802 **SYNOPSIS**

```
50803 OB XSI #include <pthread.h>
50804 int pthread_getconcurrency(void);
50805 int pthread_setconcurrency(int new_level);
```

50806 **DESCRIPTION**

50807 Unbound threads in a process may or may not be required to be simultaneously active. By
 50808 default, the threads implementation ensures that a sufficient number of threads are active so that
 50809 the process can continue to make progress. While this conserves system resources, it may not
 50810 produce the most effective level of concurrency.

50811 The *pthread_setconcurrency()* function allows an application to inform the threads
 50812 implementation of its desired concurrency level, *new_level*. The actual level of concurrency
 50813 provided by the implementation as a result of this function call is unspecified.

50814 If *new_level* is zero, it causes the implementation to maintain the concurrency level at its
 50815 discretion as if *pthread_setconcurrency()* had never been called.

50816 The *pthread_getconcurrency()* function shall return the value set by a previous call to the
 50817 *pthread_setconcurrency()* function. If the *pthread_setconcurrency()* function was not previously
 50818 called, this function shall return zero to indicate that the implementation is maintaining the
 50819 concurrency level.

50820 A call to *pthread_setconcurrency()* shall inform the implementation of its desired concurrency
 50821 level. The implementation shall use this as a hint, not a requirement.

50822 If an implementation does not support multiplexing of user threads on top of several kernel-
 50823 scheduled entities, the *pthread_setconcurrency()* and *pthread_getconcurrency()* functions are
 50824 provided for source code compatibility but they shall have no effect when called. To maintain
 50825 the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency()* is called
 50826 so that a subsequent call to *pthread_getconcurrency()* shall return the same value.

50827 **RETURN VALUE**

50828 If successful, the *pthread_setconcurrency()* function shall return zero; otherwise, an error number
 50829 shall be returned to indicate the error.

50830 The *pthread_getconcurrency()* function shall always return the concurrency level set by a previous
 50831 call to *pthread_setconcurrency()*. If the *pthread_setconcurrency()* function has never been called,
 50832 *pthread_getconcurrency()* shall return zero.

50833 **ERRORS**

50834 The *pthread_setconcurrency()* function shall fail if:

- | | | |
|-------|----------|---|
| 50835 | [EINVAL] | The value specified by <i>new_level</i> is negative. |
| 50836 | [EAGAIN] | The value specified by <i>new_level</i> would cause a system resource to be exceeded. |

50838 The *pthread_setconcurrency()* function shall not return an error code of [EINTR].

pthread_getconcurrency()*System Interfaces*50839
50840
50841
50842
50843
50844
50845
50846
50847
50848
50849
50850
50851
50852
50853
50854
50855**EXAMPLES**

None.

APPLICATION USAGE

Application developers should note that an implementation can always ignore any calls to *pthread_setconcurrency()* and return a constant for *pthread_getconcurrency()*. For this reason, it is not recommended that portable applications use this function.

RATIONALE

None.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSOXBD <[pthread.h](#)>**CHANGE HISTORY**

First released in Issue 5.

Issue 7

SD5-XSH-ERN-184 is applied.

The *pthread_getconcurrency()* and *pthread_setconcurrency()* functions are marked obsolescent.

50856 **NAME**

50857 pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME**
50858 **THREADS**)

50859 **SYNOPSIS**

```
50860 TCT #include <pthread.h>
50861 #include <time.h>
50862 int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

50863 **DESCRIPTION**

50864 The *pthread_getcpuclockid()* function shall return in *clock_id* the clock ID of the CPU-time clock of
50865 the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

50866 **RETURN VALUE**

50867 Upon successful completion, *pthread_getcpuclockid()* shall return zero; otherwise, an error
50868 number shall be returned to indicate the error.

50869 **ERRORS**

50870 The *pthread_getcpuclockid()* function may fail if:

50871 [ESRCH] The value specified by *thread_id* does not refer to an existing thread.

50872 **EXAMPLES**

50873 None.

50874 **APPLICATION USAGE**

50875 The *pthread_getcpuclockid()* function is part of the Thread CPU-Time Clocks option and need not
50876 be provided on all implementations.

50877 **RATIONALE**

50878 None.

50879 **FUTURE DIRECTIONS**

50880 None.

50881 **SEE ALSO**

50882 *clock_getcpuclockid()*, *clock_getres()*, *timer_create()*

50883 XBD [<pthread.h>](#), [<time.h>](#)

50884 **CHANGE HISTORY**

50885 First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

50886 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.

50887 **Issue 7**

50888 The *pthread_getcpuclockid()* function is moved from the Threads option.

pthread_getschedparam()

System Interfaces

50889 NAME

50890 pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters
 50891 access (**REALTIME THREADS**)

50892 SYNOPSIS

```
50893 TPS #include <pthread.h>
50894
50894 int pthread_getschedparam(pthread_t thread, int *restrict policy,
50895 struct sched_param *restrict param);
50896 int pthread_setschedparam(pthread_t thread, int policy,
50897 const struct sched_param *param);
```

50898 DESCRIPTION

50899 The *pthread_getschedparam()* and *pthread_setschedparam()* functions shall, respectively, get and set
 50900 the scheduling policy and parameters of individual threads within a multi-threaded process to
 50901 be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the
 50902 **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected
 50903 scheduling parameters are implementation-defined.

50904 The *pthread_getschedparam()* function shall retrieve the scheduling policy and scheduling
 50905 parameters for the thread whose thread ID is given by *thread* and shall store those values in
 50906 *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam()* shall be
 50907 the value specified by the most recent *pthread_setschedparam()*, *pthread_setschedprio()*, or
 50908 *pthread_create()* call affecting the target thread. It shall not reflect any temporary adjustments to
 50909 its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam()*
 50910 function shall set the scheduling policy and associated scheduling parameters for the thread
 50911 whose thread ID is given by *thread* to the policy and associated parameters provided in *policy*
 50912 and *param*, respectively.

50913 The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The
 50914 scheduling parameters for the SCHED_OTHER policy are implementation-defined. The
 50915 SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

50916 TSP If **_POSIX_THREAD_SPORADIC_SERVER** is defined, then the *policy* argument may have the
 50917 value SCHED_SPORADIC, with the exception for the *pthread_setschedparam()* function that if the
 50918 scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-
 50919 defined whether the function is supported; in other words, the implementation need not allow
 50920 the application to dynamically change the scheduling policy to SCHED_SPORADIC. The
 50921 sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*,
 50922 *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified
 50923 *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the
 50924 function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall
 50925 be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function
 50926 shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are
 50927 stored as provided by this function or are rounded to align with the resolution of the clock being
 50928 used. +

50929 If the *pthread_setschedparam()* function fails, the scheduling parameters shall not be changed for
 50930 the target thread.

50931 RETURN VALUE

50932 If successful, the *pthread_getschedparam()* and *pthread_setschedparam()* functions shall return zero;
 50933 otherwise, an error number shall be returned to indicate the error.

ERRORS

- 50934
- 50935 The *pthread_getschedparam()* function may fail if:
- 50936 [ESRCH] The value specified by *thread* does not refer to an existing thread.
- 50937 The *pthread_setschedparam()* function may fail if:
- 50938 [EINVAL] The value specified by *policy* or one of the scheduling parameters associated with the scheduling policy *policy* is invalid.
- 50939
- 50940 [ENOTSUP] An attempt was made to set the policy or scheduling parameters to an unsupported value.
- 50941
- 50942 TSP [ENOTSUP] An attempt was made to dynamically change the scheduling policy to SCHED_SPORADIC, and the implementation does not support this change.
- 50943
- 50944 [EPERM] The caller does not have the appropriate permission to set either the scheduling parameters or the scheduling policy of the specified thread.
- 50945
- 50946 [EPERM] The implementation does not allow the application to modify one of the parameters to the value specified.
- 50947
- 50948 [ESRCH] The value specified by *thread* does not refer to a existing thread.
- 50949 These functions shall not return an error code of [EINTR].

EXAMPLES

50950 None.

50951

APPLICATION USAGE

50952 None.

50953

RATIONALE

50954 None.

50955

FUTURE DIRECTIONS

50956 None.

50957

SEE ALSO

50958 *pthread_setschedprio()*, *sched_getparam()*, *sched_getscheduler()*

50959

50960 XBD <pthread.h>, <sched.h>

CHANGE HISTORY

50961 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

50962

Issue 6

50963 The *pthread_getschedparam()* and *pthread_setschedparam()* functions are marked as part of the Threads and Thread Execution Scheduling options.

50964

50965

50966 The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Thread Execution Scheduling option.

50967

50968 The Open Group Corrigendum U026/2 is applied, correcting the prototype for the *pthread_setschedparam()* function so that its second argument is of type **int**.

50969

50970 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

50971 The **restrict** keyword is added to the *pthread_getschedparam()* prototype for alignment with the ISO/IEC 9899:1999 standard.

50972

50973 The Open Group Corrigendum U047/1 is applied.

50974 IEEE PASC Interpretation 1003.1 #96 is applied, noting that priority values can also be set by a call to the *pthread_setschedprio()* function.

50975

pthread_getschedparam()

50976

Issue 7

50977

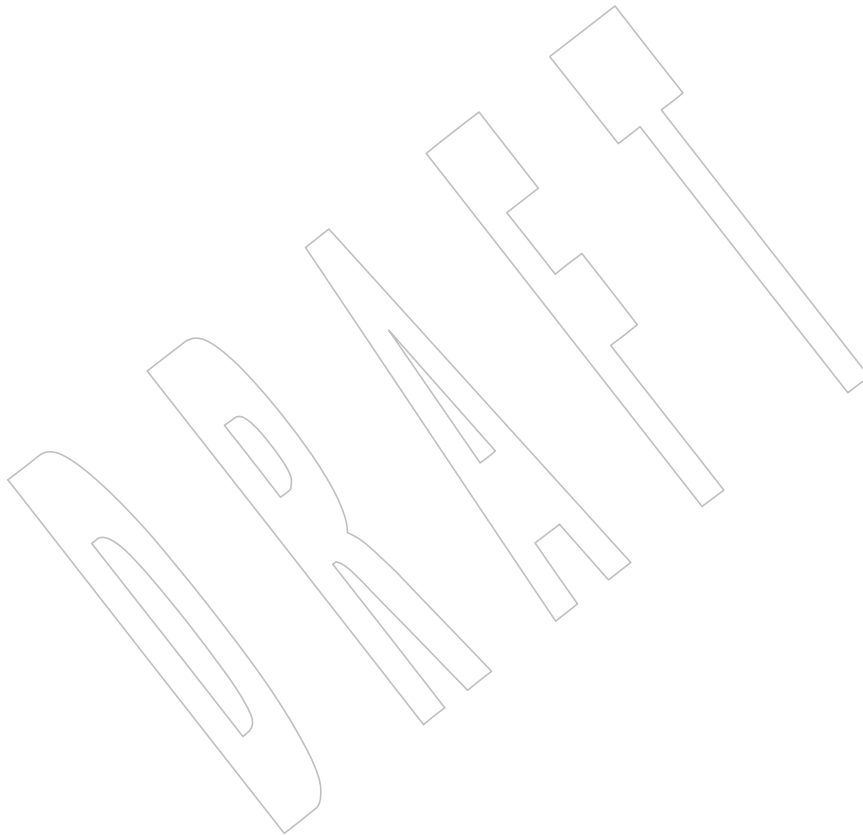
The *pthread_getschedparam()* and *pthread_setschedparam()* functions are moved from the Threads option.

50978

50979

Austin Group Interpretation 1003.1-2001 #119 is applied.

+



50980 **NAME**

50981 pthread_getspecific, pthread_setspecific — thread-specific data management

50982 **SYNOPSIS**

50983 #include <pthread.h>

50984 void *pthread_getspecific(pthread_key_t key);

50985 int pthread_setspecific(pthread_key_t key, const void *value);

50986 **DESCRIPTION**50987 The *pthread_getspecific()* function shall return the value currently bound to the specified *key* on
50988 behalf of the calling thread.50989 The *pthread_setspecific()* function shall associate a thread-specific *value* with a *key* obtained via a
50990 previous call to *pthread_key_create()*. Different threads may bind different values to the same
50991 key. These values are typically pointers to blocks of dynamically allocated memory that have
50992 been reserved for use by the calling thread.50993 The effect of calling *pthread_getspecific()* or *pthread_setspecific()* with a *key* value not obtained
50994 from *pthread_key_create()* or after *key* has been deleted with *pthread_key_delete()* is undefined.50995 Both *pthread_getspecific()* and *pthread_setspecific()* may be called from a thread-specific data
50996 destructor function. A call to *pthread_getspecific()* for the thread-specific data key being
50997 destroyed shall return the value NULL, unless the value is changed (after the destructor starts)
50998 by a call to *pthread_setspecific()*. Calling *pthread_setspecific()* from a thread-specific data
50999 destructor routine may result either in lost storage (after at least
51000 PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

51001 Both functions may be implemented as macros.

51002 **RETURN VALUE**51003 The *pthread_getspecific()* function shall return the thread-specific data value associated with the
51004 given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be
51005 returned.51006 If successful, the *pthread_setspecific()* function shall return zero; otherwise, an error number shall
51007 be returned to indicate the error.51008 **ERRORS**51009 No errors are returned from *pthread_getspecific()*.51010 The *pthread_setspecific()* function shall fail if:

51011 [ENOMEM] Insufficient memory exists to associate the non-NULL value with the key.

51012 The *pthread_setspecific()* function may fail if:

51013 [EINVAL] The key value is invalid.

51014 The *pthread_setspecific()* function shall not return an error code of [EINTR].

EXAMPLES

51015
51016 None.

APPLICATION USAGE

51017
51018 None.

RATIONALE

51019
51020 Performance and ease-of-use of *pthread_getspecific()* are critical for functions that rely on
51021 maintaining state in thread-specific data. Since no errors are required to be detected by it, and
51022 since the only error that could be detected is the use of an invalid key, the function to
51023 *pthread_getspecific()* has been designed to favor speed and simplicity over error reporting.

FUTURE DIRECTIONS

51024
51025 None.

SEE ALSO

51026
51027 *pthread_key_create()*

51028 XBD <pthread.h>

CHANGE HISTORY

51029
51030 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

51031
51032 The *pthread_getspecific()* and *pthread_setspecific()* functions are marked as part of the Threads
51033 option.

51034 IEEE PASC Interpretation 1003.1c #3 (Part 6) is applied, updating the DESCRIPTION.

51035 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/96 is applied, updating the ERRORS
51036 section so that the [ENOMEM] error case is changed from “to associate the value with the key”
51037 to “to associate the non-NULL value with the key”.

Issue 7

51038
51039 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the ERRORS section.

51040 The *pthread_getspecific()* and *pthread_setspecific()* functions are moved from the Threads option to
51041 the Base.

51042 **NAME**

51043 pthread_join — wait for thread termination

51044 **SYNOPSIS**

51045 #include <pthread.h>

51046 int pthread_join(pthread_t thread, void **value_ptr);

51047 **DESCRIPTION**

51048 The *pthread_join()* function shall suspend execution of the calling thread until the target *thread*
 51049 terminates, unless the target *thread* has already terminated. On return from a successful
 51050 *pthread_join()* call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit()* by
 51051 the terminating thread shall be made available in the location referenced by *value_ptr*. When a
 51052 *pthread_join()* returns successfully, the target thread has been terminated. The results of multiple
 51053 simultaneous calls to *pthread_join()* specifying the same target thread are undefined. If the
 51054 thread calling *pthread_join()* is canceled, then the target thread shall not be detached.

51055 It is unspecified whether a thread that has exited but remains unjoined counts against
 51056 {PTHREAD_THREADS_MAX}.

51057 **RETURN VALUE**

51058 If successful, the *pthread_join()* function shall return zero; otherwise, an error number shall be
 51059 returned to indicate the error.

51060 **ERRORS**51061 The *pthread_join()* function shall fail if:

51062 [ESRCH] No thread could be found corresponding to that specified by the given thread
 51063 ID.

51064 The *pthread_join()* function may fail if:

51065 [EDEADLK] A deadlock was detected or the value of *thread* specifies the calling thread.

51066 [EINVAL] The value specified by *thread* does not refer to a joinable thread.

51067 The *pthread_join()* function shall not return an error code of [EINTR].

51068 **EXAMPLES**

51069 An example of thread creation and deletion follows:

```

51070 typedef struct {
51071     int *ar;
51072     long n;
51073 } subarray;

51074 void *
51075 incer(void *arg)
51076 {
51077     long i;

51078     for (i = 0; i < ((subarray *)arg)->n; i++)
51079         ((subarray *)arg)->ar[i]++;
51080 }

51081 int main(void)
51082 {
51083     int ar[1000000];
51084     pthread_t th1, th2;
51085     subarray sb1, sb2;

```

pthread_join()

```

51086         sb1.ar = &ar[0];
51087         sb1.n  = 500000;
51088         (void) pthread_create(&th1, NULL, incer, &sb1);
51089
51090         sb2.ar = &ar[500000];
51091         sb2.n  = 500000;
51092         (void) pthread_create(&th2, NULL, incer, &sb2);
51093
51094         (void) pthread_join(th1, NULL);
51095         (void) pthread_join(th2, NULL);
51096         return 0;
51097     }

```

APPLICATION USAGE

None.

RATIONALE

The *pthread_join()* function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start_routine()*. The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread_join()* function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread_join()* in this volume of POSIX.1-200x was considered valuable.

The *pthread_join()* function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread_join()* returns, any application-provided stack storage could be reclaimed.

The *pthread_join()* or *pthread_detach()* function should eventually be called for every thread that is created with the *detachstate* attribute set to `PTHREAD_CREATE_JOINABLE` so that storage associated with the thread may be reclaimed.

The interaction between *pthread_join()* and cancellation is well-defined for the following reasons:

- The *pthread_join()* function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
- Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread_join()* call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread_join()* returns. There are no race conditions since *pthread_join()* was called in the deferred cancelability state.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create(), *wait*

XBD Section 4.11 (on page 98), `<pthread.h>`

51129

CHANGE HISTORY

51130

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51131

Issue 6

51132

The *pthread_join()* function is marked as part of the Threads option.

51133

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/97 is applied, updating the ERRORS section so that the [EINVAL] error is made optional and the words “the implementation has detected” are removed from it.

51134

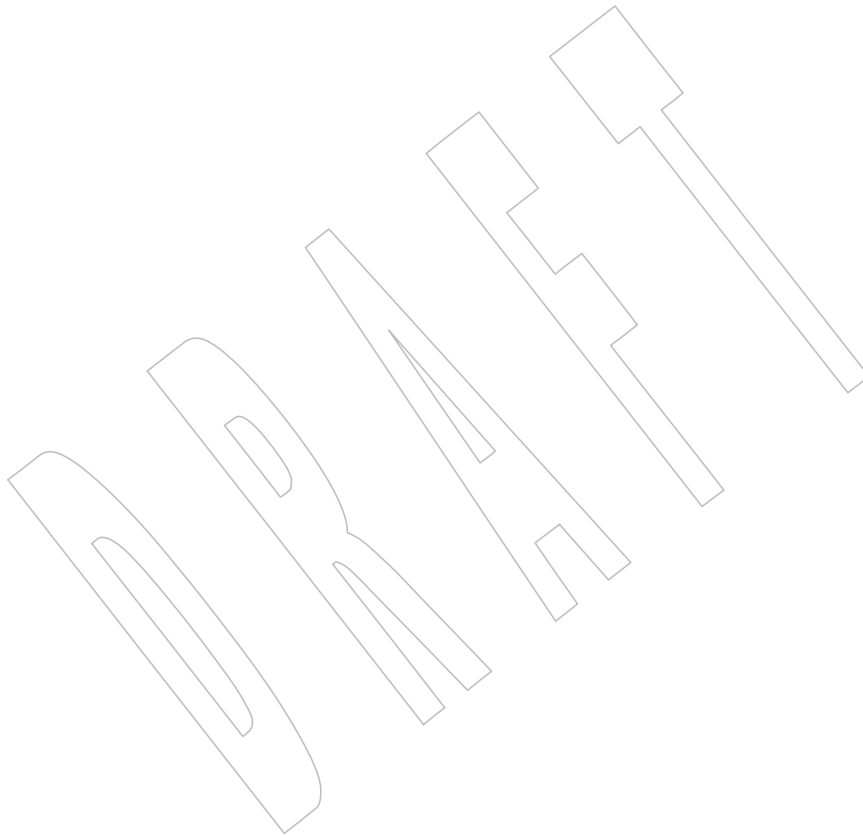
51135

51136

Issue 7

51137

The *pthread_join()* function is moved from the Threads option to the Base.



51138 **NAME**

51139 pthread_key_create — thread-specific data key creation

51140 **SYNOPSIS**

51141 #include <pthread.h>

51142 int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));

51143 **DESCRIPTION**

51144 The *pthread_key_create()* function shall create a thread-specific data key visible to all threads in
 51145 the process. Key values provided by *pthread_key_create()* are opaque objects used to locate
 51146 thread-specific data. Although the same key value may be used by different threads, the values
 51147 bound to the key by *pthread_setspecific()* are maintained on a per-thread basis and persist for the
 51148 life of the calling thread.

51149 Upon key creation, the value NULL shall be associated with the new key in all active threads.
 51150 Upon thread creation, the value NULL shall be associated with all defined keys in the new
 51151 thread.

51152 An optional destructor function may be associated with each key value. At thread exit, if a key
 51153 value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with
 51154 that key, the value of the key is set to NULL, and then the function pointed to is called with the
 51155 previously associated value as its sole argument. The order of destructor calls is unspecified if
 51156 more than one destructor exists for a thread when it exits.

51157 If, after all the destructors have been called for all non-NULL values with associated destructors,
 51158 there are still some non-NULL values with associated destructors, then the process is repeated.
 51159 If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for
 51160 outstanding non-NULL values, there are still some non-NULL values with associated
 51161 destructors, implementations may stop calling destructors, or they may continue calling
 51162 destructors until no non-NULL values with associated destructors exist, even though this might
 51163 result in an infinite loop.

51164 **RETURN VALUE**

51165 If successful, the *pthread_key_create()* function shall store the newly created key value at **key* and
 51166 shall return zero. Otherwise, an error number shall be returned to indicate the error.

51167 **ERRORS**51168 The *pthread_key_create()* function shall fail if:

51169 [EAGAIN] The system lacked the necessary resources to create another thread-specific
 51170 data key, or the system-imposed limit on the total number of keys per process
 51171 {PTHREAD_KEYS_MAX} has been exceeded.

51172 [ENOMEM] Insufficient memory exists to create the key.

51173 The *pthread_key_create()* function shall not return an error code of [EINTR].

51174 **EXAMPLES**

51175 The following example demonstrates a function that initializes a thread-specific data key when it
 51176 is first called, and associates a thread-specific object with each calling thread, initializing this
 51177 object when necessary.

```
51178 static pthread_key_t key;
51179 static pthread_once_t key_once = PTHREAD_ONCE_INIT;

51180 static void
51181 make_key()
51182 {
```

```

51183         (void) pthread_key_create(&key, NULL);
51184     }
51185     func()
51186     {
51187         void *ptr;
51188
51189         (void) pthread_once(&key_once, make_key);
51190         if ((ptr = pthread_getspecific(key)) == NULL) {
51191             ptr = malloc(OBJECT_SIZE);
51192             ...
51193             (void) pthread_setspecific(key, ptr);
51194         }
51195     }

```

Note that the key has to be initialized before *pthread_getspecific()* or *pthread_setspecific()* can be used. The *pthread_key_create()* call could either be explicitly made in a module initialization routine, or it can be done implicitly by the first call to a module as in this example. Any attempt to use the key before it is initialized is a programming error, making the code below incorrect.

```

51200     static pthread_key_t key;
51201     func()
51202     {
51203         void *ptr;
51204
51205         /* KEY NOT INITIALIZED!!! THIS WON'T WORK!!! */
51206         if ((ptr = pthread_getspecific(key)) == NULL &&
51207             pthread_setspecific(key, NULL) != 0) {
51208             pthread_key_create(&key, NULL);
51209             ...
51210         }
51211     }

```

APPLICATION USAGE

None.

RATIONALE

Destructor Functions

Normally, the value bound to a key on behalf of a particular thread is a pointer to storage allocated dynamically on behalf of the calling thread. The destructor functions specified with *pthread_key_create()* are intended to be used to free this storage when the thread exits. Thread cancellation cleanup handlers cannot be used for this purpose because thread-specific data may persist outside the lexical scope in which the cancellation cleanup handlers operate.

If the value associated with a key needs to be updated during the lifetime of the thread, it may be necessary to release the storage associated with the old value before the new value is bound. Although the *pthread_setspecific()* function could do this automatically, this feature is not needed often enough to justify the added complexity. Instead, the programmer is responsible for freeing the stale storage:

```

51225     pthread_getspecific(key, &old);
51226     new = allocate();
51227     destructor(old);
51228     pthread_setspecific(key, new);

```

Note: The above example could leak storage if run with asynchronous cancellation enabled. No such problems occur in the default cancellation state if no cancellation points occur between the get and set.

There is no notion of a destructor-safe function. If an application does not call *pthread_exit()* from a signal handler, or if it blocks any signal whose handler may call *pthread_exit()* while calling async-unsafe functions, all functions may be safely called from destructors.

Non-Idempotent Data Key Creation

There were requests to make *pthread_key_create()* idempotent with respect to a given *key* address parameter. This would allow applications to call *pthread_key_create()* multiple times for a given *key* address and be guaranteed that only one key would be created. Doing so would require the key value to be previously initialized (possibly at compile time) to a known null value and would require that implicit mutual-exclusion be performed based on the address and contents of the *key* parameter in order to guarantee that exactly one key would be created.

Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create()*. On many implementations, implicit mutual-exclusion would also have to be performed by *pthread_getspecific()* and *pthread_setspecific()* in order to guard against using incompletely stored or not-yet-visible key values. This could significantly increase the cost of important operations, particularly *pthread_getspecific()*.

Thus, this proposal was rejected. The *pthread_key_create()* function performs no implicit synchronization. It is the responsibility of the programmer to ensure that it is called exactly once per key before use of the key. Several straightforward mechanisms can already be used to accomplish this, including calling explicit module initialization functions, using mutexes, and using *pthread_once()*. This places no significant burden on the programmer, introduces no possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows commonly used thread-specific data operations to be more efficient.

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_getspecific\(\)*](#), [*pthread_key_delete\(\)*](#)

XBD [**<pthread.h>**](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_key_create()* function is marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #8 is applied, updating the DESCRIPTION.

Issue 7

The *pthread_key_create()* function is moved from the Threads option to the Base.

51265 **NAME**

51266 pthread_key_delete — thread-specific data key deletion

51267 **SYNOPSIS**

51268 #include <pthread.h>

51269 int pthread_key_delete(pthread_key_t key);

51270 **DESCRIPTION**

51271 The *pthread_key_delete()* function shall delete a thread-specific data key previously returned by
 51272 *pthread_key_create()*. The thread-specific data values associated with *key* need not be NULL at
 51273 the time *pthread_key_delete()* is called. It is the responsibility of the application to free any
 51274 application storage or perform any cleanup actions for data structures related to the deleted key
 51275 or associated thread-specific data in any threads; this cleanup can be done either before or after
 51276 *pthread_key_delete()* is called. Any attempt to use *key* following the call to *pthread_key_delete()*
 51277 results in undefined behavior.

51278 The *pthread_key_delete()* function shall be callable from within destructor functions. No
 51279 destructor functions shall be invoked by *pthread_key_delete()*. Any destructor function that may
 51280 have been associated with *key* shall no longer be called upon thread exit.

51281 **RETURN VALUE**

51282 If successful, the *pthread_key_delete()* function shall return zero; otherwise, an error number shall
 51283 be returned to indicate the error.

51284 **ERRORS**

51285 The *pthread_key_delete()* function may fail if:

51286 [EINVAL] The *key* value is invalid.

51287 The *pthread_key_delete()* function shall not return an error code of [EINTR].

51288 **EXAMPLES**

51289 None.

51290 **APPLICATION USAGE**

51291 None.

51292 **RATIONALE**

51293 A thread-specific data key deletion function has been included in order to allow the resources
 51294 associated with an unused thread-specific data key to be freed. Unused thread-specific data keys
 51295 can arise, among other scenarios, when a dynamically loaded module that allocated a key is
 51296 unloaded.

51297 Conforming applications are responsible for performing any cleanup actions needed for data
 51298 structures associated with the key to be deleted, including data referenced by thread-specific
 51299 data values. No such cleanup is done by *pthread_key_delete()*. In particular, destructor functions
 51300 are not called. There are several reasons for this division of responsibility:

- 51301 1. The associated destructor functions used to free thread-specific data at thread exit time
 51302 are only guaranteed to work correctly when called in the thread that allocated the thread-
 51303 specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot
 51304 be used to free thread-specific data in other threads at key deletion time. Attempting to
 51305 have them called by other threads at key deletion time would require other threads to be
 51306 asynchronously interrupted. But since interrupted threads could be in an arbitrary state,
 51307 including holding locks necessary for the destructor to run, this approach would fail. In
 51308 general, there is no safe mechanism whereby an implementation could free thread-
 51309 specific data at key deletion time.

51310 2. Even if there were a means of safely freeing thread-specific data associated with keys to
 51311 be deleted, doing so would require that implementations be able to enumerate the
 51312 threads with non-NULL data and potentially keep them from creating more thread-
 51313 specific data while the key deletion is occurring. This special case could cause extra
 51314 synchronization in the normal case, which would otherwise be unnecessary.

51315 For an application to know that it is safe to delete a key, it has to know that all the threads that
 51316 might potentially ever use the key do not attempt to use it again. For example, it could know
 51317 this if all the client threads have called a cleanup procedure declaring that they are through with
 51318 the module that is being shut down, perhaps by setting a reference count to zero.

FUTURE DIRECTIONS

51319 None.
 51320

SEE ALSO

51321 *pthread_key_create()*

51322 XBD <pthread.h>

CHANGE HISTORY

51324 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
 51325

Issue 6

51326 The *pthread_key_delete()* function is marked as part of the Threads option.
 51327

Issue 7

51328 The *pthread_key_delete()* function is moved from the Threads option to the Base.
 51329

51330 **NAME**

51331 pthread_kill — send a signal to a thread

51332 **SYNOPSIS**

51333 CX #include <signal.h>

51334 int pthread_kill(pthread_t thread, int sig);

51335 **DESCRIPTION**51336 The *pthread_kill()* function shall request that a signal be delivered to the specified thread.51337 As in *kill()*, if *sig* is zero, error checking shall be performed but no signal shall actually be sent.51338 **RETURN VALUE**51339 Upon successful completion, the function shall return a value of zero. Otherwise, the function
51340 shall return an error number. If the *pthread_kill()* function fails, no signal shall be sent.51341 **ERRORS**51342 The *pthread_kill()* function shall fail if:51343 [ESRCH] No thread could be found corresponding to that specified by the given thread
51344 ID.51345 [EINVAL] The value of the *sig* argument is an invalid or unsupported signal number.51346 The *pthread_kill()* function shall not return an error code of [EINTR].51347 **EXAMPLES**

51348 None.

51349 **APPLICATION USAGE**51350 The *pthread_kill()* function provides a mechanism for asynchronously directing a signal at a
51351 thread in the calling process. This could be used, for example, by one thread to affect broadcast
51352 delivery of a signal to a set of threads.51353 Note that *pthread_kill()* only causes the signal to be handled in the context of the given thread;
51354 the signal action (termination or stopping) affects the process as a whole.51355 **RATIONALE**

51356 None.

51357 **FUTURE DIRECTIONS**

51358 None.

51359 **SEE ALSO**51360 *kill*, *pthread_self()*, *raise()*

51361 XBD <signal.h>

51362 **CHANGE HISTORY**

51363 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51364 **Issue 6**51365 The *pthread_kill()* function is marked as part of the Threads option.

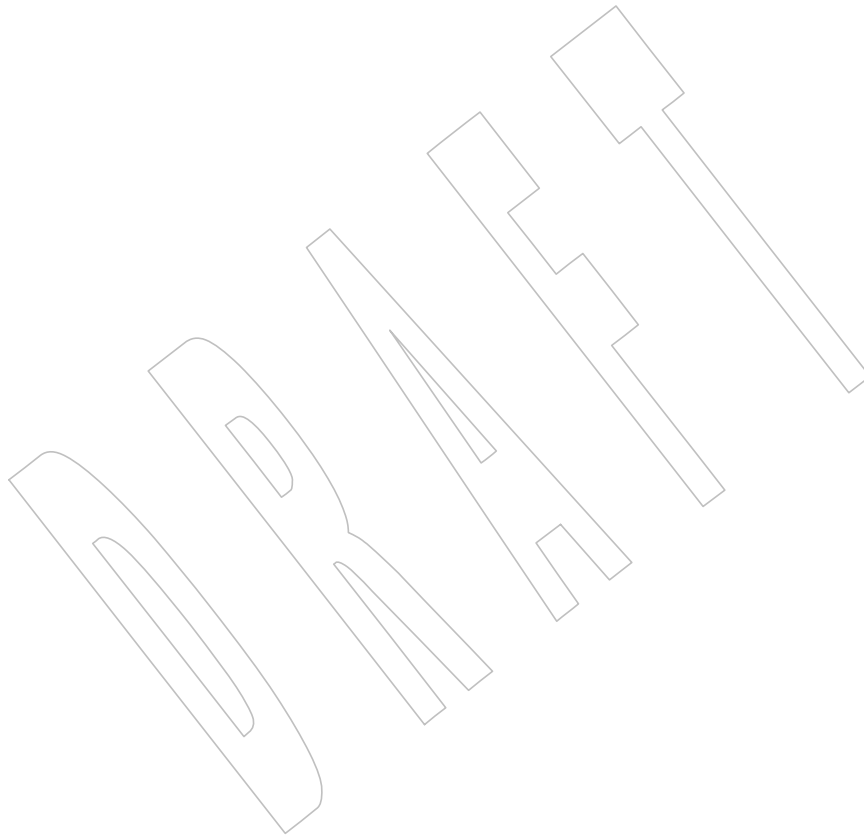
51366 The APPLICATION USAGE section is added.

51367

Issue 7

51368

The *pthread_kill()* function is moved from the Threads option to the Base.



51369 **NAME**

51370 pthread_mutex_consistent — mark state protected by robust mutex as consistent

51371 **SYNOPSIS**

51372 #include <pthread.h>

51373 int pthread_mutex_consistent(pthread_mutex_t *mutex);

51374 **DESCRIPTION**51375 If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent()* function can be
51376 used to mark the state protected by the mutex referenced by *mutex* as consistent again.51377 If an owner of a robust mutex terminates while holding the mutex, the mutex becomes
51378 inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the
51379 return value [EOWNERDEAD]. In this case, the mutex does not become normally usable again
51380 until the state is marked consistent.51381 If the thread which acquired the mutex lock with the return value [EOWNERDEAD] terminates
51382 before calling either *pthread_mutex_consistent()* or *pthread_mutex_unlock()*, the next thread that
51383 acquires the mutex lock shall be notified about the state of the mutex by the return value
51384 [EOWNERDEAD].51385 **RETURN VALUE**51386 Upon successful completion, the *pthread_mutex_consistent()* function shall return zero.
51387 Otherwise, an error value shall be returned to indicate the error.51388 **ERRORS**51389 The *pthread_mutex_consistent()* function shall fail if:51390 [EINVAL] The mutex object referenced by *mutex* is not robust or does not protect an
51391 inconsistent state.51392 The *pthread_mutex_consistent()* function may fail if:51393 [EINVAL] The value *mutex* is invalid.

51394 These functions shall not return an error code of [EINTR].

51395 **EXAMPLES**

51396 None.

51397 **APPLICATION USAGE**51398 The *pthread_mutex_consistent()* function is only responsible for notifying the implementation that
51399 the state protected by the mutex has been recovered and that normal operations with the mutex
51400 can be resumed. It is the responsibility of the application to recover the state so it can be reused.
51401 If the application is not able to perform the recovery, it can notify the implementation that the
51402 situation is unrecoverable by a call to *pthread_mutex_unlock()* without a prior call to
51403 *pthread_mutex_consistent()*, in which case subsequent threads that attempt to lock the mutex will
51404 fail to acquire the lock and be returned [ENOTRECOVERABLE].51405 **RATIONALE**

51406 None.

51407 **FUTURE DIRECTIONS**

51408 None.

pthread_mutex_consistent()*System Interfaces*

51409

SEE ALSO

51410

pthread_mutex_lock(), *pthread_mutexattr_getrobust()*

51411

XBD <pthread.h>

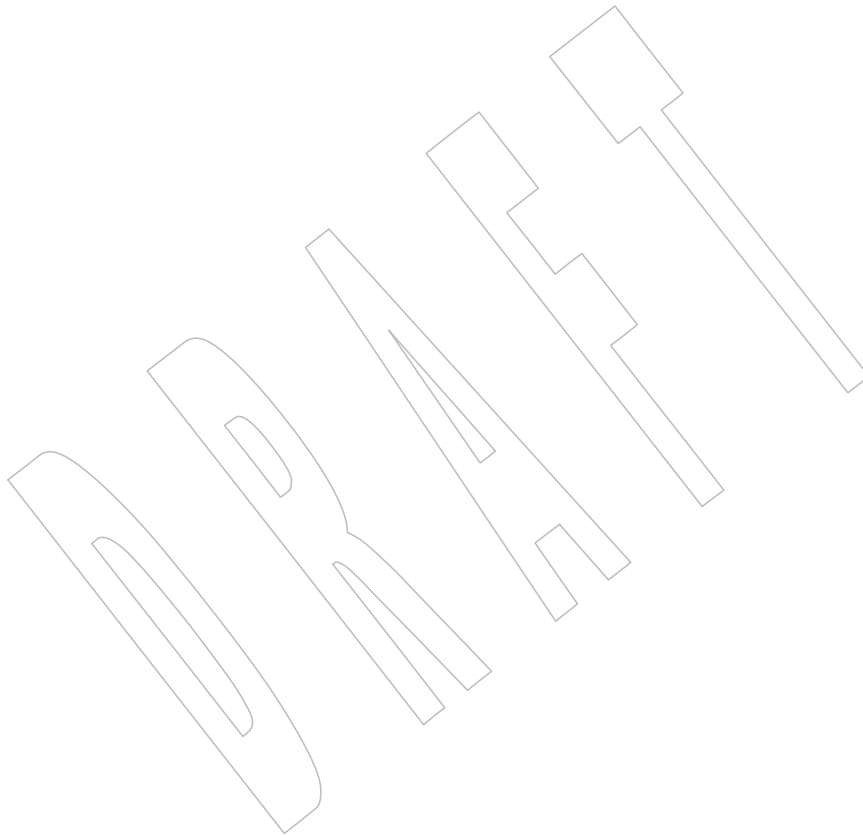
51412

CHANGE HISTORY

51413

First released in Issue 7.

51414

The *pthread_mutex_consistent()* function is moved from the Threads option to the Base.

51415 **NAME**

51416 pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

51417 **SYNOPSIS**

51418 #include <pthread.h>

51419 int pthread_mutex_destroy(pthread_mutex_t *mutex);

51420 int pthread_mutex_init(pthread_mutex_t *restrict mutex,

51421 const pthread_mutexattr_t *restrict attr);

51422 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

51423 **DESCRIPTION**

51424 The *pthread_mutex_destroy()* function shall destroy the mutex object referenced by *mutex*; the
 51425 mutex object becomes, in effect, uninitialized. An implementation may cause
 51426 *pthread_mutex_destroy()* to set the object referenced by *mutex* to an invalid value. A destroyed
 51427 mutex object can be reinitialized using *pthread_mutex_init()*; the results of otherwise referencing
 51428 the object after it has been destroyed are undefined.

51429 It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked
 51430 mutex results in undefined behavior.

51431 The *pthread_mutex_init()* function shall initialize the mutex referenced by *mutex* with attributes
 51432 specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the
 51433 same as passing the address of a default mutex attributes object. Upon successful initialization,
 51434 the state of the mutex becomes initialized and unlocked.

51435 Only *mutex* itself may be used for performing synchronization. The result of referring to copies
 51436 of *mutex* in calls to *pthread_mutex_lock()*, *pthread_mutex_trylock()*, *pthread_mutex_unlock()*, and
 51437 *pthread_mutex_destroy()* is undefined.

51438 Attempting to initialize an already initialized mutex results in undefined behavior.

51439 In cases where default mutex attributes are appropriate, the macro
 51440 PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes that are statically allocated.
 51441 The effect shall be equivalent to dynamic initialization by a call to *pthread_mutex_init()* with
 51442 parameter *attr* specified as NULL, except that no error checks are performed.

51443 **RETURN VALUE**

51444 If successful, the *pthread_mutex_destroy()* and *pthread_mutex_init()* functions shall return zero;
 51445 otherwise, an error number shall be returned to indicate the error.

51446 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed
 51447 immediately at the beginning of processing for the function and shall cause an error return prior
 51448 to modifying the state of the mutex specified by *mutex*.

51449 **ERRORS**

51450 The *pthread_mutex_destroy()* function may fail if:

51451 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 51452 by *mutex* while it is locked or referenced (for example, while being used in a
 51453 *pthread_cond_timedwait()* or *pthread_cond_wait()*) by another thread.

51454 [EINVAL] The value specified by *mutex* is invalid.

51455 The *pthread_mutex_init()* function shall fail if:

51456 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
 51457 another mutex.

- 51458 [ENOMEM] Insufficient memory exists to initialize the mutex.
- 51459 [EPERM] The caller does not have the privilege to perform the operation.
- 51460 The *pthread_mutex_init()* function may fail if:
- 51461 [EBUSY] The implementation has detected an attempt to reinitialize the object
51462 referenced by *mutex*, a previously initialized, but not yet destroyed, mutex.
- 51463 [EINVAL] The value specified by *attr* is invalid.
- 51464 [EINVAL] The attributes object referenced by *attr* has the robust mutex attribute set
51465 without the process-shared attribute being set.
- 51466 These functions shall not return an error code of [EINTR].

EXAMPLES

51467 None.
51468

APPLICATION USAGE

51469 None.
51470

RATIONALE**Alternate Implementations Possible**

51472 This volume of POSIX.1-200x supports several alternative implementations of mutexes. An
51473 implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively,
51474 an implementation may store the lock in the heap and merely store a pointer, handle, or unique
51475 ID in the mutex object. Either implementation has advantages or may be required on certain
51476 hardware configurations. So that portable code can be written that is invariant to this choice, this
51477 volume of POSIX.1-200x does not define assignment or equality for this type, and it uses the
51478 term “initialize” to reinforce the (more restrictive) notion that the lock may actually reside in the
51479 mutex object itself.
51480

51481 Note that this precludes an over-specification of the type of the mutex or condition variable and
51482 motivates the opaqueness of the type.

51483 An implementation is permitted, but not required, to have *pthread_mutex_destroy()* store an
51484 illegal value into the mutex. This may help detect erroneous programs that try to lock (or
51485 otherwise reference) a mutex that has already been destroyed.

Tradeoff Between Error Checks and Performance Supported

51486 Many of the error checks were made optional in order to let implementations trade off
51487 performance *versus* degree of error checking according to the needs of their specific applications
51488 and execution environment. As a general rule, errors or conditions caused by the system (such
51489 as insufficient memory) always need to be reported, but errors due to an erroneously coded
51490 application (such as failing to provide adequate synchronization to prevent a mutex from being
51491 deleted while in use) are made optional.
51492

51493 A wide range of implementations is thus made possible. For example, an implementation
51494 intended for application debugging may implement all of the error checks, but an
51495 implementation running a single, provably correct application under very tight performance
51496 constraints in an embedded computer might implement minimal checks. An implementation
51497 might even be provided in two versions, similar to the options that compilers provide: a full-
51498 checking, but slower version; and a limited-checking, but faster version. To forbid this
51499 optionality would be a disservice to users.

51500 By carefully limiting the use of “undefined behavior” only to things that an erroneous (badly
51501 coded) application might do, and by defining that resource-not-available errors are mandatory,
51502 this volume of POSIX.1-200x ensures that a fully-conforming application is portable across the

51503 full range of implementations, while not forcing all implementations to add overhead to check
51504 for numerous things that a correct program never does.

51505 **Why No Limits are Defined**

51506 Defining symbols for the maximum number of mutexes and condition variables was considered
51507 but rejected because the number of these objects may change dynamically. Furthermore, many
51508 implementations place these objects into application memory; thus, there is no explicit
51509 maximum.

51510 **Static Initializers for Mutexes and Condition Variables**

51511 Providing for static initialization of statically allocated synchronization objects allows modules
51512 with private static synchronization variables to avoid runtime initialization tests and overhead.
51513 Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in
51514 C libraries, where for various reasons the design calls for self-initialization instead of requiring
51515 an explicit module initialization function to be called. An example use of static initialization
51516 follows.

51517 Without static initialization, a self-initializing routine *foo()* might look as follows:

```
51518 static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
51519 static pthread_mutex_t foo_mutex;

51520 void foo_init()
51521 {
51522     pthread_mutex_init(&foo_mutex, NULL);
51523 }

51524 void foo()
51525 {
51526     pthread_once(&foo_once, foo_init);
51527     pthread_mutex_lock(&foo_mutex);
51528     /* Do work. */
51529     pthread_mutex_unlock(&foo_mutex);
51530 }
```

51531 With static initialization, the same routine could be coded as follows:

```
51532 static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

51533 void foo()
51534 {
51535     pthread_mutex_lock(&foo_mutex);
51536     /* Do work. */
51537     pthread_mutex_unlock(&foo_mutex);
51538 }
```

51539 Note that the static initialization both eliminates the need for the initialization test inside
51540 *pthread_once()* and the fetch of *&foo_mutex* to learn the address to be passed to
51541 *pthread_mutex_lock()* or *pthread_mutex_unlock()*.

51542 Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a
51543 large class of systems; those where the (entire) synchronization object can be stored in
51544 application memory.

51545 Yet the locking performance question is likely to be raised for machines that require mutexes to
51546 be allocated out of special memory. Such machines actually have to have mutexes and possibly
51547 condition variables contain pointers to the actual hardware locks. For static initialization to work
51548 on such machines, *pthread_mutex_lock()* also has to test whether or not the pointer to the actual

51549 lock has been allocated. If it has not, *pthread_mutex_lock()* has to initialize it before use. The
 51550 reservation of such resources can be made when the program is loaded, and hence return codes
 51551 have not been added to mutex locking and condition variable waiting to indicate failure to
 51552 complete initialization.

51553 This runtime test in *pthread_mutex_lock()* would at first seem to be extra work; an extra test is
 51554 required to see whether the pointer has been initialized. On most machines this would actually
 51555 be implemented as a fetch of the pointer, testing the pointer against zero, and then using the
 51556 pointer if it has already been initialized. While the test might seem to add extra work, the extra
 51557 effort of testing a register is usually negligible since no extra memory references are actually
 51558 done. As more and more machines provide caches, the real expenses are memory references, not
 51559 instructions executed.

51560 Alternatively, depending on the machine architecture, there are often ways to eliminate *all*
 51561 overhead in the most important case: on the lock operations that occur *after* the lock has been
 51562 initialized. This can be done by shifting more overhead to the less frequent operation:
 51563 initialization. Since out-of-line mutex allocation also means that an address has to be
 51564 dereferenced to find the actual lock, one technique that is widely applicable is to have static
 51565 initialization store a bogus value for that address; in particular, an address that causes a machine
 51566 fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity
 51567 checks can be done, and then the correct address for the actual lock can be filled in. Subsequent
 51568 lock operations incur no extra overhead since they do not “fault”. This is merely one technique
 51569 that can be used to support static initialization, while not adversely affecting the performance of
 51570 lock acquisition. No doubt there are other techniques that are highly machine-dependent.

51571 The locking overhead for machines doing out-of-line mutex allocation is thus similar for
 51572 modules being implicitly initialized, where it is improved for those doing mutex allocation
 51573 entirely inline. The inline case is thus made much faster, and the out-of-line case is not
 51574 significantly worse.

51575 Besides the issue of locking performance for such machines, a concern is raised that it is possible
 51576 that threads would serialize contending for initialization locks when attempting to finish
 51577 initializing statically allocated mutexes. (Such finishing would typically involve taking an
 51578 internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing
 51579 the internal lock.) First, many implementations would reduce such serialization by hashing on
 51580 the mutex address. Second, such serialization can only occur a bounded number of times. In
 51581 particular, it can happen at most as many times as there are statically allocated synchronization
 51582 objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init()* or
 51583 *pthread_cond_init()*.

51584 Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient
 51585 performance for an application on some implementation, the application can avoid static
 51586 initialization altogether by explicitly initializing all synchronization objects with the
 51587 corresponding *pthread_*_init()* functions, which are supported by all implementations. An
 51588 implementation can also document the tradeoffs and advise which initialization technique is
 51589 more efficient for that particular implementation.

51590 **Destroying Mutexes**

51591 A mutex can be destroyed immediately after it is unlocked. For example, consider the following
 51592 code:

```
51593 struct obj {
51594     pthread_mutex_t om;
51595     int refcnt;
51596     ...
51597 };
```

```

51598     obj_done(struct obj *op)
51599     {
51600         pthread_mutex_lock(&op->om);
51601         if (--op->refcnt == 0) {
51602             pthread_mutex_unlock(&op->om);
51603             (A)     pthread_mutex_destroy(&op->om);
51604             (B)     free(op);
51605         } else
51606             (C)     pthread_mutex_unlock(&op->om);
51607     }

```

51608 In this case *obj* is reference counted and *obj_done()* is called whenever a reference to the object is
51609 dropped. Implementations are required to allow an object to be destroyed and freed and
51610 potentially unmapped (for example, lines A and B) immediately after the object is unlocked (line
51611 C).

51612 **Robust Mutexes**

51613 Implementations are required to provide robust mutexes for mutexes with the process-shared
51614 attribute set to PTHREAD_PROCESS_SHARED. Implementations are allowed, but not required,
51615 to provide robust mutexes when the process-shared attribute is set to
51616 PTHREAD_PROCESS_PRIVATE.

51617 **FUTURE DIRECTIONS**

51618 None.

51619 **SEE ALSO**

51620 *pthread_mutex_getprioceiling()*, *pthread_mutexattr_getrobust()*, *pthread_mutex_lock()*,
51621 *pthread_mutex_timedlock()*, *pthread_mutexattr_getpshared()*

51622 XBD [<pthread.h>](#) +

51623 **CHANGE HISTORY**

51624 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51625 **Issue 6**

51626 The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are marked as part of the
51627 Threads option.

51628 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
51629 IEEE Std 1003.1d-1999.

51630 IEEE PASC Interpretation 1003.1c #34 is applied, updating the DESCRIPTION.

51631 The **restrict** keyword is added to the *pthread_mutex_init()* prototype for alignment with the
51632 ISO/IEC 9899:1999 standard.

51633 **Issue 7**

51634 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

51635 The *pthread_mutex_destroy()* and *pthread_mutex_init()* functions are moved from the Threads
51636 option to the Base.

51637 **NAME**

51638 pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling
 51639 of a mutex (**REALTIME THREADS**)

51640 **SYNOPSIS**

```
51641 RPP|TPP #include <pthread.h>
51642
51642 int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
51643 int *restrict prioceiling);
51644 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
51645 int prioceiling, int *restrict old_ceiling);
```

51646 **DESCRIPTION**

51647 The *pthread_mutex_getprioceiling()* function shall return the current priority ceiling of the mutex.

51648 The *pthread_mutex_setprioceiling()* function shall either lock the mutex if it is unlocked, or block
 51649 until it can successfully lock the mutex, then it shall change the mutex's priority ceiling and
 51650 release the mutex. When the change is successful, the previous value of the priority ceiling shall
 51651 be returned in *old_ceiling*. The process of locking the mutex need not adhere to the priority
 51652 protect protocol.

51653 If *pthread_mutex_setprioceiling()* is called while holding the mutex, the result is undefined unless
 51654 the mutex is of type PTHREAD_MUTEX_RECURSIVE.

51655 If the *pthread_mutex_setprioceiling()* function fails, the mutex priority ceiling shall not be
 51656 changed.

51657 **RETURN VALUE**

51658 If successful, the *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions shall
 51659 return zero; otherwise, an error number shall be returned to indicate the error.

51660 **ERRORS**

51661 These functions shall fail if:

51662 [EINVAL] The protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

51663 These functions may fail if:

51664 [EDEADLK] The current thread already owns the mutex.

51665 [EINVAL] The priority requested by *prioceiling* is out of range.

51666 [EINVAL] The value specified by *mutex* does not refer to a currently existing mutex.

51667 [EPERM] The caller does not have the privilege to perform the operation.

51668 These functions shall not return an error code of [EINTR].

51669 **EXAMPLES**

51670 None.

51671 **APPLICATION USAGE**

51672 None.

51673 **RATIONALE**

51674 None.

51675 **FUTURE DIRECTIONS**

51676 None.

51677 **SEE ALSO**51678 *pthread_mutex_destroy()*, *pthread_mutex_lock()*, *pthread_mutex_timedlock()*

51679 XBD <pthread.h>

51680 **CHANGE HISTORY**

51681 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

51682 Marked as part of the Realtime Threads Feature Group.

51683 **Issue 6**51684 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are marked as
51685 part of the Threads and Thread Priority Protection options.

51686 The [ENOSYS] error conditions have been removed.

51687 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
51688 IEEE Std 1003.1d-1999.51689 The **restrict** keyword is added to the *pthread_mutex_getprioceiling()* and
51690 *pthread_mutex_setprioceiling()* prototypes for alignment with the ISO/IEC 9899:1999 standard.51691 **Issue 7**

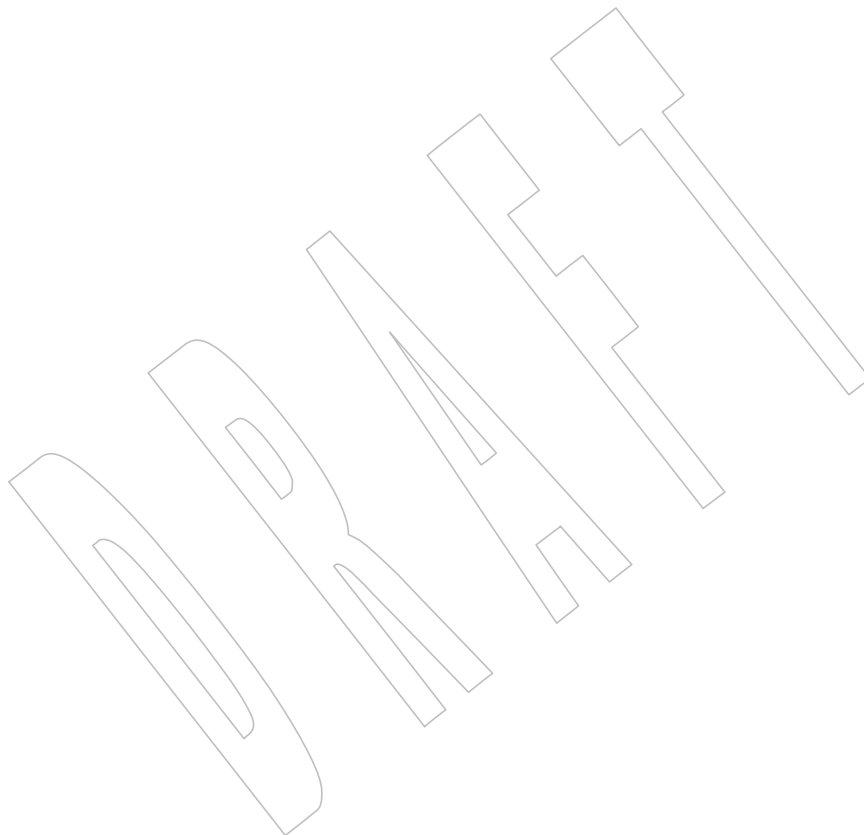
51692 SD5-XSH-ERN-39 is applied.

51693 Austin Group Interpretation 1003.1-2001 #052 is applied, adding [EDEADLK] as a “may fail”
51694 error.51695 SD5-XSH-ERN-158 is applied, updating the ERRORS section to include a “shall fail” error case
51696 for when the protocol attribute of *mutex* is PTHREAD_PRIO_NONE.51697 The *pthread_mutex_getprioceiling()* and *pthread_mutex_setprioceiling()* functions are moved from
51698 the Threads option to require support of either the Robust Mutex Priority Protection option or
51699 the Non-Robust Mutex Priority Protection option.

51700 **NAME**
51701 pthread_mutex_init — destroy and initialize a mutex

51702 **SYNOPSIS**
51703 #include <pthread.h>
51704 int pthread_mutex_init(pthread_mutex_t *restrict mutex,
51705 const pthread_mutexattr_t *restrict attr);
51706 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

51707 **DESCRIPTION**
51708 Refer to *pthread_mutex_destroy()*.



51709 **NAME**

51710 pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a
51711 mutex

51712 **SYNOPSIS**

51713 #include <pthread.h>

51714 int pthread_mutex_lock(pthread_mutex_t *mutex);
51715 int pthread_mutex_trylock(pthread_mutex_t *mutex);
51716 int pthread_mutex_unlock(pthread_mutex_t *mutex);

51717 **DESCRIPTION**

51718 The mutex object referenced by *mutex* shall be locked by calling *pthread_mutex_lock()*. If the
51719 mutex is already locked, the calling thread shall block until the mutex becomes available. This
51720 operation shall return with the mutex object referenced by *mutex* in the locked state with the
51721 calling thread as its owner.

51722 If the mutex type is PTHREAD_MUTEX_NORMAL, deadlock detection shall not be provided.
51723 Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it
51724 has not locked or a mutex which is unlocked, undefined behavior results.

51725 If the mutex type is PTHREAD_MUTEX_ERRORCHECK, then error checking shall be provided.
51726 If a thread attempts to relock a mutex that it has already locked, an error shall be returned. If a
51727 thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error
51728 shall be returned.

51729 If the mutex type is PTHREAD_MUTEX_RECURSIVE, then the mutex shall maintain the
51730 concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock
51731 count shall be set to one. Every time a thread relocks this mutex, the lock count shall be
51732 incremented by one. Each time the thread unlocks the mutex, the lock count shall be
51733 decremented by one. When the lock count reaches zero, the mutex shall become available for
51734 other threads to acquire. If a thread attempts to unlock a mutex that it has not locked or a mutex
51735 which is unlocked, an error shall be returned.

51736 If the mutex type is PTHREAD_MUTEX_DEFAULT, attempting to recursively lock the mutex
51737 results in undefined behavior. Attempting to unlock the mutex if it was not locked by the calling
51738 thread results in undefined behavior. Attempting to unlock the mutex if it is not locked results in
51739 undefined behavior.

51740 The *pthread_mutex_trylock()* function shall be equivalent to *pthread_mutex_lock()*, except that if
51741 the mutex object referenced by *mutex* is currently locked (by any thread, including the current
51742 thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE
51743 and the mutex is currently owned by the calling thread, the mutex lock count shall be
51744 incremented by one and the *pthread_mutex_trylock()* function shall immediately return success.

51745 The *pthread_mutex_unlock()* function shall release the mutex object referenced by *mutex*. The
51746 manner in which a mutex is released is dependent upon the mutex's type attribute. If there are
51747 threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock()* is called,
51748 resulting in the mutex becoming available, the scheduling policy shall determine which thread
51749 shall acquire the mutex.

51750 (In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available
51751 when the count reaches zero and the calling thread no longer has any locks on this mutex.)

51752 If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the
51753 thread shall resume waiting for the mutex as if it was not interrupted.

51754 If *mutex* is a robust mutex and the process containing the owning thread terminated while

pthread_mutex_lock()

51755 holding the mutex lock, a call to *pthread_mutex_lock()* shall return the error value
 51756 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding
 51757 the mutex lock, a call to *pthread_mutex_lock()* may return the error value [EOWNERDEAD] even
 51758 if the process in which the owning thread resides has not terminated. In these cases, the mutex is
 51759 locked by the thread but the state it protects is marked as inconsistent. The application should
 51760 ensure that the state is made consistent for reuse and when that is complete call
 51761 *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the
 51762 mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked
 51763 permanently unusable.

RETURN VALUE

51764 If successful, the *pthread_mutex_lock()* and *pthread_mutex_unlock()* functions shall return zero;
 51765 otherwise, an error number shall be returned to indicate the error.

51767 The *pthread_mutex_trylock()* function shall return zero if a lock on the mutex object referenced by
 51768 *mutex* is acquired. Otherwise, an error number is returned to indicate the error.

ERRORS

51769 The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions shall fail if:

51771 RPP|TPP [EINVAL] The *mutex* was created with the protocol attribute having the value
 51772 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
 51773 the mutex's current priority ceiling.

51774 [ENOTRECOVERABLE]

51775 The state protected by the mutex is not recoverable. The mutex is not locked.

51776 [EOWNERDEAD]

51777 The mutex is a robust mutex and the process containing the previous owning
 51778 thread terminated while holding the mutex lock. The mutex lock has been
 51779 acquired and it is up to the new owner to make the state consistent.

51780 The *pthread_mutex_trylock()* function shall fail if:

51781 [EBUSY] The *mutex* could not be acquired because it was already locked.

51782 The *pthread_mutex_unlock()* function shall fail if:

51783 [EPERM] The current thread does not own the mutex and the mutex is a robust mutex.

51784 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions may fail
 51785 if:

51786 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.

51787 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
 51788 locks for *mutex* has been exceeded.

51789 The *pthread_mutex_lock()* and *pthread_mutex_trylock()* functions may fail if:

51790 [EOWNERDEAD]

51791 The mutex is a robust mutex and the previous owning thread terminated
 51792 while holding the mutex lock. The mutex lock has been acquired and it is up
 51793 to the new owner to make the state consistent.

51794 The *pthread_mutex_lock()* function may fail if:

51795 [EDEADLK] A deadlock condition was detected or the current thread already owns the
 51796 mutex.

51797 The *pthread_mutex_unlock()* function may fail if:

51798 [EPERM] The current thread does not own the mutex.

51799 These functions shall not return an error code of [EINTR].

EXAMPLES

51800 None.

APPLICATION USAGE

51803 Applications that have assumed that non-zero return values are errors will need updating for
 51804 use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting
 51805 a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error
 51806 returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If
 51807 an application is supposed to work with normal and robust mutexes it should check all return
 51808 values for error conditions and if necessary take appropriate action.

RATIONALE

51809 Mutex objects are intended to serve as a low-level primitive from which other thread
 51810 synchronization functions can be built. As such, the implementation of mutexes should be as
 51811 efficient as possible, and this has ramifications on the features available at the interface.
 51812

51813 The mutex functions and the particular default settings of the mutex attributes have been
 51814 motivated by the desire to not preclude fast, inlined implementations of mutex locking and
 51815 unlocking.

51816 Since most attributes only need to be checked when a thread is going to be blocked, the use of
 51817 attributes does not slow the (common) mutex-locking case.

51818 Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it
 51819 would require storing the current thread ID when each mutex is locked, and this could incur
 51820 unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

51821 For further rationale on the extended mutex types, see the Rationale (Informative) volume of
 51822 POSIX.1-200x.

FUTURE DIRECTIONS

51823 None.

SEE ALSO

51826 [pthread_mutex_consistent\(\)](#), [pthread_mutex_destroy\(\)](#), [pthread_mutex_timedlock\(\)](#),
 51827 [pthread_mutexattr_getrobust\(\)](#)

51828 XBD Section 4.11 (on page 98), [<pthread.h>](#)

CHANGE HISTORY

51829 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

51831 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
 51832 marked as part of the Threads option.
 51833

51834 The following new requirements on POSIX implementations derive from alignment with the
 51835 Single UNIX Specification:

- 51836 • The behavior when attempting to relock a mutex is defined.

51837 The *pthread_mutex_timedlock()* function is added to the SEE ALSO section for alignment with
 51838 IEEE Std 1003.1d-1999.

51839 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/98 is applied, updating the ERRORS
 51840 section so that the [EDEADLK] error includes detection of a deadlock condition. The
 51841 RATIONALE section is also reworded to take into account non-XSI-conformant systems.

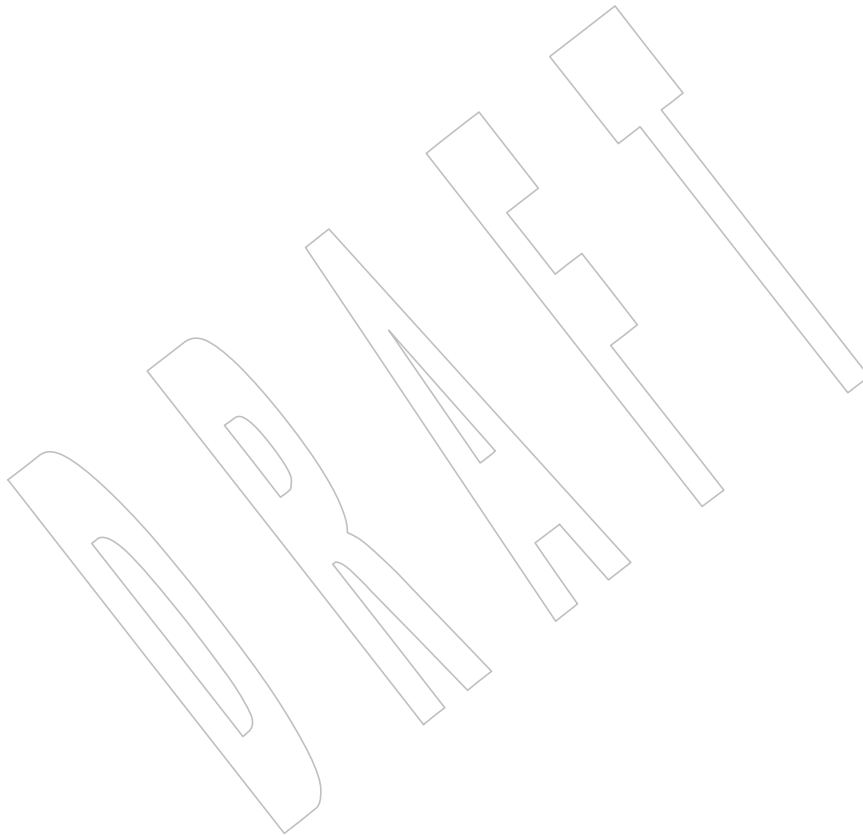
pthread_mutex_lock()51842 **Issue 7**

51843 SD5-XSH-ERN-43 is applied, marking the “shall fail” case of the [EINVAL] error as dependent
51844 on the Thread Priority Protection option.

51845 Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

51846 The *pthread_mutex_lock()*, *pthread_mutex_trylock()*, and *pthread_mutex_unlock()* functions are
51847 moved from the Threads option to the Base.

51848 The PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRORCHECK,
51849 PTHREAD_MUTEX_RECURSIVE, and PTHREAD_MUTEX_DEFAULT extended mutex types
51850 are moved from the XSI option to the Base.



51851 **NAME**

51852 pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME**
51853 **THREADS**)

51854 **SYNOPSIS**

```
51855 RPP|TPP #include <pthread.h>  
51856 int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,  
51857 int prioceiling, int *restrict old_ceiling);
```

51858 **DESCRIPTION**

51859 Refer to [pthread_mutex_getprioceiling\(\)](#).

51860 **NAME**

51861 pthread_mutex_timedlock — lock a mutex

51862 **SYNOPSIS**

51863 #include <pthread.h>

51864 #include <time.h>

51865 int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
51866 const struct timespec *restrict abstime);51867 **DESCRIPTION**51868 The *pthread_mutex_timedlock()* function shall lock the mutex object referenced by *mutex*. If the
51869 mutex is already locked, the calling thread shall block until the mutex becomes available as in
51870 the *pthread_mutex_lock()* function. If the mutex cannot be locked without waiting for another
51871 thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.51872 The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the
51873 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
51874 *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the
51875 call.51876 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
51877 be the resolution of the clock on which it is based. The **timespec** data type is defined in the
51878 <**time.h**> header.51879 Under no circumstance shall the function fail with a timeout if the mutex can be locked
51880 immediately. The validity of the *abstime* parameter need not be checked if the mutex can be
51881 locked immediately.51882 RPI | TPI As a consequence of the priority inheritance rules (for mutexes initialized with the
51883 PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the
51884 priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this
51885 thread is no longer among the threads waiting for the mutex.51886 If *mutex* is a robust mutex and the process containing the owning thread terminated while
51887 holding the mutex lock, a call to *pthread_mutex_timedlock()* shall return the error value
51888 [EOWNERDEAD]. If *mutex* is a robust mutex and the owning thread terminated while holding
51889 the mutex lock, a call to *pthread_mutex_timedlock()* may return the error value [EOWNERDEAD]
51890 even if the process in which the owning thread resides has not terminated. In these cases, the
51891 mutex is locked by the thread but the state it protects is marked as inconsistent. The application
51892 should ensure that the state is made consistent for reuse and when that is complete call
51893 *pthread_mutex_consistent()*. If the application is unable to recover the state, it should unlock the
51894 mutex without a prior call to *pthread_mutex_consistent()*, after which the mutex is marked
51895 permanently unusable.51896 **RETURN VALUE**51897 If successful, the *pthread_mutex_timedlock()* function shall return zero; otherwise, an error
51898 number shall be returned to indicate the error.51899 **ERRORS**51900 The *pthread_mutex_timedlock()* function shall fail if:51901 [EINVAL] The mutex was created with the protocol attribute having the value
51902 PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than
51903 the mutex' current priority ceiling.

- 51904 [EINVAL] The process or thread would have blocked, and the *abstime* parameter
51905 specified a nanoseconds field value less than zero or greater than or equal to
51906 1 000 million.
- 51907 [ENOTRECOVERABLE]
51908 The state protected by the mutex is not recoverable. The mutex is not locked.
- 51909 [EOWNERDEAD]
51910 The mutex is a robust mutex and the process containing the previous owning
51911 thread terminated while holding the mutex lock. The mutex lock has been
51912 acquired and it is up to the new owner to make the state consistent.
- 51913 [ETIMEDOUT] The mutex could not be locked before the specified timeout expired.
- 51914 The *pthread_mutex_timedlock()* function may fail if:
- 51915 [EINVAL] The value specified by *mutex* does not refer to an initialized mutex object.
- 51916 [EAGAIN] The mutex could not be acquired because the maximum number of recursive
51917 locks for *mutex* has been exceeded.
- 51918 [EDEADLK] A deadlock condition was detected or the current thread already owns the
51919 mutex.
- 51920 [EOWNERDEAD]
51921 The mutex is a robust mutex and the previous owning thread terminated
51922 while holding the mutex lock. The mutex lock has been acquired and it is up
51923 to the new owner to make the state consistent.
- 51924 This function shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is [EOWNERDEAD]. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO*pthread_mutex_destroy()*, *pthread_mutex_lock()*, *time*XBD Section 4.11 (on page 98), [<pthread.h>](#), [<time.h>](#)**CHANGE HISTORY**

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/99 is applied, marking the last paragraph in the DESCRIPTION as part of the Thread Priority Inheritance option.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/100 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

pthread_mutex_timedlock()

51947

Issue 7

51948

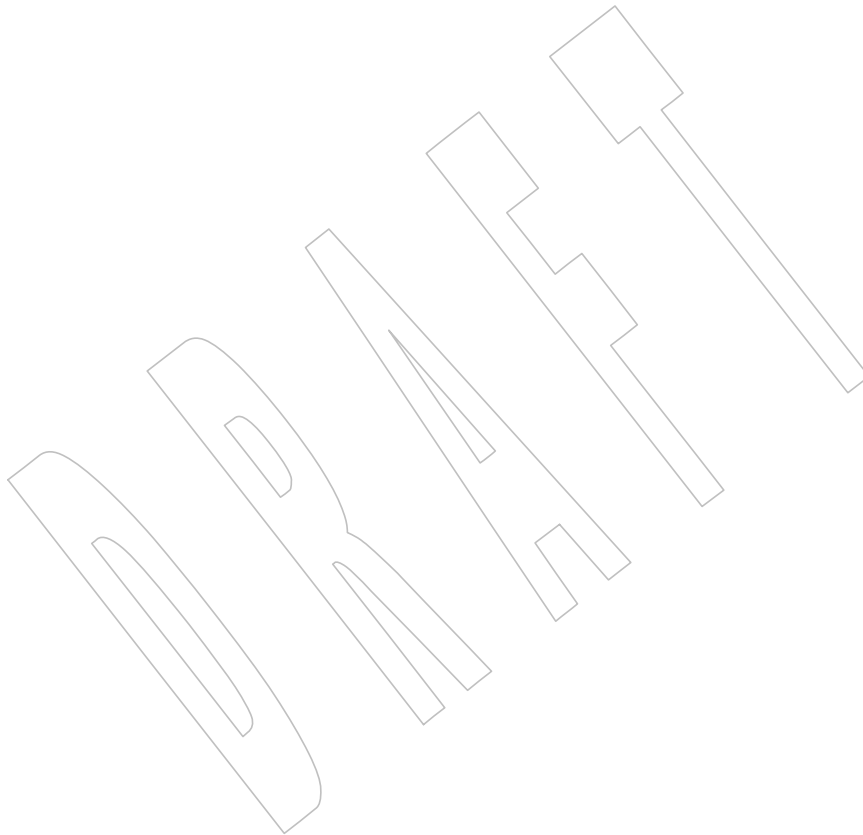
Changes are made from The Open Group Technical Standard, 2006, Extended API Set Part 3.

51949

The *pthread_mutex_timedlock()* function is moved from the Timeouts option to the Base.

51950

Functionality relating to the Timers option is moved to the Base.



51951 **NAME**
51952 pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

51953 **SYNOPSIS**
51954 #include <pthread.h>

51955 int pthread_mutex_trylock(pthread_mutex_t *mutex);
51956 int pthread_mutex_unlock(pthread_mutex_t *mutex);

51957 **DESCRIPTION**
51958 Refer to *pthread_mutex_lock()*.



51959 **NAME**

51960 pthread_mutexattr_destroy, pthread_mutexattr_init — destroy and initialize the mutex
 51961 attributes object

51962 **SYNOPSIS**

51963 #include <pthread.h>

51964 int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
 51965 int pthread_mutexattr_init(pthread_mutexattr_t *attr);

51966 **DESCRIPTION**

51967 The *pthread_mutexattr_destroy()* function shall destroy a mutex attributes object; the object
 51968 becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy()* to
 51969 set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be
 51970 reinitialized using *pthread_mutexattr_init()*; the results of otherwise referencing the object after it
 51971 has been destroyed are undefined.

51972 The *pthread_mutexattr_init()* function shall initialize a mutex attributes object *attr* with the
 51973 default value for all of the attributes defined by the implementation.

51974 Results are undefined if *pthread_mutexattr_init()* is called specifying an already initialized *attr*
 51975 attributes object.

51976 After a mutex attributes object has been used to initialize one or more mutexes, any function
 51977 affecting the attributes object (including destruction) shall not affect any previously initialized
 51978 mutexes.

51979 **RETURN VALUE**

51980 Upon successful completion, *pthread_mutexattr_destroy()* and *pthread_mutexattr_init()* shall
 51981 return zero; otherwise, an error number shall be returned to indicate the error.

51982 **ERRORS**

51983 The *pthread_mutexattr_destroy()* function may fail if:

51984 [EINVAL] The value specified by *attr* is invalid.

51985 The *pthread_mutexattr_init()* function shall fail if:

51986 [ENOMEM] Insufficient memory exists to initialize the mutex attributes object.

51987 These functions shall not return an error code of [EINTR].

51988 **EXAMPLES**

51989 None.

51990 **APPLICATION USAGE**

51991 None.

51992 **RATIONALE**

51993 See *pthread_attr_destroy()* for a general explanation of attributes. Attributes objects allow
 51994 implementations to experiment with useful extensions and permit extension of this volume of
 51995 POSIX.1-200x without changing the existing functions. Thus, they provide for future
 51996 extensibility of this volume of POSIX.1-200x and reduce the temptation to standardize
 51997 prematurely on semantics that are not yet widely implemented or understood.

51998 Examples of possible additional mutex attributes that have been discussed are *spin_only*,
 51999 *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean:
 52000 recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes
 52001 would transparently keep records of queue length, wait time, and so on.) Since there is not yet
 52002 wide agreement on the usefulness of these resulting from shared implementation and usage

52003 experience, they are not yet specified in this volume of POSIX.1-200x. Mutex attributes objects,
52004 however, make it possible to test out these concepts for possible standardization at a later time.

52005 **Mutex Attributes and Performance**

52006 Care has been taken to ensure that the default values of the mutex attributes have been defined
52007 such that mutexes initialized with the defaults have simple enough semantics so that the locking
52008 and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few
52009 other basic instructions).

52010 There is at least one implementation method that can be used to reduce the cost of testing at
52011 lock-time if a mutex has non-default attributes. One such method that an implementation can
52012 employ (and this can be made fully transparent to fully conforming POSIX applications) is to
52013 secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to
52014 lock such a mutex causes the implementation to branch to the “slow path” as if the mutex were
52015 unavailable; then, on the slow path, the implementation can do the “real work” to lock a non-
52016 default mutex. The underlying unlock operation is more complicated since the implementation
52017 never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending
52018 on the hardware, there may be certain optimizations that can be used so that whatever mutex
52019 attributes are considered “most frequently used” can be processed most efficiently.

52020 **Process Shared Memory and Synchronization**

52021 The existence of memory mapping functions in this volume of POSIX.1-200x leads to the
52022 possibility that an application may allocate the synchronization objects from this section in
52023 memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

52024 In order to permit such usage, while at the same time keeping the usual case (that is, usage
52025 within a single process) efficient, a *process-shared* option has been defined.

52026 If an implementation supports the `_POSIX_THREAD_PROCESS_SHARED` option, then the
52027 *process-shared* attribute can be used to indicate that mutexes or condition variables may be
52028 accessed by threads of multiple processes.

52029 The default setting of `PTHREAD_PROCESS_PRIVATE` has been chosen for the *process-shared*
52030 attribute so that the most efficient forms of these synchronization objects are created by default.

52031 Synchronization variables that are initialized with the `PTHREAD_PROCESS_PRIVATE` *process-*
52032 *shared* attribute may only be operated on by threads in the process that initialized them.
52033 Synchronization variables that are initialized with the `PTHREAD_PROCESS_SHARED` *process-*
52034 *shared* attribute may be operated on by any thread in any process that has access to it. In
52035 particular, these processes may exist beyond the lifetime of the initializing process. For example,
52036 the following code implements a simple counting semaphore in a mapped file that may be used
52037 by many processes.

```
52038 /* sem.h */
52039 struct semaphore {
52040     pthread_mutex_t lock;
52041     pthread_cond_t nonzero;
52042     unsigned count;
52043 };
52044 typedef struct semaphore semaphore_t;

52045 semaphore_t *semaphore_create(char *semaphore_name);
52046 semaphore_t *semaphore_open(char *semaphore_name);
52047 void semaphore_post(semaphore_t *semap);
52048 void semaphore_wait(semaphore_t *semap);
52049 void semaphore_close(semaphore_t *semap);
```

pthread_mutexattr_destroy()

```

52050     /* sem.c */
52051     #include <sys/types.h>
52052     #include <sys/stat.h>
52053     #include <sys/mman.h>
52054     #include <fcntl.h>
52055     #include <pthread.h>
52056     #include "sem.h"

52057     semaphore_t *
52058     semaphore_create(char *semaphore_name)
52059     {
52060     int fd;
52061         semaphore_t *semap;
52062         pthread_mutexattr_t psharedm;
52063         pthread_condattr_t psharedc;

52064         fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
52065         if (fd < 0)
52066             return (NULL);
52067         (void) ftruncate(fd, sizeof(semaphore_t));
52068         (void) pthread_mutexattr_init(&psharedm);
52069         (void) pthread_mutexattr_setpshared(&psharedm,
52070             PTHREAD_PROCESS_SHARED);
52071         (void) pthread_condattr_init(&psharedc);
52072         (void) pthread_condattr_setpshared(&psharedc,
52073             PTHREAD_PROCESS_SHARED);
52074         semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
52075             PROT_READ | PROT_WRITE, MAP_SHARED,
52076             fd, 0);
52077         close (fd);
52078         (void) pthread_mutex_init(&semap->lock, &psharedm);
52079         (void) pthread_cond_init(&semap->nonzero, &psharedc);
52080         semap->count = 0;
52081         return (semap);
52082     }

52083     semaphore_t *
52084     semaphore_open(char *semaphore_name)
52085     {
52086     int fd;
52087         semaphore_t *semap;

52088         fd = open(semaphore_name, O_RDWR, 0666);
52089         if (fd < 0)
52090             return (NULL);
52091         semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
52092             PROT_READ | PROT_WRITE, MAP_SHARED,
52093             fd, 0);
52094         close (fd);
52095         return (semap);
52096     }

52097     void
52098     semaphore_post(semaphore_t *semap)
52099     {
52100         pthread_mutex_lock(&semap->lock);
52101         if (semap->count == 0)

```

```

52102         pthread_cond_signal(&semapx->nonzero);
52103         semap->count++;
52104         pthread_mutex_unlock(&semap->lock);
52105     }
52106
52107 void
52108 semaphore_wait(semaphore_t *semap)
52109 {
52110     pthread_mutex_lock(&semap->lock);
52111     while (semap->count == 0)
52112         pthread_cond_wait(&semap->nonzero, &semap->lock);
52113     semap->count--;
52114     pthread_mutex_unlock(&semap->lock);
52115 }

```

```

52115 void
52116 semaphore_close(semaphore_t *semap)
52117 {
52118     munmap((void *) semap, sizeof(semaphore_t));
52119 }

```

52120 The following code is for three separate processes that create, post, and wait on a semaphore in
52121 the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and
52122 decrement the counting semaphore (waiting and waking as required) even though they did not
52123 initialize the semaphore.

```

52124 /* create.c */
52125 #include "pthread.h"
52126 #include "sem.h"
52127
52128 int
52129 main()
52130 {
52131     semaphore_t *semap;
52132     semap = semaphore_create("/tmp/semaphore");
52133     if (semap == NULL)
52134         exit(1);
52135     semaphore_close(semap);
52136     return (0);
52137 }
52138
52139 /* post.c */
52140 #include "pthread.h"
52141 #include "sem.h"
52142
52143 int
52144 main()
52145 {
52146     semaphore_t *semap;
52147
52148     semap = semaphore_open("/tmp/semaphore");
52149     if (semap == NULL)
52150         exit(1);
52151     semaphore_post(semap);
52152     semaphore_close(semap);
52153     return (0);
52154 }

```

pthread_mutexattr_destroy()

```

52151     /* wait */
52152     #include "pthread.h"
52153     #include "sem.h"

52154     int
52155     main()
52156     {
52157         semaphore_t *semap;

52158         semap = semaphore_open("/tmp/semaphore");
52159         if (semap == NULL)
52160             exit(1);
52161         semaphore_wait(semap);
52162         semaphore_close(semap);
52163         return (0);
52164     }

```

FUTURE DIRECTIONS

None.

SEE ALSO

[*pthread_cond_destroy\(\)*](#), [*pthread_create\(\)*](#), [*pthread_mutex_destroy\(\)*](#), [*pthread_mutexattr_destroy\(\)*](#)

XBD [**<pthread.h>**](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

Issue 6

The [*pthread_mutexattr_destroy\(\)*](#) and [*pthread_mutexattr_init\(\)*](#) functions are marked as part of the Threads option.

IEEE PASC Interpretation 1003.1c #27 is applied, updating the ERRORS section.

Issue 7

The [*pthread_mutexattr_destroy\(\)*](#) and [*pthread_mutexattr_init\(\)*](#) functions are moved from the Threads option to the Base.

52179 **NAME**

52180 pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the
 52181 prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

52182 **SYNOPSIS**

```
52183 RPP|TPP #include <pthread.h>
52184
52184 int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
52185 *restrict attr, int *restrict prioceiling);
52186 int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
52187 int prioceiling);
```

52188 **DESCRIPTION**

52189 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions,
 52190 respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to
 52191 by *attr* which was previously created by the function *pthread_mutexattr_init()*.

52192 The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of
 52193 *prioceiling* are within the maximum range of priorities defined by SCHED_FIFO.

52194 The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum
 52195 priority level at which the critical section guarded by the mutex is executed. In order to avoid
 52196 priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal
 52197 to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are
 52198 within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.

52199 **RETURN VALUE**

52200 Upon successful completion, the *pthread_mutexattr_getprioceiling()* and
 52201 *pthread_mutexattr_setprioceiling()* functions shall return zero; otherwise, an error number shall be
 52202 returned to indicate the error.

52203 **ERRORS**

52204 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions may fail if:

52205 [EINVAL] The value specified by *attr* or *prioceiling* is invalid.

52206 [EPERM] The caller does not have the privilege to perform the operation.

52207 These functions shall not return an error code of [EINTR].

52208 **EXAMPLES**

52209 None.

52210 **APPLICATION USAGE**

52211 None.

52212 **RATIONALE**

52213 None.

52214 **FUTURE DIRECTIONS**

52215 None.

52216 **SEE ALSO**

52217 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*

52218 XBD [<pthread.h>](#)

52219 **CHANGE HISTORY**

52220 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52221 Marked as part of the Realtime Threads Feature Group.

52222 **Issue 6**52223 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are marked
52224 as part of the Threads and Thread Priority Protection options.52225 The [ENOSYS] error condition has been removed as stubs need not be provided if an
52226 implementation does not support the Thread Priority Protection option.52227 The [ENOTSUP] error condition has been removed since these functions do not have a *protocol*
52228 argument.52229 The **restrict** keyword is added to the *pthread_mutexattr_getprioceiling()* prototype for alignment
52230 with the ISO/IEC 9899:1999 standard.52231 **Issue 7**52232 The *pthread_mutexattr_getprioceiling()* and *pthread_mutexattr_setprioceiling()* functions are moved
52233 from the Threads option to require support of either the Robust Mutex Priority Protection option
52234 or the Non-Robust Mutex Priority Protection option.

52235 **NAME**

52236 pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol
 52237 attribute of the mutex attributes object (**REALTIME THREADS**)

52238 **SYNOPSIS**

```
52239 MC1 #include <pthread.h>
52240
52241 int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
52242 *restrict attr, int *restrict protocol);
52243 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
52244 int protocol);
```

52244 **DESCRIPTION**

52245 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions, respectively,
 52246 shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was
 52247 previously created by the function *pthread_mutexattr_init()*.

52248 The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of
 52249 *protocol* may be one of:

```
52250 RPI | TPI PTHREAD_PRIO_INHERIT
52251 MC1 PTHREAD_PRIO_NONE
52252 RPP | TPP PTHREAD_PRIO_PROTECT
```

52253 which are defined in the **<pthread.h>** header. The default value of the attribute shall be
 52254 PTHREAD_PRIO_NONE.

52255 When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority
 52256 and scheduling shall not be affected by its mutex ownership.

52257 RPI When a thread is blocking higher priority threads because of owning one or more robust
 52258 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 52259 its priority or the priority of the highest priority thread waiting on any of the robust mutexes
 52260 owned by this thread and initialized with this protocol.

52261 TPI When a thread is blocking higher priority threads because of owning one or more non-robust
 52262 mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of
 52263 its priority or the priority of the highest priority thread waiting on any of the non-robust
 52264 mutexes owned by this thread and initialized with this protocol.

52265 RPP When a thread owns one or more robust mutexes initialized with the
 52266 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 52267 of the priority ceilings of all the robust mutexes owned by this thread and initialized with this
 52268 attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

52269 TPP When a thread owns one or more non-robust mutexes initialized with the
 52270 PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest
 52271 of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with
 52272 this attribute, regardless of whether other threads are blocked on any of these non-robust
 52273 mutexes or not.

52274 While a thread is holding a mutex which has been initialized with the
 52275 PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be
 52276 subject to being moved to the tail of the scheduling queue at its priority in the event that its
 52277 original priority is changed, such as by a call to *sched_setparam()*. Likewise, when a thread

pthread_mutexattr_getprotocol()

System Interfaces

52278 unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or
 52279 PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail
 52280 of the scheduling queue at its priority in the event that its original priority is changed.

52281 If a thread simultaneously owns several mutexes initialized with different protocols, it shall
 52282 execute at the highest of the priorities that it would have obtained by each of these protocols.

52283 RPI | TPI When a thread makes a call to *pthread_mutex_lock()*, the mutex was initialized with the protocol
 52284 attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked
 52285 because the mutex is owned by another thread, that owner thread shall inherit the priority level
 52286 of the calling thread as long as it continues to own the mutex. The implementation shall update
 52287 its execution priority to the maximum of its assigned priority and all its inherited priorities.
 52288 Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol*
 52289 attribute having the value PTHREAD_PRIO_INHERIT, the same priority inheritance effect shall
 52290 be propagated to this other owner thread, in a recursive manner.

RETURN VALUE

52291 Upon successful completion, the *pthread_mutexattr_getprotocol()* and
 52292 *pthread_mutexattr_setprotocol()* functions shall return zero; otherwise, an error number shall be
 52293 returned to indicate the error.
 52294

ERRORS

52295 The *pthread_mutexattr_setprotocol()* function shall fail if:

52296 [ENOTSUP] The value specified by *protocol* is an unsupported value.

52297 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions may fail if:

52298 [EINVAL] The value specified by *attr* or *protocol* is invalid.

52299 [EPERM] The caller does not have the privilege to perform the operation.

52300 These functions shall not return an error code of [EINTR].
 52301

EXAMPLES

52302 None.
 52303

APPLICATION USAGE

52304 None.
 52305

RATIONALE

52306 None.
 52307

FUTURE DIRECTIONS

52308 None.
 52309

SEE ALSO

52310 *pthread_cond_destroy()*, *pthread_create()*, *pthread_mutex_destroy()*

52311 XBD <pthread.h>
 52312

CHANGE HISTORY

52313 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

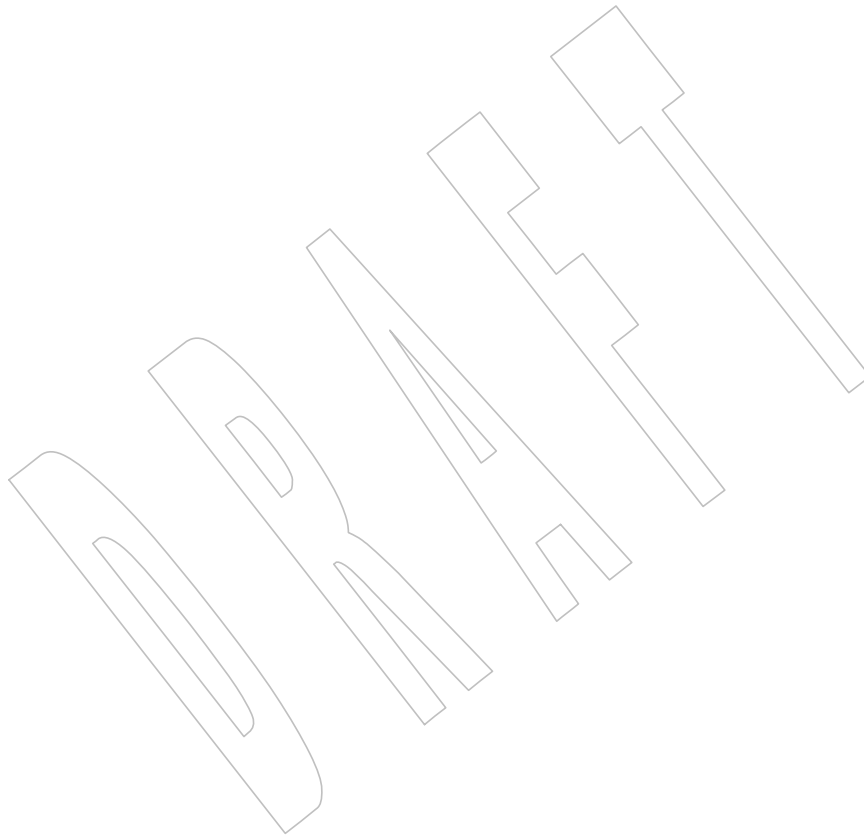
52314 Marked as part of the Realtime Threads Feature Group.
 52315

Issue 6

52316 The *pthread_mutexattr_getprotocol()* and *pthread_mutexattr_setprotocol()* functions are marked as
 52317 part of the Threads option and either the Thread Priority Protection or Thread Priority
 52318 Inheritance options.
 52319

52320 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 52321 implementation does not support the Thread Priority Protection or Thread Priority Inheritance
 52322 options.

52323	The restrict keyword is added to the <i>pthread_mutexattr_getprotocol()</i> prototype for alignment	
52324	with the ISO/IEC 9899:1999 standard.	
52325	Issue 7	
52326	SD5-XSH-ERN-135 is applied, updating the DESCRIPTION to define a default value for the	+
52327	<i>protocol</i> attribute.	+
52328	SD5-XSH-ERN-188 is applied, updating the DESCRIPTION.	+
52329	The <i>pthread_mutexattr_getprotocol()</i> and <i>pthread_mutexattr_setprotocol()</i> functions are moved from	
52330	the Threads option to require support of either the Non-Robust Mutex Priority Protection option	
52331	or the Non-Robust Mutex Priority Inheritance option or the Robust Mutex Priority Protection	
52332	option or the Robust Mutex Priority Inheritance option.	-



52333 **NAME**

52334 pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-shared
 52335 attribute

52336 **SYNOPSIS**

```
52337 TSH #include <pthread.h>
52338
52338 int pthread_mutexattr_getpshared(const pthread_mutexattr_t
52339 *restrict attr, int *restrict pshared);
52340 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
52341 int pshared);
```

52342 **DESCRIPTION**

52343 The *pthread_mutexattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 52344 from the attributes object referenced by *attr*. The *pthread_mutexattr_setpshared()* function shall
 52345 set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

52346 The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a mutex to be
 52347 operated upon by any thread that has access to the memory where the mutex is allocated, even if
 52348 the mutex is allocated in memory that is shared by multiple processes. If the *process-shared*
 52349 attribute is PTHREAD_PROCESS_PRIVATE, the mutex shall only be operated upon by threads
 52350 created within the same process as the thread that initialized the mutex; if threads of differing
 52351 processes attempt to operate on such a mutex, the behavior is undefined. The default value of
 52352 the attribute shall be PTHREAD_PROCESS_PRIVATE.

52353 **RETURN VALUE**

52354 Upon successful completion, *pthread_mutexattr_setpshared()* shall return zero; otherwise, an error
 52355 number shall be returned to indicate the error.

52356 Upon successful completion, *pthread_mutexattr_getpshared()* shall return zero and store the value
 52357 of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter.
 52358 Otherwise, an error number shall be returned to indicate the error.

52359 **ERRORS**

52360 The *pthread_mutexattr_getpshared()* and *pthread_mutexattr_setpshared()* functions may fail if:

52361 [EINVAL] The value specified by *attr* is invalid.

52362 The *pthread_mutexattr_setpshared()* function may fail if:

52363 [EINVAL] The new value specified for the attribute is outside the range of legal values
 52364 for that attribute.

52365 These functions shall not return an error code of [EINTR].

52366 **EXAMPLES**

52367 None.

52368 **APPLICATION USAGE**

52369 None.

52370 **RATIONALE**

52371 None.

52372
52373
52374
52375
52376
52377
52378
52379
52380
52381
52382
52383
52384
52385
52386

FUTURE DIRECTIONS

None.

SEE ALSO

[pthread_cond_destroy\(\)](#), [pthread_create\(\)](#), [pthread_mutex_destroy\(\)](#), [pthread_mutexattr_destroy\(\)](#)

XBD [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Threads Extension.

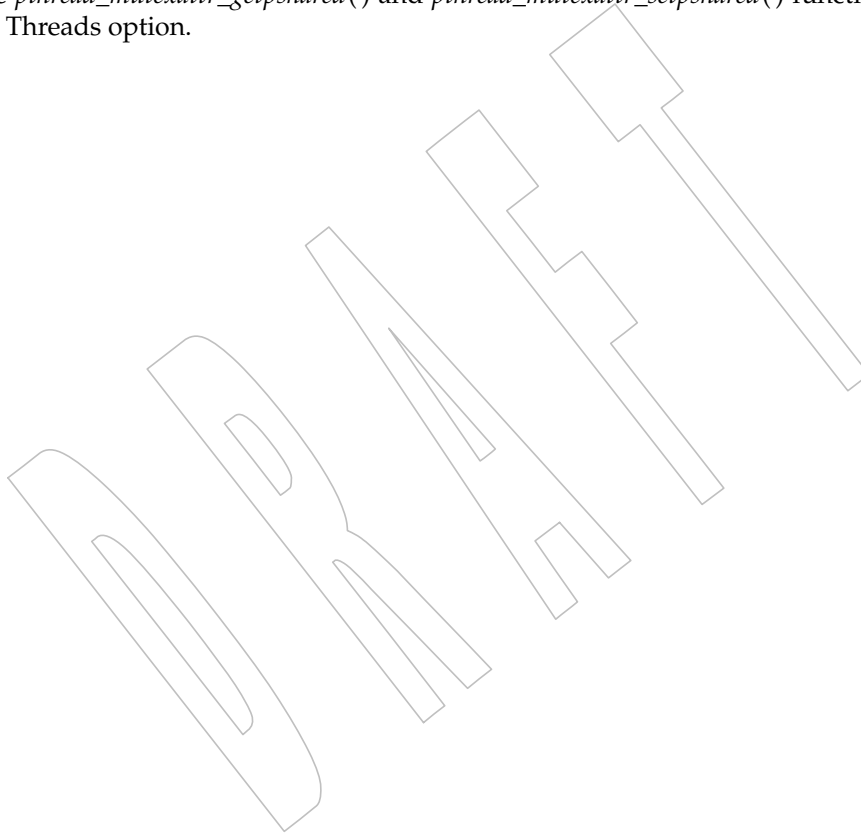
Issue 6

The [pthread_mutexattr_getpshared\(\)](#) and [pthread_mutexattr_setpshared\(\)](#) functions are marked as part of the Threads and Thread Process-Shared Synchronization options.

The **restrict** keyword is added to the [pthread_mutexattr_getpshared\(\)](#) prototype for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

The [pthread_mutexattr_getpshared\(\)](#) and [pthread_mutexattr_setpshared\(\)](#) functions are moved from the Threads option.



52387 **NAME**

52388 pthread_mutexattr_getrobust, pthread_mutexattr_setrobust — get and set the mutex robust
 52389 attribute

52390 **SYNOPSIS**

```
52391 #include <pthread.h>

52392 int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
52393     attr, int *restrict robust);
52394 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
52395     int robust);
```

52396 **DESCRIPTION**

52397 The *pthread_mutexattr_getrobust()* and *pthread_mutexattr_setrobust()* functions, respectively, shall
 52398 get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values
 52399 for *robust* include:

52400 **PTHREAD_MUTEX_STALLED**

52401 No special actions are taken if the owner of the mutex is terminated while holding the
 52402 mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.
 52403 This is the default value.

52404 **PTHREAD_MUTEX_ROBUST**

52405 If the process containing the owning thread of a robust mutex terminates while holding the
 52406 mutex lock, the next thread that acquires the mutex shall be notified about the termination
 52407 by the return value [EOWNERDEAD] from the locking function. If the owning thread of a
 52408 robust mutex terminates while holding the mutex lock, the next thread that acquires the
 52409 mutex may be notified about the termination by the return value [EOWNERDEAD]. The
 52410 notified thread can then attempt to mark the state protected by the mutex as consistent
 52411 again by a call to *pthread_mutex_consistent()*. After a subsequent successful call to
 52412 *pthread_mutex_unlock()*, the mutex lock shall be released and can be used normally by other
 52413 threads. If the mutex is unlocked without a call to *pthread_mutex_consistent()*, it shall be in a
 52414 permanently unusable state and all attempts to lock the mutex shall fail with the error
 52415 [ENOTRECOVERABLE]. The only permissible operation on such a mutex is
 52416 *pthread_mutex_destroy()*.

52417 **RETURN VALUE**

52418 Upon successful completion, the *pthread_mutexattr_getrobust()* function shall return zero and
 52419 store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter.
 52420 Otherwise, an error value shall be returned to indicate the error. If successful, the
 52421 *pthread_mutexattr_setrobust()* function shall return zero; otherwise, an error number shall be
 52422 returned to indicate the error.

52423 **ERRORS**

52424 The *pthread_mutexattr_setrobust()* function shall fail if:

52425 [EINVAL] The value of *robust* is invalid.

52426 The *pthread_mutexattr_getrobust()* and *pthread_mutexattr_setrobust()* functions may fail if:

52427 [EINVAL] The value specified by *attr* is invalid.

52428 These functions shall not return an error code of [EINTR].

52429

EXAMPLES

52430

None.

52431

APPLICATION USAGE

52432

The actions required to make the state protected by the mutex consistent again are solely dependent on the application. If it is not possible to make the state of a mutex consistent, robust mutexes can be used to notify this situation by calling *pthread_mutex_unlock()* without a prior call to *pthread_mutex_consistent()*.

52433

52434

52435

52436

If the state is declared inconsistent by calling *pthread_mutex_unlock()* without a prior call to *pthread_mutex_consistent()*, a possible approach could be to destroy the mutex and then reinitialize it. However, it should be noted that this is possible only in certain situations where the state protected by the mutex has to be reinitialized and coordination achieved with other threads blocked on the mutex, because otherwise a call to a locking function with a reference to a mutex object invalidated by a call to *pthread_mutex_destroy()* results in undefined behavior.

52437

52438

52439

52440

52441

52442

RATIONALE

52443

None.

52444

FUTURE DIRECTIONS

52445

None.

52446

SEE ALSO

52447

pthread_mutex_consistent(), *pthread_mutex_destroy()*, *pthread_mutex_lock()*

52448

XBD <pthread.h>

52449

CHANGE HISTORY

52450

First released in Issue 7.

52451

The *pthread_mutexattr_getrobust()* and *pthread_mutexattr_setrobust()* functions are moved from the Threads option to the Base.

52452

52453 **NAME**

52454 pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

52455 **SYNOPSIS**

52456 #include <pthread.h>

```
52457 int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict attr,
52458 int *restrict type);
52459 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

52460 **DESCRIPTION**

52461 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions, respectively, shall get
 52462 and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The
 52463 default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

52464 The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types
 52465 include:

52466 **PTHREAD_MUTEX_NORMAL**

52467 This type of mutex does not detect deadlock. A thread attempting to relock this mutex
 52468 without first unlocking it shall deadlock. Attempting to unlock a mutex locked by a
 52469 different thread results in undefined behavior. Attempting to unlock an unlocked mutex
 52470 results in undefined behavior.

52471 **PTHREAD_MUTEX_ERRORCHECK**

52472 This type of mutex provides error checking. A thread attempting to relock this mutex
 52473 without first unlocking it shall return with an error. A thread attempting to unlock a mutex
 52474 which another thread has locked shall return with an error. A thread attempting to unlock
 52475 an unlocked mutex shall return with an error.

52476 **PTHREAD_MUTEX_RECURSIVE**

52477 A thread attempting to relock this mutex without first unlocking it shall succeed in locking
 52478 the mutex. The relocking deadlock which can occur with mutexes of type
 52479 PTHREAD_MUTEX_NORMAL cannot occur with this type of mutex. Multiple locks of this
 52480 mutex shall require the same number of unlocks to release the mutex before another thread
 52481 can acquire the mutex. A thread attempting to unlock a mutex which another thread has
 52482 locked shall return with an error. A thread attempting to unlock an unlocked mutex shall
 52483 return with an error.

52484 **PTHREAD_MUTEX_DEFAULT**

52485 Attempting to recursively lock a mutex of this type results in undefined behavior.
 52486 Attempting to unlock a mutex of this type which was not locked by the calling thread
 52487 results in undefined behavior. Attempting to unlock a mutex of this type which is not
 52488 locked results in undefined behavior. An implementation may map this mutex to one of the
 52489 other mutex types.

52490 **RETURN VALUE**

52491 Upon successful completion, the *pthread_mutexattr_gettype()* function shall return zero and store
 52492 the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise,
 52493 an error shall be returned to indicate the error.

52494 If successful, the *pthread_mutexattr_settype()* function shall return zero; otherwise, an error
 52495 number shall be returned to indicate the error.

52496 ERRORS

52497 The *pthread_mutexattr_settype()* function shall fail if:

52498 [EINVAL] The value *type* is invalid.

52499 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions may fail if:

52500 [EINVAL] The value specified by *attr* is invalid.

52501 These functions shall not return an error code of [EINTR].

52502 EXAMPLES

52503 None.

52504 APPLICATION USAGE

52505 It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with
 52506 condition variables because the implicit unlock performed for a *pthread_cond_timedwait()* or
 52507 *pthread_cond_wait()* may not actually release the mutex (if it had been locked multiple times). If
 52508 this happens, no other thread can satisfy the condition of the predicate.

52509 RATIONALE

52510 None.

52511 FUTURE DIRECTIONS

52512 None.

52513 SEE ALSO

52514 [*pthread_cond_timedwait\(\)*](#)

52515 XBD <[pthread.h](#)>

52516 CHANGE HISTORY

52517 First released in Issue 5.

52518 Issue 6

52519 The Open Group Corrigendum U033/3 is applied. The SYNOPSIS for
 52520 *pthread_mutexattr_gettype()* is updated so that the first argument is of type **const**
 52521 **pthread_mutexattr_t***.

52522 The **restrict** keyword is added to the *pthread_mutexattr_gettype()* prototype for alignment with
 52523 the ISO/IEC 9899:1999 standard.

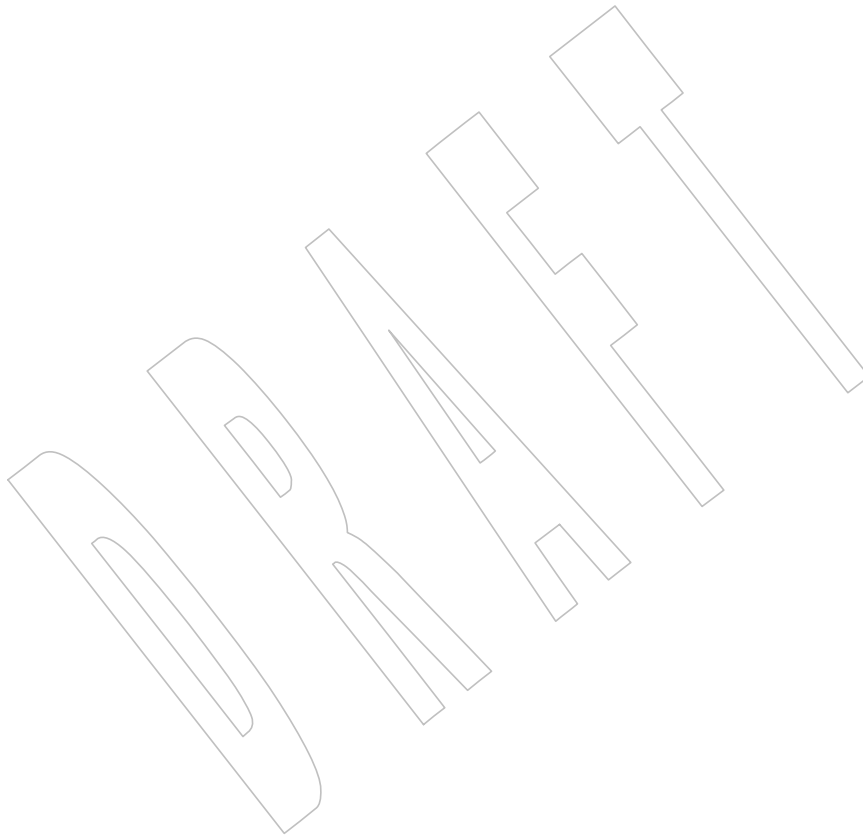
52524 Issue 7

52525 The *pthread_mutexattr_gettype()* and *pthread_mutexattr_settype()* functions are moved from the
 52526 XSI option to the Base.

52527 **NAME**
52528 pthread_mutexattr_init — initialize the mutex attributes object

52529 **SYNOPSIS**
52530 #include <pthread.h>
52531 int pthread_mutexattr_init(pthread_mutexattr_t *attr);

52532 **DESCRIPTION**
52533 Refer to *pthread_mutexattr_destroy()*.



52534

NAME

52535

pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object
(**REALTIME THREADS**)

52536

52537

SYNOPSIS

52538

```
RPP|TPP #include <pthread.h>
```

52539

```
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,  
int prioceiling);
```

52540

52541

DESCRIPTION

52542

Refer to [pthread_mutexattr_getprioceiling\(\)](#).

pthread_mutexattr_setprotocol()*System Interfaces*52543 **NAME**

52544 pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object
52545 (**REALTIME THREADS**)

52546 **SYNOPSIS**

```
52547 MC1 #include <pthread.h>  
52548 int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,  
52549 int protocol);
```

52550 **DESCRIPTION**

52551 Refer to [pthread_mutexattr_getprotocol\(\)](#).

52552 **NAME**

52553 pthread_mutexattr_setpshared — set the process-shared attribute

52554 **SYNOPSIS**

```
52555 TSH #include <pthread.h>
52556 int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
52557 int pshared);
```

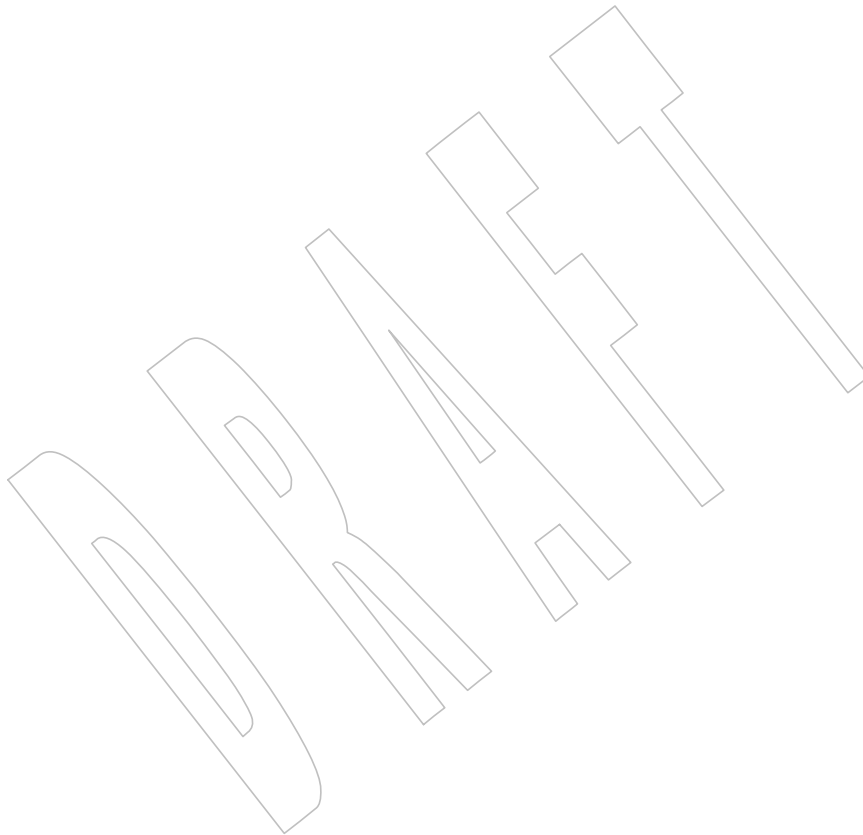
52558 **DESCRIPTION**52559 Refer to *pthread_mutexattr_getpshared()*.

pthread_mutexattr_setrobust()

52560 **NAME**
52561 pthread_mutexattr_setrobust — get and set the mutex robust attribute

52562 **SYNOPSIS**
52563 #include <pthread.h>
52564 int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
52565 int robust);

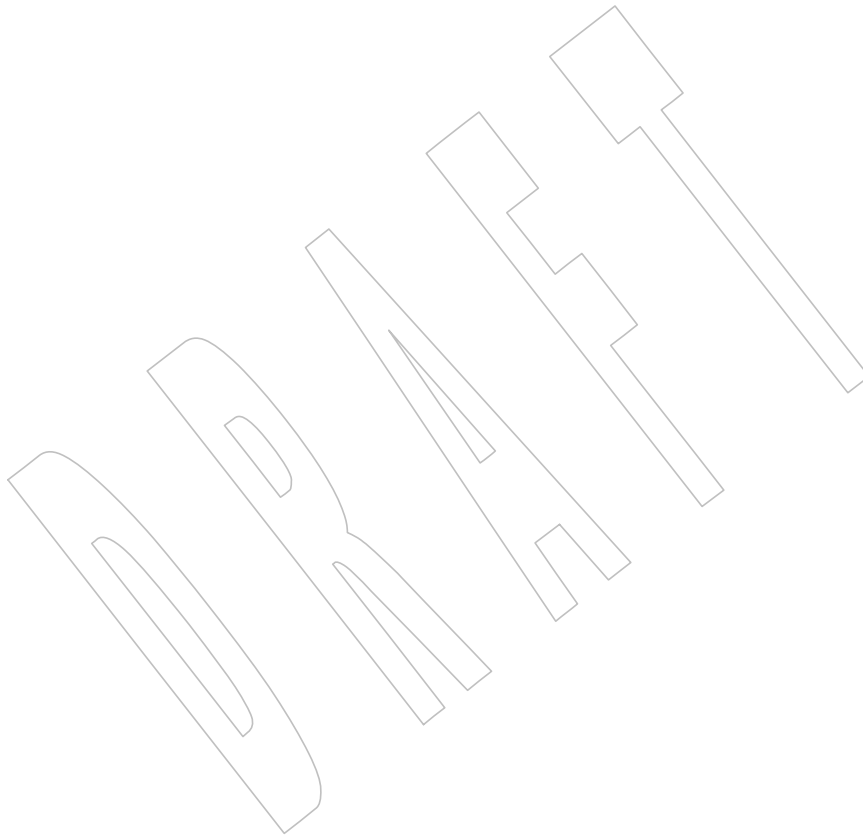
52566 **DESCRIPTION**
52567 Refer to [pthread_mutexattr_getrobust\(\)](#).



52568 **NAME**
52569 pthread_mutexattr_settype — set the mutex type attribute

52570 **SYNOPSIS**
52571 #include <pthread.h>
52572 int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type); |

52573 **DESCRIPTION**
52574 Refer to *pthread_mutexattr_gettype()*.



52575 **NAME**
 52576 pthread_once — dynamic package initialization

52577 **SYNOPSIS**
 52578 #include <pthread.h>
 52579 int pthread_once(pthread_once_t *once_control,
 52580 void (*init_routine)(void));
 52581 pthread_once_t once_control = PTHREAD_ONCE_INIT;

52582 **DESCRIPTION**
 52583 The first call to *pthread_once()* by any thread in a process, with a given *once_control*, shall call the
 52584 *init_routine* with no arguments. Subsequent calls of *pthread_once()* with the same *once_control*
 52585 shall not call the *init_routine*. On return from *pthread_once()*, *init_routine* shall have completed.
 52586 The *once_control* parameter shall determine whether the associated initialization routine has been
 52587 called.

52588 The *pthread_once()* function is not a cancellation point. However, if *init_routine* is a cancellation
 52589 point and is canceled, the effect on *once_control* shall be as if *pthread_once()* was never called.

52590 The constant PTHREAD_ONCE_INIT is defined in the <pthread.h> header.

52591 The behavior of *pthread_once()* is undefined if *once_control* has automatic storage duration or is
 52592 not initialized by PTHREAD_ONCE_INIT.

52593 **RETURN VALUE**
 52594 Upon successful completion, *pthread_once()* shall return zero; otherwise, an error number shall
 52595 be returned to indicate the error.

52596 **ERRORS**
 52597 The *pthread_once()* function may fail if:
 52598 [EINVAL] If either *once_control* or *init_routine* is invalid.
 52599 The *pthread_once()* function shall not return an error code of [EINTR].

52600 **EXAMPLES**
 52601 None.

52602 **APPLICATION USAGE**
 52603 None.

52604 **RATIONALE**
 52605 Some C libraries are designed for dynamic initialization. That is, the global initialization for the
 52606 library is performed when the first procedure in the library is called. In a single-threaded
 52607 program, this is normally implemented using a static variable whose value is checked on entry
 52608 to a routine, as follows:

```
52609 static int random_is_initialized = 0;
52610 extern int initialize_random();

52611 int random_function()
52612 {
52613     if (random_is_initialized == 0) {
52614         initialize_random();
52615         random_is_initialized = 1;
52616     }
52617     ... /* Operations performed after initialization. */
52618 }
```

52619 To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise,
 52620 library initialization has to be accomplished by an explicit call to a library-exported initialization
 52621 function prior to any use of the library.

52622 For dynamic library initialization in a multi-threaded process, a simple initialization flag is not
 52623 sufficient; the flag needs to be protected against modification by multiple threads
 52624 simultaneously calling into the library. Protecting the flag requires the use of a mutex; however,
 52625 mutexes have to be initialized before they are used. Ensuring that the mutex is only initialized
 52626 once requires a recursive solution to this problem.

52627 The use of *pthread_once()* not only supplies an implementation-guaranteed means of dynamic
 52628 initialization, it provides an aid to the reliable construction of multi-threaded and realtime
 52629 systems. The preceding example then becomes:

```
52630 #include <pthread.h>
52631 static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
52632 extern int initialize_random();

52633 int random_function()
52634 {
52635     (void) pthread_once(&random_is_initialized, initialize_random);
52636     ... /* Operations performed after initialization. */
52637 }
```

52638 Note that a **pthread_once_t** cannot be an array because some compilers do not accept the
 52639 construct **&<array_name>**.

52640 FUTURE DIRECTIONS

52641 None.

52642 SEE ALSO

52643 XBD [<pthread.h>](#)

52644 CHANGE HISTORY

52645 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

52646 Issue 6

52647 The *pthread_once()* function is marked as part of the Threads option.

52648 The [EINVAL] error is added as a “may fail” case for if either argument is invalid.

52649 Issue 7

52650 The *pthread_once()* function is moved from the Threads option to the Base.

52651 **NAME**

52652 pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

52653 **SYNOPSIS**

52654 #include <pthread.h>

```
52655 int pthread_rwlock_destroy(pthread_rwlock_t *rwlck,
52656 int pthread_rwlock_init(pthread_rwlock_t *restrict rwlck,
52657 const pthread_rwlockattr_t *restrict attr);
```

52658 XSI pthread_rwlock_t rwlck = PTHREAD_RWLOCK_INITIALIZER;

52659 **DESCRIPTION**

52660 The *pthread_rwlock_destroy()* function shall destroy the read-write lock object referenced by
 52661 *rwlck* and release any resources used by the lock. The effect of subsequent use of the lock is
 52662 undefined until the lock is reinitialized by another call to *pthread_rwlock_init()*. An
 52663 implementation may cause *pthread_rwlock_destroy()* to set the object referenced by *rwlck* to an
 52664 invalid value. Results are undefined if *pthread_rwlock_destroy()* is called when any thread holds
 52665 *rwlck*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

52666 The *pthread_rwlock_init()* function shall allocate any resources required to use the read-write lock
 52667 referenced by *rwlck* and initializes the lock to an unlocked state with attributes referenced by
 52668 *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as
 52669 passing the address of a default read-write lock attributes object. Once initialized, the lock can be
 52670 used any number of times without being reinitialized. Results are undefined if
 52671 *pthread_rwlock_init()* is called specifying an already initialized read-write lock. Results are
 52672 undefined if a read-write lock is used without first being initialized.

52673 If the *pthread_rwlock_init()* function fails, *rwlck* shall not be initialized and the contents of *rwlck*
 52674 are undefined.

52675 Only the object referenced by *rwlck* may be used for performing synchronization. The result of
 52676 referring to copies of that object in calls to *pthread_rwlock_destroy()*, *pthread_rwlock_rdlock()*,
 52677 *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_tryrdlock()*,
 52678 *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*, or *pthread_rwlock_wrlock()* is undefined.

52679 XSI In cases where default read-write lock attributes are appropriate, the macro
 52680 PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks that are statically
 52681 allocated. The effect shall be equivalent to dynamic initialization by a call to *pthread_rwlock_init()*
 52682 with the *attr* parameter specified as NULL, except that no error checks are performed.

52683 **RETURN VALUE**

52684 If successful, the *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions shall return zero;
 52685 otherwise, an error number shall be returned to indicate the error.

52686 The [EBUSY] and [EINVAL] error checks, if implemented, act as if they were performed
 52687 immediately at the beginning of processing for the function and caused an error return prior to
 52688 modifying the state of the read-write lock specified by *rwlck*.

52689 **ERRORS**

52690 The *pthread_rwlock_destroy()* function may fail if:

52691 [EBUSY] The implementation has detected an attempt to destroy the object referenced
 52692 by *rwlck* while it is locked.

52693 [EINVAL] The value specified by *rwlck* is invalid.

52694 The *pthread_rwlock_init()* function shall fail if:

- 52695 [EAGAIN] The system lacked the necessary resources (other than memory) to initialize
52696 another read-write lock.
- 52697 [ENOMEM] Insufficient memory exists to initialize the read-write lock.
- 52698 [EPERM] The caller does not have the privilege to perform the operation.
- 52699 The *pthread_rwlock_init()* function may fail if:
- 52700 [EBUSY] The implementation has detected an attempt to reinitialize the object
52701 referenced by *rwlock*, a previously initialized but not yet destroyed read-write
52702 lock.
- 52703 [EINVAL] The value specified by *attr* is invalid.
- 52704 These functions shall not return an error code of [EINTR].

EXAMPLES

None.

APPLICATION USAGE

Applications using these and related read-write lock functions may be subject to priority inversion, as discussed in XBD [Section 3.284](#) (on page 72).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[pthread_rwlock_rdlock\(\)](#), [pthread_rwlock_timedrdlock\(\)](#), [pthread_rwlock_timedwrlock\(\)](#),
[pthread_rwlock_trywrlock\(\)](#), [pthread_rwlock_unlock\(\)](#)

XBD [Section 3.284](#) (on page 72), [<pthread.h>](#)

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension). The initializer macro is also deleted from the SYNOPSIS.
- The DESCRIPTION is updated as follows:
 - It explicitly notes allocation of resources upon initialization of a read-write lock object.
 - A paragraph is added specifying that copies of read-write lock objects may not be used.
- An [EINVAL] error is added to the ERRORS section for *pthread_rwlock_init()*, indicating that the *rwlock* value is invalid.
- The SEE ALSO section is updated.

The **restrict** keyword is added to the *pthread_rwlock_init()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/45 is applied, adding APPLICATION USAGE relating to priority inversion.

pthread_rwlock_destroy()*System Interfaces*

52738

Issue 7

52739

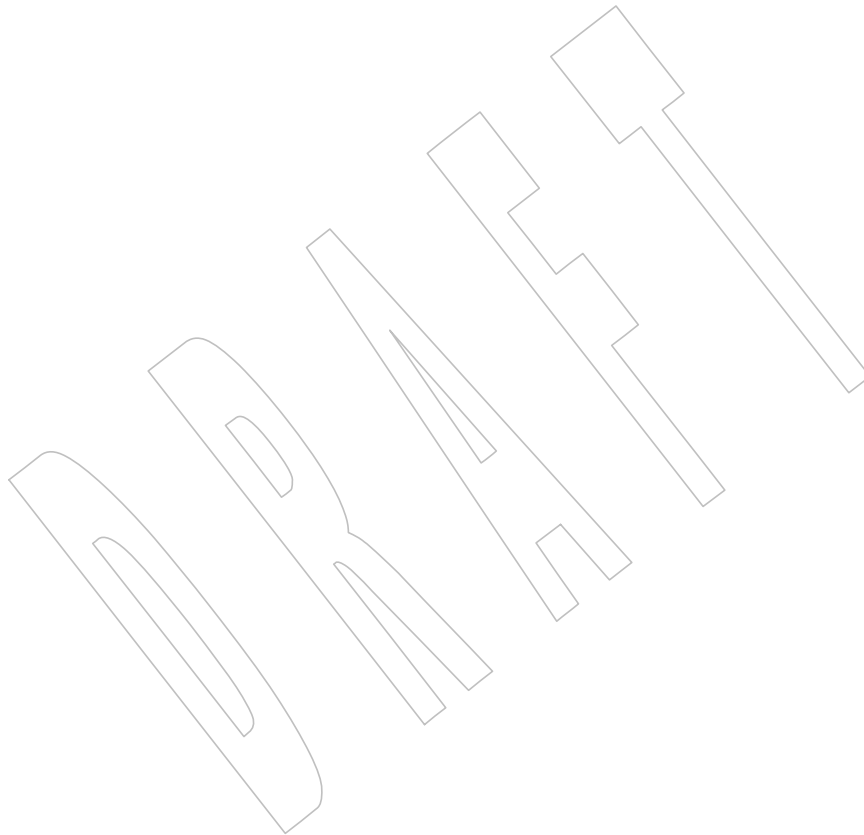
Austin Group Interpretation 1003.1-2001 #048 is applied, adding the PTHREAD_RWLOCK_INITIALIZER macro.

52740

52741

The *pthread_rwlock_destroy()* and *pthread_rwlock_init()* functions are moved from the Threads option to the Base.

52742



52743 **NAME**
 52744 `pthread_rwlock_rdlock, pthread_rwlock_tryrdlock` — lock a read-write lock object for reading

52745 **SYNOPSIS**

52746 `#include <pthread.h>`

52747 `int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);`
 52748 `int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);`

52749 **DESCRIPTION**

52750 The `pthread_rwlock_rdlock()` function shall apply a read lock to the read-write lock referenced by
 52751 `rwlock`. The calling thread acquires the read lock if a writer does not hold the lock and there are
 52752 no writers blocked on the lock.

52753 TPS If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
 52754 executing with the scheduling policies `SCHED_FIFO` or `SCHED_RR`, the calling thread shall not
 52755 acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on
 52756 the lock; otherwise, the calling thread shall acquire the lock.

52757 TPS TSP If the Thread Execution Scheduling option is supported, and the threads involved in the lock are
 52758 executing with the `SCHED_SPORADIC` scheduling policy, the calling thread shall not acquire
 52759 the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock;
 52760 otherwise, the calling thread shall acquire the lock.

52761 If the Thread Execution Scheduling option is not supported, it is implementation-defined
 52762 whether the calling thread acquires the lock when a writer does not hold the lock and there are
 52763 writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the
 52764 read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the
 52765 lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

52766 A thread may hold multiple concurrent read locks on `rwlock` (that is, successfully call the
 52767 `pthread_rwlock_rdlock()` function n times). If so, the application shall ensure that the thread
 52768 performs matching unlocks (that is, it calls the `pthread_rwlock_unlock()` function n times).

52769 The maximum number of simultaneous read locks that an implementation guarantees can be
 52770 applied to a read-write lock shall be implementation-defined. The `pthread_rwlock_rdlock()`
 52771 function may fail if this maximum would be exceeded.

52772 The `pthread_rwlock_tryrdlock()` function shall apply a read lock as in the `pthread_rwlock_rdlock()`
 52773 function, with the exception that the function shall fail if the equivalent `pthread_rwlock_rdlock()`
 52774 call would have blocked the calling thread. In no case shall the `pthread_rwlock_tryrdlock()`
 52775 function ever block; it always either acquires the lock or fails and returns immediately.

52776 Results are undefined if any of these functions are called with an uninitialized read-write lock.

52777 If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the
 52778 signal handler the thread resumes waiting for the read-write lock for reading as if it was not
 52779 interrupted.

52780 **RETURN VALUE**

52781 If successful, the `pthread_rwlock_rdlock()` function shall return zero; otherwise, an error number
 52782 shall be returned to indicate the error.

52783 The `pthread_rwlock_tryrdlock()` function shall return zero if the lock for reading on the read-write
 52784 lock object referenced by `rwlock` is acquired. Otherwise, an error number shall be returned to
 52785 indicate the error.

ERRORS

52786

52787

The *pthread_rwlock_tryrdlock()* function shall fail if:

52788

52789

[EBUSY] The read-write lock could not be acquired for reading because a writer holds the lock or a writer with the appropriate priority was blocked on it.

52790

The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions may fail if:

52791

52792

[EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock object.

52793

52794

[EAGAIN] The read lock could not be acquired because the maximum number of read locks for *rwlock* has been exceeded.

52795

The *pthread_rwlock_rdlock()* function may fail if:

52796

52797

[EDEADLK] A deadlock condition was detected or the current thread already owns the read-write lock for writing.

52798

These functions shall not return an error code of [EINTR].

EXAMPLES

52799

52800

None.

APPLICATION USAGE

52801

52802

52803

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.284 (on page 72).

RATIONALE

52804

52805

None.

FUTURE DIRECTIONS

52806

52807

None.

SEE ALSO

52808

52809

52810

pthread_rwlock_destroy(), *pthread_rwlock_timedrdlock()*, *pthread_rwlock_timedwrlock()*, *pthread_rwlock_trywrlock()*, *pthread_rwlock_unlock()*

52811

XBD Section 3.284 (on page 72), Section 4.11 (on page 98), **<pthread.h>**

CHANGE HISTORY

52812

52813

First released in Issue 5.

Issue 6

52814

52815

The following changes are made for alignment with IEEE Std 1003.1j-2000:

52816

52817

52818

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).

52819

- The DESCRIPTION is updated as follows:

52820

52821

52822

- Conditions under which writers have precedence over readers are specified.

- Failure of *pthread_rwlock_tryrdlock()* is clarified.

- A paragraph on the maximum number of read locks is added.

52823

52824

- In the ERRORS sections, [EBUSY] is modified to take into account write priority, and [EDEADLK] is deleted as a *pthread_rwlock_tryrdlock()* error.

52825

- The SEE ALSO section is updated.

52826

52827

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/101 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

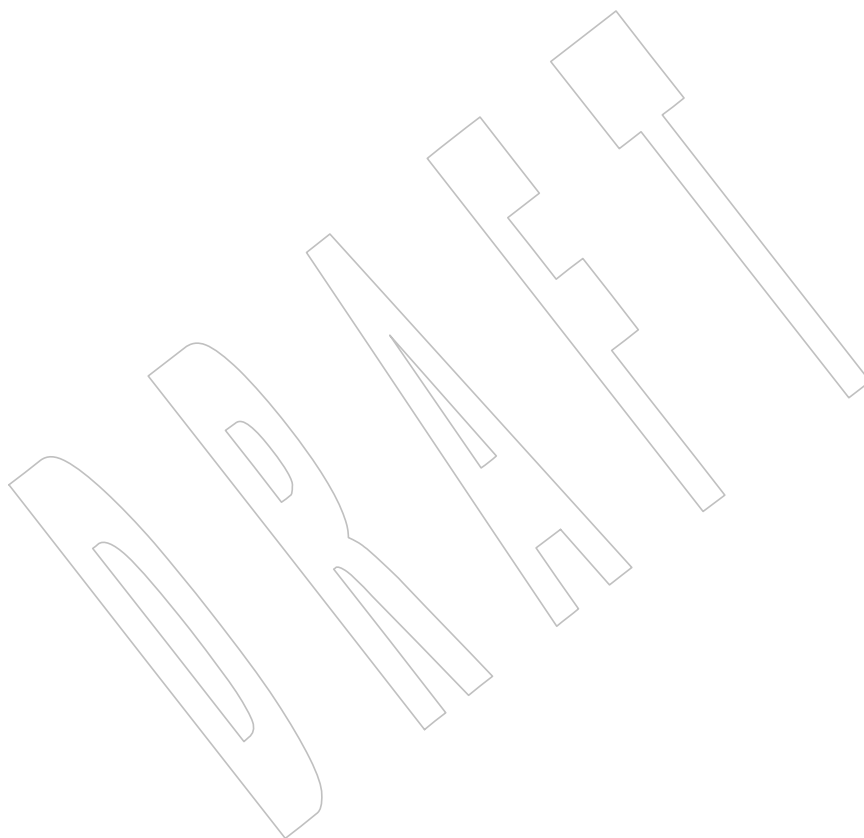
52828

Issue 7

52829

52830

The *pthread_rwlock_rdlock()* and *pthread_rwlock_tryrdlock()* functions are moved from the Threads option to the Base.



52831 **NAME**

52832 pthread_rwlock_timedrdlock — lock a read-write lock for reading

52833 **SYNOPSIS**

52834 #include <pthread.h>

52835 #include <time.h>

52836 int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict *rwlock*,
52837 const struct timespec *restrict *abstime*);52838 **DESCRIPTION**52839 The *pthread_rwlock_timedrdlock()* function shall apply a read lock to the read-write lock
52840 referenced by *rwlock* as in the *pthread_rwlock_rdlock()* function. However, if the lock cannot be
52841 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
52842 when the specified timeout expires. The timeout shall expire when the absolute time specified
52843 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value
52844 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already
52845 been passed at the time of the call.52846 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
52847 be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the
52848 <**time.h**> header. Under no circumstances shall the function fail with a timeout if the lock can be
52849 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can
52850 be immediately acquired.52851 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
52852 write lock via a call to *pthread_rwlock_timedrdlock()*, upon return from the signal handler the
52853 thread shall resume waiting for the lock as if it was not interrupted.52854 The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*.
52855 The results are undefined if this function is called with an uninitialized read-write lock.52856 **RETURN VALUE**52857 The *pthread_rwlock_timedrdlock()* function shall return zero if the lock for reading on the read-
52858 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
52859 to indicate the error.52860 **ERRORS**52861 The *pthread_rwlock_timedrdlock()* function shall fail if:

52862 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

52863 The *pthread_rwlock_timedrdlock()* function may fail if:52864 [EAGAIN] The read lock could not be acquired because the maximum number of read
52865 locks for lock would be exceeded.52866 [EDEADLK] A deadlock condition was detected or the calling thread already holds a write
52867 lock on *rwlock*.52868 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
52869 object, or the *abstime* nanosecond value is less than zero or greater than or
52870 equal to 1 000 million.

52871 This function shall not return an error code of [EINTR].

52872

EXAMPLES

52873

None.

52874

APPLICATION USAGE

52875

Applications using this function may be subject to priority inversion, as discussed in XBD Section 3.284 (on page 72).

52876

52877

RATIONALE

52878

None.

52879

FUTURE DIRECTIONS

52880

None.

52881

SEE ALSO

52882

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedwrlock()*, -

52883

pthread_rwlock_trywrlock(), *pthread_rwlock_unlock()*

52884

XBD Section 3.284 (on page 72), Section 4.11 (on page 98), **<pthread.h>**, **<time.h>** +

52885

CHANGE HISTORY

52886

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52887

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/102 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

52888

52889

Issue 7

52890

The *pthread_rwlock_timedrdlock()* function is moved from the Timeouts option to the Base.

DRAFT

52891 **NAME**

52892 pthread_rwlock_timedwrlock — lock a read-write lock for writing

52893 **SYNOPSIS**

52894 #include <pthread.h>

52895 #include <time.h>

52896 int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict *rwlock*,
52897 const struct timespec *restrict *abstime*);52898 **DESCRIPTION**52899 The *pthread_rwlock_timedwrlock()* function shall apply a write lock to the read-write lock
52900 referenced by *rwlock* as in the *pthread_rwlock_wrlock()* function. However, if the lock cannot be
52901 acquired without waiting for other threads to unlock the lock, this wait shall be terminated
52902 when the specified timeout expires. The timeout shall expire when the absolute time specified
52903 by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value
52904 of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already
52905 been passed at the time of the call.52906 The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall
52907 be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the
52908 <**time.h**> header. Under no circumstances shall the function fail with a timeout if the lock can be
52909 acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can
52910 be immediately acquired.52911 If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-
52912 write lock via a call to *pthread_rwlock_timedwrlock()*, upon return from the signal handler the
52913 thread shall resume waiting for the lock as if it was not interrupted.52914 The calling thread may deadlock if at the time the call is made it holds the read-write lock. The
52915 results are undefined if this function is called with an uninitialized read-write lock.52916 **RETURN VALUE**52917 The *pthread_rwlock_timedwrlock()* function shall return zero if the lock for writing on the read-
52918 write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned
52919 to indicate the error.52920 **ERRORS**52921 The *pthread_rwlock_timedwrlock()* function shall fail if:

52922 [ETIMEDOUT] The lock could not be acquired before the specified timeout expired.

52923 The *pthread_rwlock_timedwrlock()* function may fail if:52924 [EDEADLK] A deadlock condition was detected or the calling thread already holds the
52925 *rwlock*.52926 [EINVAL] The value specified by *rwlock* does not refer to an initialized read-write lock
52927 object, or the *abstime* nanosecond value is less than zero or greater than or
52928 equal to 1 000 million.

52929 This function shall not return an error code of [EINTR].

52930

EXAMPLES

52931

None.

52932

APPLICATION USAGE

52933

Applications using this function may be subject to priority inversion, as discussed in XBD Section 3.284 (on page 72).

52934

52935

RATIONALE

52936

None.

52937

FUTURE DIRECTIONS

52938

None.

52939

SEE ALSO

52940

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*, -

52941

pthread_rwlock_trywrlock(), *pthread_rwlock_unlock()*

52942

XBD Section 3.284 (on page 72), Section 4.11 (on page 98), **<pthread.h>**, **<time.h>** +

52943

CHANGE HISTORY

52944

First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

52945

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/103 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

52946

52947

Issue 7

52948

The *pthread_rwlock_timedwrlock()* function is moved from the Timeouts option to the Base.

DRAFT

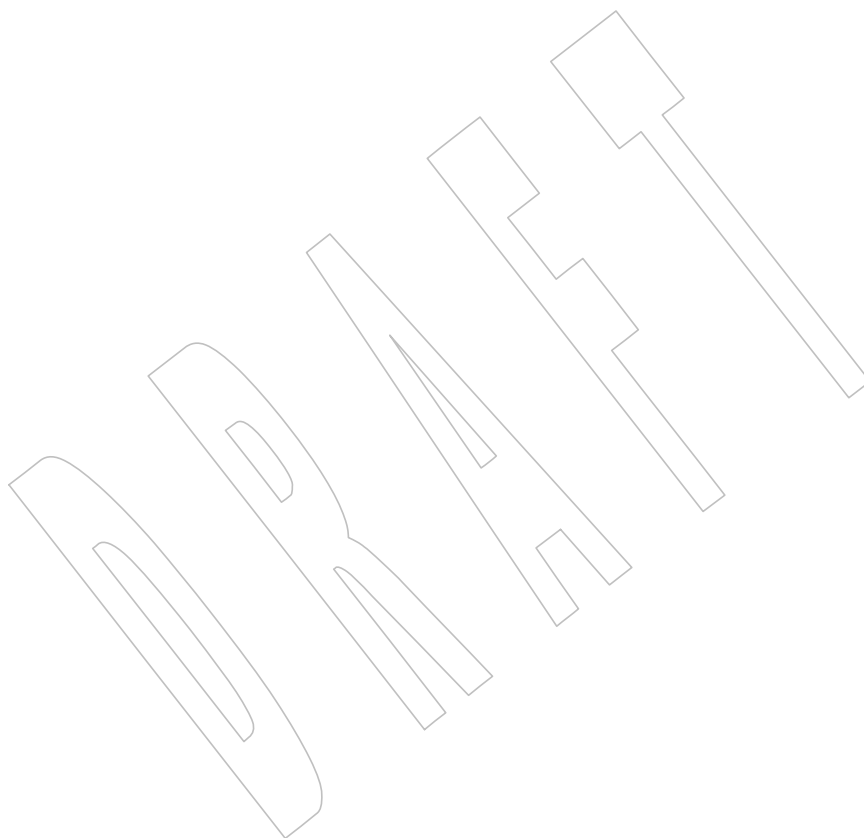
52949 **NAME**

52950 pthread_rwlock_tryrdlock — lock a read-write lock object for reading

52951 **SYNOPSIS**

52952 #include <pthread.h>

52953 int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);

52954 **DESCRIPTION**52955 Refer to *pthread_rwlock_rdlock()*.

52956 **NAME**

52957 pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

52958 **SYNOPSIS**

52959 #include <pthread.h>

52960 int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlck);

52961 int pthread_rwlock_wrlock(pthread_rwlock_t *rwlck);

52962 **DESCRIPTION**

52963 The *pthread_rwlock_trywrlock()* function shall apply a write lock like the *pthread_rwlock_wrlock()*
 52964 function, with the exception that the function shall fail if any thread currently holds *rwlck* (for
 52965 reading or writing).

52966 The *pthread_rwlock_wrlock()* function shall apply a write lock to the read-write lock referenced by
 52967 *rwlck*. The calling thread acquires the write lock if no other thread (reader or writer) holds the
 52968 read-write lock *rwlck*. Otherwise, the thread shall block until it can acquire the lock. The calling
 52969 thread may deadlock if at the time the call is made it holds the read-write lock (whether a read
 52970 or write lock).

52971 Implementations may favor writers over readers to avoid writer starvation.

52972 Results are undefined if any of these functions are called with an uninitialized read-write lock.

52973 If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the
 52974 signal handler the thread resumes waiting for the read-write lock for writing as if it was not
 52975 interrupted.

52976 **RETURN VALUE**

52977 The *pthread_rwlock_trywrlock()* function shall return zero if the lock for writing on the read-write
 52978 lock object referenced by *rwlck* is acquired. Otherwise, an error number shall be returned to
 52979 indicate the error.

52980 If successful, the *pthread_rwlock_wrlock()* function shall return zero; otherwise, an error number
 52981 shall be returned to indicate the error.

52982 **ERRORS**52983 The *pthread_rwlock_trywrlock()* function shall fail if:

52984 [EBUSY] The read-write lock could not be acquired for writing because it was already
 52985 locked for reading or writing.

52986 The *pthread_rwlock_trywrlock()* and *pthread_rwlock_wrlock()* functions may fail if:

52987 [EINVAL] The value specified by *rwlck* does not refer to an initialized read-write lock
 52988 object.

52989 The *pthread_rwlock_wrlock()* function may fail if:

52990 [EDEADLK] A deadlock condition was detected or the current thread already owns the
 52991 read-write lock for writing or reading.

52992 These functions shall not return an error code of [EINTR].

pthread_rwlock_trywrlock()

System Interfaces

52993
52994
52995
52996
52997
52998
52999
53000
53001
53002
53003
53004
53005
53006
53007
53008
53009
53010
53011
53012
53013
53014
53015
53016
53017
53018
53019**EXAMPLES**

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.284 (on page 72).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
pthread_rwlock_timedwrlock(), *pthread_rwlock_unlock()*

XBD Section 3.284 (on page 72), Section 4.11 (on page 98), <pthread.h>

+

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The [EDEADLK] error is deleted as a *pthread_rwlock_trywrlock()* error.
- The SEE ALSO section is updated.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/104 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The *pthread_rwlock_trywrlock()* and *pthread_rwlock_wrlock()* functions are moved from the Threads option to the Base.

53020 **NAME**
 53021 pthread_rwlock_unlock — unlock a read-write lock object

53022 **SYNOPSIS**
 53023 #include <pthread.h>

53024 int pthread_rwlock_unlock(pthread_rwlock_t *rwlck);

53025 **DESCRIPTION**

53026 The *pthread_rwlock_unlock()* function shall release a lock held on the read-write lock object
 53027 referenced by *rwlck*. Results are undefined if the read-write lock *rwlck* is not held by the
 53028 calling thread.

53029 If this function is called to release a read lock from the read-write lock object and there are other
 53030 read locks currently held on this read-write lock object, the read-write lock object remains in the
 53031 read locked state. If this function releases the last read lock for this read-write lock object, the
 53032 read-write lock object shall be put in the unlocked state with no owners.

53033 If this function is called to release a write lock for this read-write lock object, the read-write lock
 53034 object shall be put in the unlocked state.

53035 If there are threads blocked on the lock when it becomes available, the scheduling policy shall
 53036 determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is
 53037 supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or
 53038 SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when
 53039 the lock becomes available. For equal priority threads, write locks shall take precedence over
 53040 read locks. If the Thread Execution Scheduling option is not supported, it is implementation-
 53041 defined whether write locks take precedence over read locks.

53042 Results are undefined if this function is called with an uninitialized read-write lock.

53043 **RETURN VALUE**

53044 If successful, the *pthread_rwlock_unlock()* function shall return zero; otherwise, an error number
 53045 shall be returned to indicate the error.

53046 **ERRORS**

53047 The *pthread_rwlock_unlock()* function may fail if:

53048 [EINVAL] The value specified by *rwlck* does not refer to an initialized read-write lock
 53049 object.

53050 [EPERM] The current thread does not hold a lock on the read-write lock.

53051 The *pthread_rwlock_unlock()* function shall not return an error code of [EINTR].

53052 **EXAMPLES**

53053 None.

53054 **APPLICATION USAGE**

53055 None.

53056 **RATIONALE**

53057 None.

53058 **FUTURE DIRECTIONS**

53059 None.

53060
53061
53062
53063
53064
53065
53066
53067
53068
53069
53070
53071
53072
53073
53074
53075
53076
53077**SEE ALSO**

pthread_rwlock_destroy(), *pthread_rwlock_rdlock()*, *pthread_rwlock_timedrdlock()*,
pthread_rwlock_timedwrlock(), *pthread_rwlock_trywrlock()*

XBD Section 4.11 (on page 98), **<pthread.h>**

+

CHANGE HISTORY

First released in Issue 5.

Issue 6

The following changes are made for alignment with IEEE Std 1003.1j-2000:

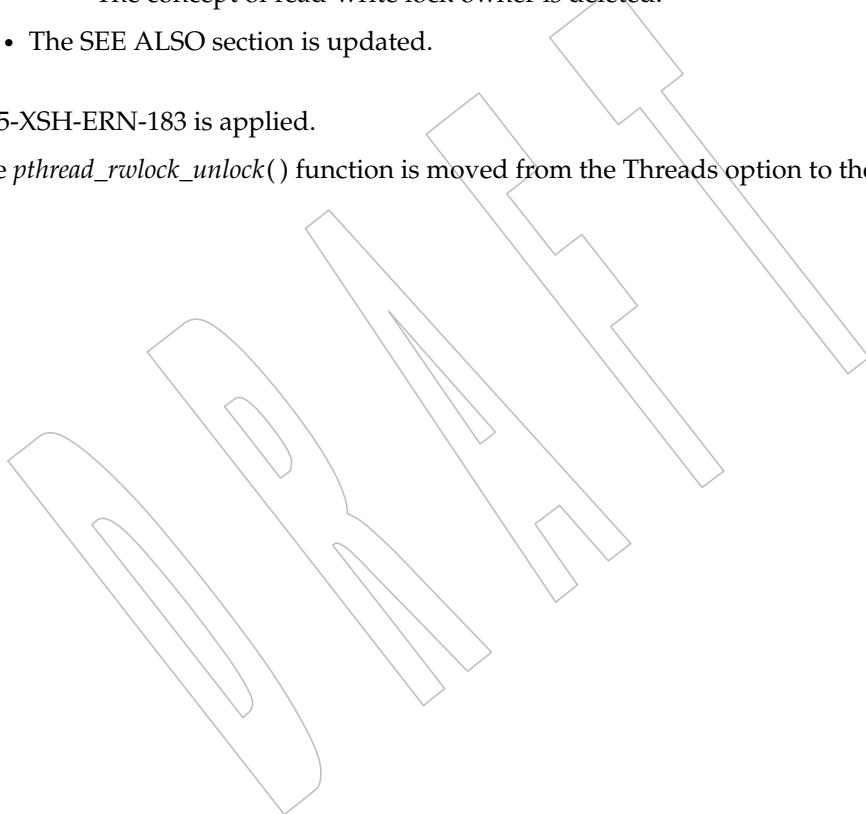
- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- The DESCRIPTION is updated as follows:
 - The conditions under which writers have precedence over readers are specified.
 - The concept of read-write lock owner is deleted.
- The SEE ALSO section is updated.

Issue 7

SD5-XSH-ERN-183 is applied.

+

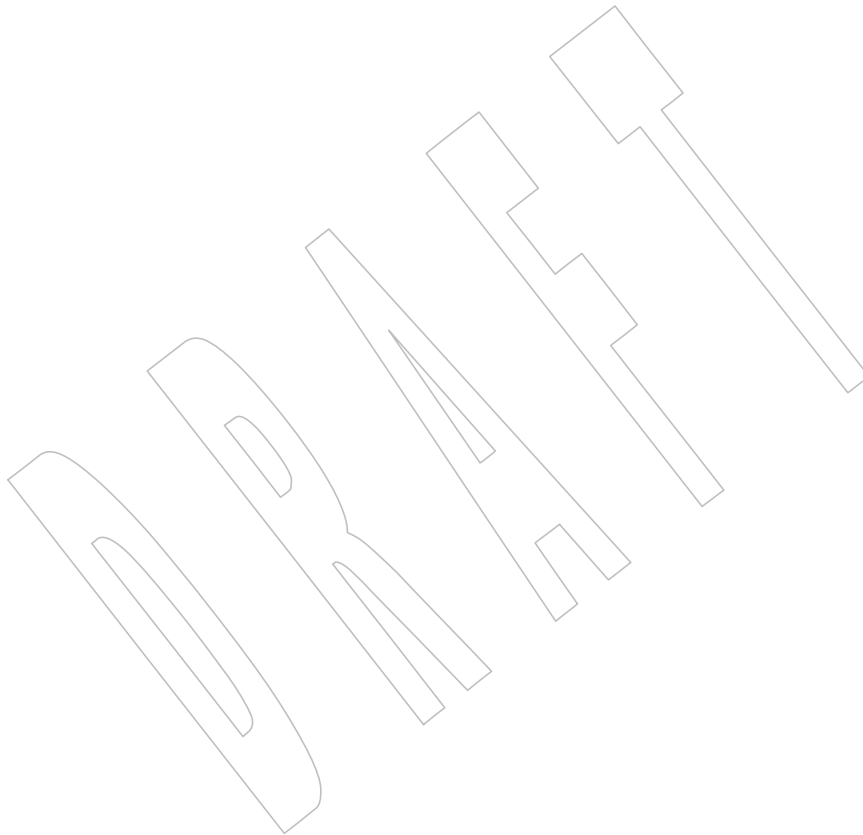
The *pthread_rwlock_unlock()* function is moved from the Threads option to the Base.



53078 **NAME**
53079 pthread_rwlock_wrlock — lock a read-write lock object for writing

53080 **SYNOPSIS**
53081 #include <pthread.h>
53082 int pthread_rwlock_wrlock(pthread_rwlock_t *rlock);

53083 **DESCRIPTION**
53084 Refer to *pthread_rwlock_trywrlock()*.



53085 **NAME**

53086 pthread_rwlockattr_destroy, pthread_rwlockattr_init — destroy and initialize the read-write
 53087 lock attributes object

53088 **SYNOPSIS**

53089 #include <pthread.h>

53090 int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);

53091 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

53092 **DESCRIPTION**

53093 The *pthread_rwlockattr_destroy()* function shall destroy a read-write lock attributes object. A
 53094 destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init()*; the results of
 53095 otherwise referencing the object after it has been destroyed are undefined. An implementation
 53096 may cause *pthread_rwlockattr_destroy()* to set the object referenced by *attr* to an invalid value.

53097 The *pthread_rwlockattr_init()* function shall initialize a read-write lock attributes object *attr* with
 53098 the default value for all of the attributes defined by the implementation.

53099 Results are undefined if *pthread_rwlockattr_init()* is called specifying an already initialized *attr*
 53100 attributes object.

53101 After a read-write lock attributes object has been used to initialize one or more read-write locks,
 53102 any function affecting the attributes object (including destruction) shall not affect any previously
 53103 initialized read-write locks.

53104 **RETURN VALUE**

53105 If successful, the *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions shall return
 53106 zero; otherwise, an error number shall be returned to indicate the error.

53107 **ERRORS**

53108 The *pthread_rwlockattr_destroy()* function may fail if:

53109 [EINVAL] The value specified by *attr* is invalid.

53110 The *pthread_rwlockattr_init()* function shall fail if:

53111 [ENOMEM] Insufficient memory exists to initialize the read-write lock attributes object.

53112 These functions shall not return an error code of [EINTR].

53113 **EXAMPLES**

53114 None.

53115 **APPLICATION USAGE**

53116 None.

53117 **RATIONALE**

53118 None.

53119 **FUTURE DIRECTIONS**

53120 None.

53121 **SEE ALSO**

53122 *pthread_rwlock_destroy()*, *pthread_rwlockattr_getpshared()*

53123 XBD <pthread.h>

53124

CHANGE HISTORY

53125

First released in Issue 5.

53126

Issue 6

53127

The following changes are made for alignment with IEEE Std 1003.1j-2000:

53128

- The margin code in the SYNOPSIS is changed to THR to indicate that the functionality is now part of the Threads option (previously it was part of the Read-Write Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).

53129

53130

53131

- The SEE ALSO section is updated.

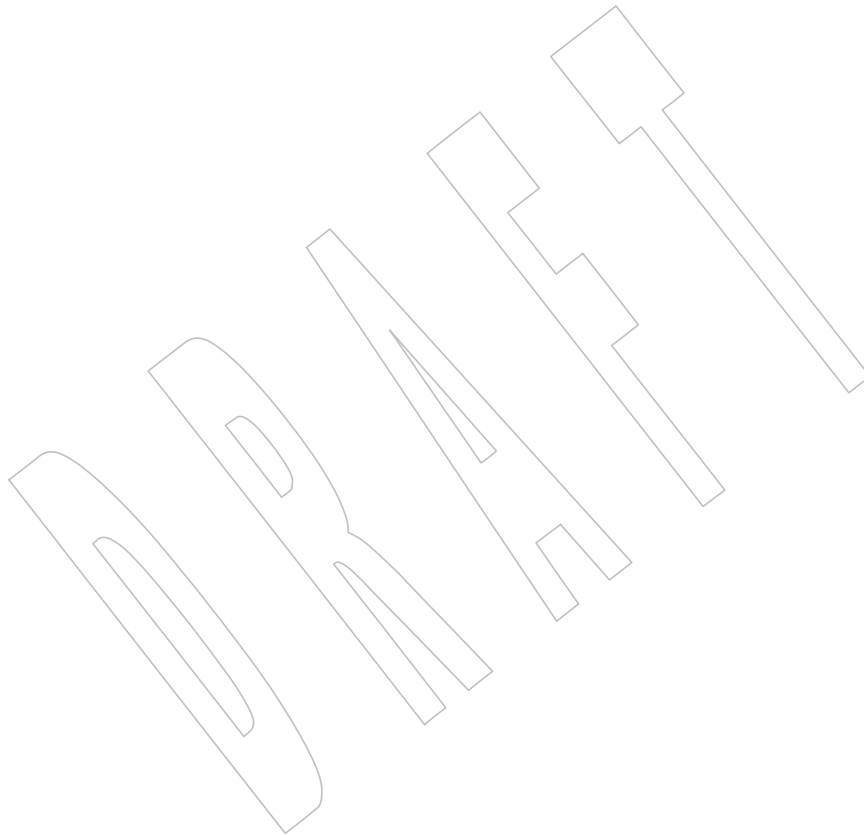
53132

Issue 7

53133

The *pthread_rwlockattr_destroy()* and *pthread_rwlockattr_init()* functions are moved from the Threads option to the Base.

53134



53135 **NAME**

53136 pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-
 53137 shared attribute of the read-write lock attributes object

53138 **SYNOPSIS**

```
53139 TSH #include <pthread.h>
53140
53141 int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
53142     *restrict attr, int *restrict pshared);
53143 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
53144     int pshared);
```

53144 **DESCRIPTION**

53145 The *pthread_rwlockattr_getpshared()* function shall obtain the value of the *process-shared* attribute
 53146 from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared()*
 53147 function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

53148 The *process-shared* attribute shall be set to PTHREAD_PROCESS_SHARED to permit a read-write
 53149 lock to be operated upon by any thread that has access to the memory where the read-write lock
 53150 is allocated, even if the read-write lock is allocated in memory that is shared by multiple
 53151 processes. If the *process-shared* attribute is PTHREAD_PROCESS_PRIVATE, the read-write lock
 53152 shall only be operated upon by threads created within the same process as the thread that
 53153 initialized the read-write lock; if threads of differing processes attempt to operate on such a
 53154 read-write lock, the behavior is undefined. The default value of the *process-shared* attribute shall
 53155 be PTHREAD_PROCESS_PRIVATE.

53156 Additional attributes, their default values, and the names of the associated functions to get and
 53157 set those attribute values are implementation-defined.

53158 **RETURN VALUE**

53159 Upon successful completion, the *pthread_rwlockattr_getpshared()* function shall return zero and
 53160 store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared*
 53161 parameter. Otherwise, an error number shall be returned to indicate the error.

53162 If successful, the *pthread_rwlockattr_setpshared()* function shall return zero; otherwise, an error
 53163 number shall be returned to indicate the error.

53164 **ERRORS**

53165 The *pthread_rwlockattr_getpshared()* and *pthread_rwlockattr_setpshared()* functions may fail if:

53166 [EINVAL] The value specified by *attr* is invalid.

53167 The *pthread_rwlockattr_setpshared()* function may fail if:

53168 [EINVAL] The new value specified for the attribute is outside the range of legal values
 53169 for that attribute.

53170 These functions shall not return an error code of [EINTR].

53171 **EXAMPLES**
 53172 None.

53173 **APPLICATION USAGE**
 53174 None.

53175 **RATIONALE**
 53176 None.

53177 **FUTURE DIRECTIONS**
 53178 None.

53179 **SEE ALSO**
 53180 *pthread_rwlock_destroy()*, *pthread_rwlockattr_destroy()*

53181 XBD <pthread.h>

53182 **CHANGE HISTORY**
 53183 First released in Issue 5.

53184 **Issue 6**
 53185 The following changes are made for alignment with IEEE Std 1003.1j-2000:

- 53186 • The margin code in the SYNOPSIS is changed to THR TSH to indicate that the
- 53187 functionality is now part of the Threads option (previously it was part of the Read-Write
- 53188 Locks option in IEEE Std 1003.1j-2000 and also part of the XSI extension).
- 53189 • The DESCRIPTION notes that additional attributes are implementation-defined.
- 53190 • The SEE ALSO section is updated.

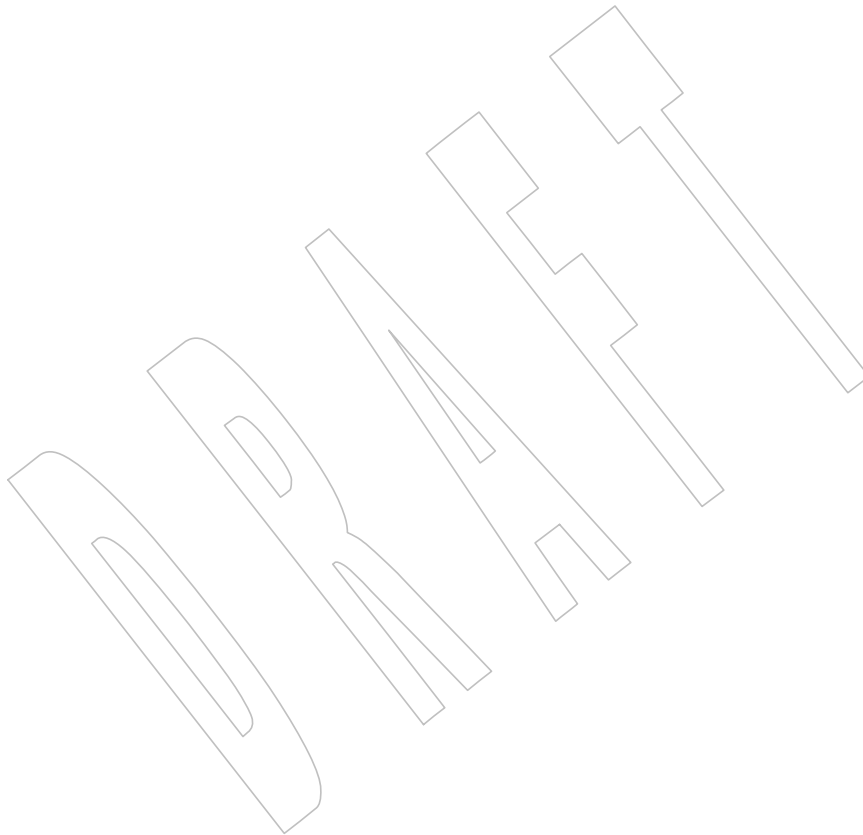
53191 The **restrict** keyword is added to the *pthread_rwlockattr_getpshared()* prototype for alignment
 53192 with the ISO/IEC 9899:1999 standard.

53193 **Issue 7**
 53194 The *pthread_rwlockattr_getpshared()* and *pthread_rwlockattr_setpshared()* functions are moved from
 53195 the Threads option.

53196 **NAME**
53197 pthread_rwlockattr_init — initialize the read-write lock attributes object

53198 **SYNOPSIS**
53199 #include <pthread.h>
53200 int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

53201 **DESCRIPTION**
53202 Refer to *pthread_rwlockattr_destroy()*.



53203 **NAME**

53204 pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock
53205 attributes object

53206 **SYNOPSIS**

```
53207 TSH #include <pthread.h>  
53208 int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,  
53209 int pshared);
```

53210 **DESCRIPTION**

53211 Refer to [pthread_rwlockattr_getpshared\(\)](#).

53212 **NAME**

53213 pthread_self — get the calling thread ID

53214 **SYNOPSIS**

53215 #include <pthread.h>

53216 pthread_t pthread_self(void);

53217 **DESCRIPTION**53218 The *pthread_self()* function shall return the thread ID of the calling thread.53219 **RETURN VALUE**53220 The *pthread_self()* function shall always be successful and no return value is reserved to indicate
53221 an error.53222 **ERRORS**

53223 No errors are defined.

53224 **EXAMPLES**

53225 None.

53226 **APPLICATION USAGE**

53227 None.

53228 **RATIONALE**53229 The *pthread_self()* function provides a capability similar to the *getpid()* function for processes
53230 and the rationale is the same: the creation call does not provide the thread ID to the created
53231 thread.53232 **FUTURE DIRECTIONS**

53233 None.

53234 **SEE ALSO**53235 *pthread_create()*, *pthread_equal()*

53236 XBD <pthread.h>

53237 **CHANGE HISTORY**

53238 First released in Issue 5. Included for alignment with the POSIX Threads Extension.

53239 **Issue 6**53240 The *pthread_self()* function is marked as part of the Threads option.53241 **Issue 7**

53242 Austin Group Interpretation 1003.1-2001 #063 is applied, updating the RETURN VALUE section.

53243 The *pthread_self()* function is moved from the Threads option to the Base.

53244 **NAME**
 53245 pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

53246 **SYNOPSIS**
 53247 #include <pthread.h>
 53248 int pthread_setcancelstate(int state, int *oldstate);
 53249 int pthread_setcanceltype(int type, int *oldtype);
 53250 void pthread_testcancel(void);

53251 **DESCRIPTION**
 53252 The *pthread_setcancelstate()* function shall atomically both set the calling thread's cancelability
 53253 state to the indicated *state* and return the previous cancelability state at the location referenced
 53254 by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and
 53255 PTHREAD_CANCEL_DISABLE.

53256 The *pthread_setcanceltype()* function shall atomically both set the calling thread's cancelability
 53257 type to the indicated *type* and return the previous cancelability type at the location referenced by
 53258 *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and
 53259 PTHREAD_CANCEL_ASYNCCHRONOUS.

53260 The cancelability state and type of any newly created threads, including the thread in which
 53261 *main()* was first invoked, shall be PTHREAD_CANCEL_ENABLE and
 53262 PTHREAD_CANCEL_DEFERRED respectively.

53263 The *pthread_testcancel()* function shall create a cancellation point in the calling thread. The
 53264 *pthread_testcancel()* function shall have no effect if cancelability is disabled.

53265 **RETURN VALUE**
 53266 If successful, the *pthread_setcancelstate()* and *pthread_setcanceltype()* functions shall return zero;
 53267 otherwise, an error number shall be returned to indicate the error.

53268 **ERRORS**
 53269 The *pthread_setcancelstate()* function may fail if:
 53270 [EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or
 53271 PTHREAD_CANCEL_DISABLE.

53272 The *pthread_setcanceltype()* function may fail if:
 53273 [EINVAL] The specified type is not PTHREAD_CANCEL_DEFERRED or
 53274 PTHREAD_CANCEL_ASYNCCHRONOUS.

53275 These functions shall not return an error code of [EINTR].

53276 **EXAMPLES**
 53277 None.

53278 **APPLICATION USAGE**
 53279 None.

53280 **RATIONALE**
 53281 The *pthread_setcancelstate()* and *pthread_setcanceltype()* functions control the points at which a
 53282 thread may be asynchronously canceled. For cancellation control to be usable in modular
 53283 fashion, some rules need to be followed.

53284 An object can be considered to be a generalization of a procedure. It is a set of procedures and
 53285 global variables written as a unit and called by clients not known by the object. Objects may
 53286 depend on other objects.

pthread_setcancelstate()*System Interfaces*

53287 First, cancelability should only be disabled on entry to an object, never explicitly enabled. On
 53288 exit from an object, the cancelability state should always be restored to its value on entry to the
 53289 object.

53290 This follows from a modularity argument: if the client of an object (or the client of an object that
 53291 uses that object) has disabled cancelability, it is because the client does not want to be concerned
 53292 about cleaning up if the thread is canceled while executing some sequence of actions. If an object
 53293 is called in such a state and it enables cancelability and a cancellation request is pending for that
 53294 thread, then the thread is canceled, contrary to the wish of the client that disabled.

53295 Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry
 53296 to an object. But as with the cancelability state, on exit from an object the cancelability type
 53297 should always be restored to its value on entry to the object.

53298 Finally, only functions that are cancel-safe may be called from a thread that is asynchronously
 53299 cancelable.

FUTURE DIRECTIONS

53300 None.
 53301

SEE ALSO

53302 [*pthread_cancel\(\)*](#)

53303 XBD [*<pthread.h>*](#)
 53304

CHANGE HISTORY

53305 First released in Issue 5. Included for alignment with the POSIX Threads Extension.
 53306

Issue 6

53307 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are marked
 53308 as part of the Threads option.
 53309

Issue 7

53310 The *pthread_setcancelstate()*, *pthread_setcanceltype()*, and *pthread_testcancel()* functions are moved
 53311 from the Threads option to the Base.
 53312

53313 **NAME**
53314 pthread_setconcurrency — set the level of concurrency

53315 **SYNOPSIS**

53316 OB XSI #include <pthread.h>
53317 int pthread_setconcurrency(int new_level);

53318 **DESCRIPTION**

53319 Refer to [pthread_getconcurrency\(\)](#).

pthread_setschedparam()*System Interfaces*53320 **NAME**

53321 pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME**
53322 **THREADS**)

53323 **SYNOPSIS**

```
53324 TPS #include <pthread.h>  
  
53325 int pthread_setschedparam(pthread_t thread, int policy,  
53326 const struct sched_param *param);
```

53327 **DESCRIPTION**

53328 Refer to *pthread_getschedparam()*.

53329 **NAME**

53330 pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME**
53331 **THREADS**)

53332 **SYNOPSIS**

```
53333 TPS #include <pthread.h>
53334 int pthread_setschedprio(pthread_t thread, int prio);
```

53335 **DESCRIPTION**

53336 The *pthread_setschedprio()* function shall set the scheduling priority for the thread whose thread
53337 ID is given by *thread* to the value given by *prio*. See [Scheduling Policies](#) (on page 479) for a
53338 description on how this function call affects the ordering of the thread in the thread list for its
53339 new priority.

53340 If the *pthread_setschedprio()* function fails, the scheduling priority of the target thread shall not be
53341 changed.

53342 **RETURN VALUE**

53343 If successful, the *pthread_setschedprio()* function shall return zero; otherwise, an error number
53344 shall be returned to indicate the error.

53345 **ERRORS**

53346 The *pthread_setschedprio()* function may fail if:

- 53347 [EINVAL] The value of *prio* is invalid for the scheduling policy of the specified thread.
- 53348 [ENOTSUP] An attempt was made to set the priority to an unsupported value.
- 53349 [EPERM] The caller does not have the appropriate permission to set the scheduling
53350 priority of the specified thread.
- 53351 [ESRCH] The value specified by *thread* does not refer to an existing thread.

53352 The *pthread_setschedprio()* function shall not return an error code of [EINTR].

53353 **EXAMPLES**

53354 None.

53355 **APPLICATION USAGE**

53356 None.

53357 **RATIONALE**

53358 The *pthread_setschedprio()* function provides a way for an application to temporarily raise its
53359 priority and then lower it again, without having the undesired side effect of yielding to other
53360 threads of the same priority. This is necessary if the application is to implement its own
53361 strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This
53362 capability is especially important if the implementation does not support the Thread Priority
53363 Protection or Thread Priority Inheritance options, but even if those options are supported it is
53364 needed if the application is to bound priority inheritance for other resources, such as
53365 semaphores.

53366 The standard developers considered that while it might be preferable conceptually to solve this
53367 problem by modifying the specification of *pthread_setschedparam()*, it was too late to make such a
53368 change, as there may be implementations that would need to be changed. Therefore, this new
53369 function was introduced.

pthread_setschedprio()*System Interfaces*53370 **FUTURE DIRECTIONS**

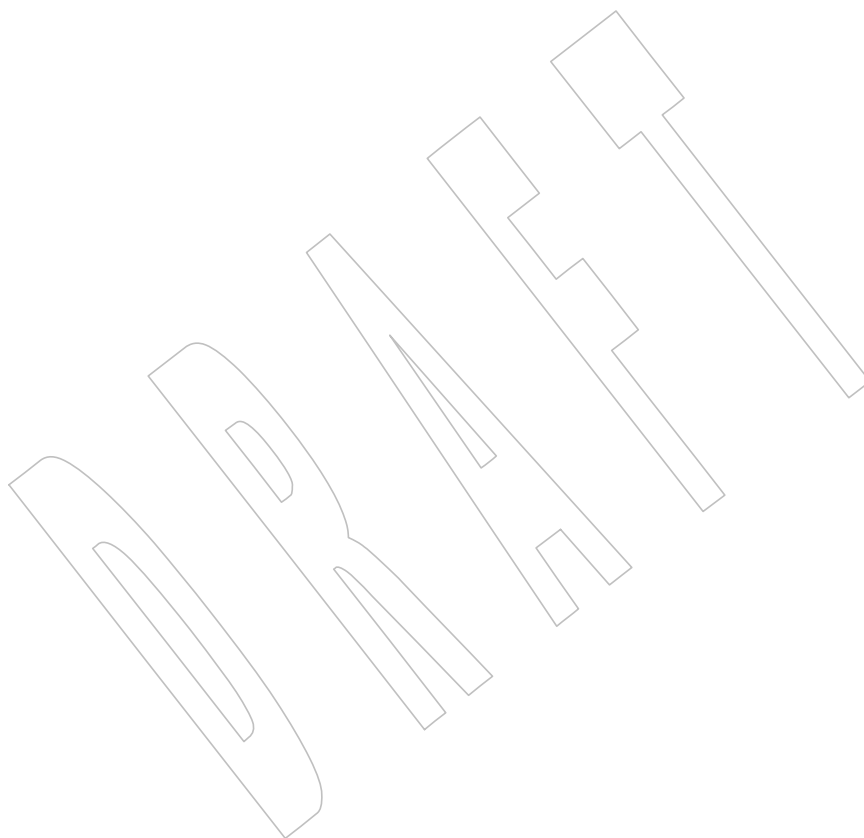
53371 None.

53372 **SEE ALSO**53373 [Scheduling Policies](#) (on page 479), [pthread_getschedparam\(\)](#)53374 XBD [<pthread.h>](#)53375 **CHANGE HISTORY**

53376 First released in Issue 6. Included as a response to IEEE PASC Interpretation 1003.1 #96.

53377 **Issue 7**

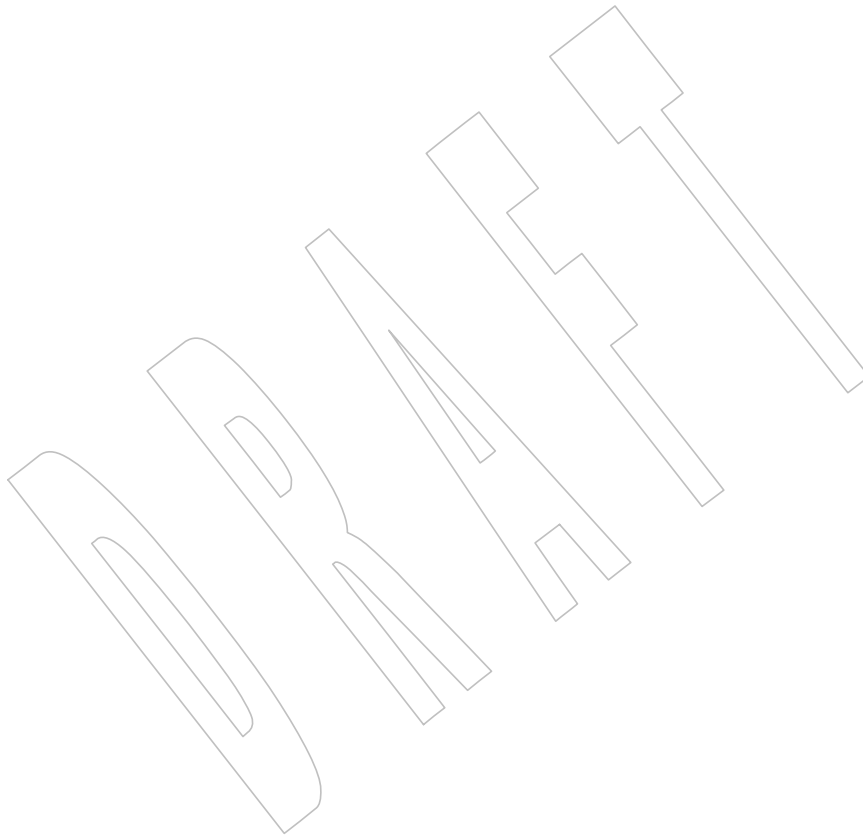
53378 Austin Group Interpretation 1003.1-2001 #069 is applied, updating the [EPERM] error.

53379 The `pthread_setschedprio()` function is moved from the Threads option.

53380 **NAME**
53381 pthread_setspecific — thread-specific data management

53382 **SYNOPSIS**
53383 #include <pthread.h>
53384 int pthread_setspecific(pthread_key_t key, const void *value);

53385 **DESCRIPTION**
53386 Refer to *pthread_getspecific()*.



53387 **NAME**

53388 pthread_sigmask, sigprocmask — examine and change blocked signals

53389 **SYNOPSIS**

```
53390 CX #include <signal.h>
53391 int pthread_sigmask(int how, const sigset_t *restrict set,
53392 sigset_t *restrict oset);
53393 int sigprocmask(int how, const sigset_t *restrict set,
53394 sigset_t *restrict oset);
```

53395 **DESCRIPTION**

53396 The *pthread_sigmask()* function shall examine or change (or both) the calling thread's signal
 53397 mask, regardless of the number of threads in the process. The function shall be equivalent to
 53398 *sigprocmask()*, without the restriction that the call be made in a single-threaded process.

53399 In a single-threaded process, the *sigprocmask()* function shall examine or change (or both) the
 53400 signal mask of the calling thread.

53401 If the argument *set* is not a null pointer, it points to a set of signals to be used to change the
 53402 currently blocked set.

53403 The argument *how* indicates the way in which the set is changed, and the application shall
 53404 ensure it consists of one of the following values:

53405 SIG_BLOCK The resulting set shall be the union of the current set and the signal set
 53406 pointed to by *set*.

53407 SIG_SETMASK The resulting set shall be the signal set pointed to by *set*.

53408 SIG_UNBLOCK The resulting set shall be the intersection of the current set and the
 53409 complement of the signal set pointed to by *set*.

53410 If the argument *oset* is not a null pointer, the previous mask shall be stored in the location
 53411 pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the
 53412 thread's signal mask shall be unchanged; thus the call can be used to enquire about currently
 53413 blocked signals.

53414 If there are any pending unblocked signals after the call to *sigprocmask()*, at least one of those
 53415 signals shall be delivered before the call to *sigprocmask()* returns.

53416 It is not possible to block those signals which cannot be ignored. This shall be enforced by the
 53417 system without causing an error to be indicated.

53418 If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked,
 53419 the result is undefined, unless the signal was generated by the *kill()* function, the *sigqueue()*
 53420 function, or the *raise()* function.

53421 If *sigprocmask()* fails, the thread's signal mask shall not be changed.

53422 The use of the *sigprocmask()* function is unspecified in a multi-threaded process.

53423 **RETURN VALUE**

53424 Upon successful completion *pthread_sigmask()* shall return 0; otherwise, it shall return the
 53425 corresponding error number.

53426 Upon successful completion, *sigprocmask()* shall return 0; otherwise, -1 shall be returned, *errno*
 53427 shall be set to indicate the error, and the signal mask of the process shall be unchanged.

53428 **ERRORS**53429 The *pthread_sigmask()* and *sigprocmask()* functions shall fail if:53430 [EINVAL] The value of the *how* argument is not equal to one of the defined values.53431 The *pthread_sigmask()* function shall not return an error code of [EINTR].53432 **EXAMPLES**53433 **Signalling in a Multi-Threaded Process**53434 This example shows the use of *pthread_sigmask()* in order to deal with signals in a multi-
53435 threaded process. It provides a fairly general framework that could be easily adapted/extended.

```

53436 #include <stdio.h>
53437 #include <stdlib.h>
53438 #include <pthread.h>
53439 #include <signal.h>
53440 #include <string.h>
53441 #include <errno.h>
53442 ...
53443 static sigset_t  signal_mask; /* signals to block          */
53444 int main (int argc, char *argv[])
53445 {
53446     pthread_t  sig_thr_id; /* signal handler thread ID */
53447     int        rc;        /* return code              */
53448     sigemptyset (&signal_mask);
53449     sigaddset (&signal_mask, SIGINT);
53450     sigaddset (&signal_mask, SIGTERM);
53451     rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
53452     if (rc != 0) {
53453         /* handle error */
53454         ...
53455     }
53456     /* any newly created threads inherit the signal mask */
53457     rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
53458     if (rc != 0) {
53459         /* handle error */
53460         ...
53461     }
53462     /* APPLICATION CODE */
53463     ...
53464 }
53465 void *signal_thread (void *arg)
53466 {
53467     int        sig_caught; /* signal caught          */
53468     int        rc;        /* returned code          */
53469     rc = sigwait (&signal_mask, &sig_caught);
53470     if (rc != 0) {
53471         /* handle error */
53472     }
53473     switch (sig_caught)
53474     {

```

```

53475         case SIGINT:      /* process SIGINT */
53476             ...
53477             break;
53478         case SIGTERM:    /* process SIGTERM */
53479             ...
53480             break;
53481         default:        /* should normally not happen */
53482             fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
53483             break;
53484     }
53485 }

```

APPLICATION USAGE

None.

RATIONALE

When a thread's signal mask is changed in a signal-catching function that is installed by *sigaction()*, the restoration of the signal mask on return from the signal-catching function overrides that change (see *sigaction()*). If the signal-catching function was installed with *signal()*, it is unspecified whether this occurs.

See *kill* for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigqueue()*, *sigsuspend()*

XBD **<signal.h>**

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

The *pthread_sigmask()* function is added for alignment with the POSIX Threads Extension.

Issue 6

The *pthread_sigmask()* function is marked as part of the Threads option.

The SYNOPSIS for *sigprocmask()* is marked as a CX extension to note that the presence of this function in the **<signal.h>** header is an extension to the ISO C standard.

The following changes are made for alignment with the ISO POSIX-1:1996 standard:

- The DESCRIPTION is updated to explicitly state the functions which may generate the signal.

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *pthread_sigmask()* and *sigprocmask()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/105 is applied, updating “process’ signal mask” to “thread’s signal mask” in the DESCRIPTION and RATIONALE sections.

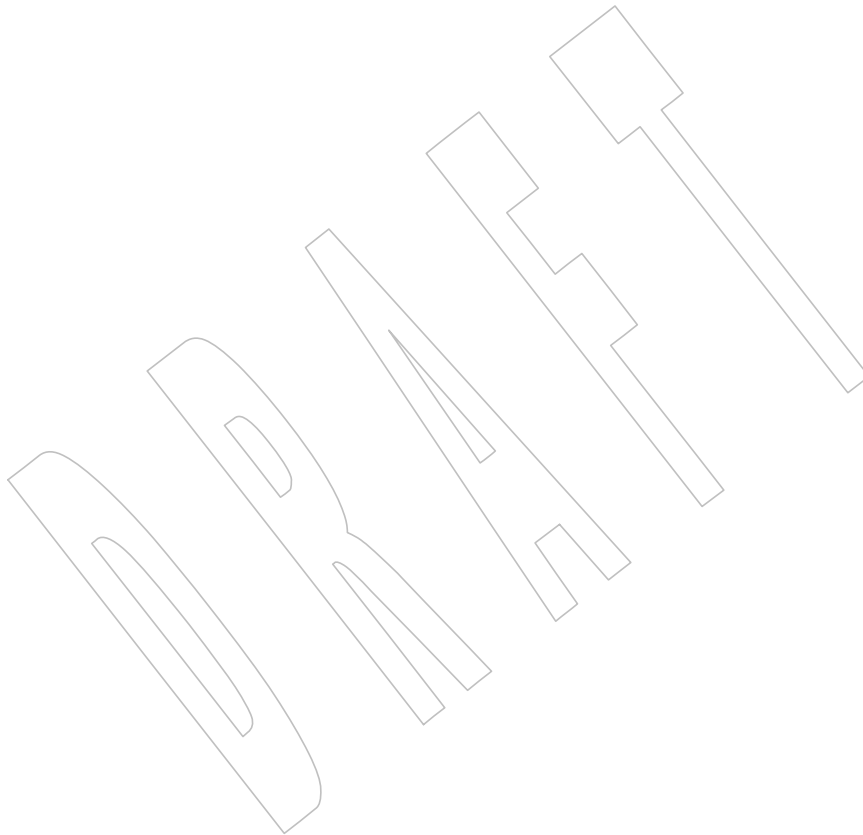
IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/106 is applied, adding the example to the EXAMPLES section.

53519

Issue 7

53520

The *pthread_sigmask()* function is moved from the Threads option to the Base.



53521 **NAME**

53522 pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object

53523 **SYNOPSIS**

53524 #include <pthread.h>

53525 int pthread_spin_destroy(pthread_spinlock_t *lock);

53526 int pthread_spin_init(pthread_spinlock_t *lock, int pshared);

53527 **DESCRIPTION**

53528 The *pthread_spin_destroy()* function shall destroy the spin lock referenced by *lock* and release any
 53529 resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is
 53530 reinitialized by another call to *pthread_spin_init()*. The results are undefined if
 53531 *pthread_spin_destroy()* is called when a thread holds the lock, or if this function is called with an
 53532 uninitialized thread spin lock.

53533 The *pthread_spin_init()* function shall allocate any resources required to use the spin lock
 53534 referenced by *lock* and initialize the lock to an unlocked state.

53535 TSH If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
 53536 PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated
 53537 upon by any thread that has access to the memory where the spin lock is allocated, even if it is
 53538 allocated in memory that is shared by multiple processes.

53539 If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is
 53540 PTHREAD_PROCESS_PRIVATE, or if the option is not supported, the spin lock shall only be
 53541 operated upon by threads created within the same process as the thread that initialized the spin
 53542 lock. If threads of differing processes attempt to operate on such a spin lock, the behavior is
 53543 undefined.

53544 The results are undefined if *pthread_spin_init()* is called specifying an already initialized spin
 53545 lock. The results are undefined if a spin lock is used without first being initialized.

53546 If the *pthread_spin_init()* function fails, the lock is not initialized and the contents of *lock* are
 53547 undefined.

53548 Only the object referenced by *lock* may be used for performing synchronization.

53549 The result of referring to copies of that object in calls to *pthread_spin_destroy()*,
 53550 *pthread_spin_lock()*, *pthread_spin_trylock()*, or *pthread_spin_unlock()* is undefined.

53551 **RETURN VALUE**

53552 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 53553 be returned to indicate the error.

53554 **ERRORS**

53555 These functions may fail if:

53556 [EBUSY] The implementation has detected an attempt to initialize or destroy a spin lock
 53557 while it is in use (for example, while being used in a *pthread_spin_lock()* call)
 53558 by another thread.

53559 [EINVAL] The value specified by *lock* is invalid.

53560 The *pthread_spin_init()* function shall fail if:

53561 [EAGAIN] The system lacks the necessary resources to initialize another spin lock.

53562 [ENOMEM] Insufficient memory exists to initialize the lock.

53563 These functions shall not return an error code of [EINTR].

53564 **EXAMPLES**

53565 None.

53566 **APPLICATION USAGE**

53567 None.

53568 **RATIONALE**

53569 None.

53570 **FUTURE DIRECTIONS**

53571 None.

53572 **SEE ALSO**

53573 *pthread_spin_lock()*, *pthread_spin_unlock()*

53574 XBD **<pthread.h>**

53575 **CHANGE HISTORY**

53576 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53577 In the SYNOPSIS, the inclusion of **<sys/types.h>** is no longer required.

53578 **Issue 7**

53579 The *pthread_spin_destroy()* and *pthread_spin_init()* functions are moved from the Spin Locks
53580 option to the Base.

53581 **NAME**
 53582 pthread_spin_lock, pthread_spin_trylock — lock a spin lock object

53583 **SYNOPSIS**
 53584 #include <pthread.h>
 53585 int pthread_spin_lock(pthread_spinlock_t *lock);
 53586 int pthread_spin_trylock(pthread_spinlock_t *lock);

53587 **DESCRIPTION**
 53588 The *pthread_spin_lock()* function shall lock the spin lock referenced by *lock*. The calling thread
 53589 shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is,
 53590 shall not return from the *pthread_spin_lock()* call) until the lock becomes available. The results are
 53591 undefined if the calling thread holds the lock at the time the call is made. The
 53592 *pthread_spin_trylock()* function shall lock the spin lock referenced by *lock* if it is not held by any
 53593 thread. Otherwise, the function shall fail.

53594 The results are undefined if any of these functions is called with an uninitialized spin lock.

53595 **RETURN VALUE**
 53596 Upon successful completion, these functions shall return zero; otherwise, an error number shall
 53597 be returned to indicate the error.

53598 **ERRORS**
 53599 These functions may fail if:
 53600 [EINVAL] The value specified by *lock* does not refer to an initialized spin lock object.
 53601 The *pthread_spin_lock()* function may fail if:
 53602 [EDEADLK] A deadlock condition was detected or the calling thread already holds the
 53603 lock.
 53604 The *pthread_spin_trylock()* function shall fail if:
 53605 [EBUSY] A thread currently holds the lock.
 53606 These functions shall not return an error code of [EINTR].

53607 **EXAMPLES**
 53608 None.

53609 **APPLICATION USAGE**
 53610 Applications using this function may be subject to priority inversion, as discussed in XBD
 53611 [Section 3.284](#) (on page 72).

53612 **RATIONALE**
 53613 None.

53614 **FUTURE DIRECTIONS**
 53615 None.

53616 **SEE ALSO**
 53617 [pthread_spin_destroy\(\)](#), [pthread_spin_unlock\(\)](#)
 53618 XBD [Section 3.284](#) (on page 72), [Section 4.11](#) (on page 98), [1 <pthread.h>](#)

53619
53620
53621
53622
53623
53624
53625
53626

CHANGE HISTORY

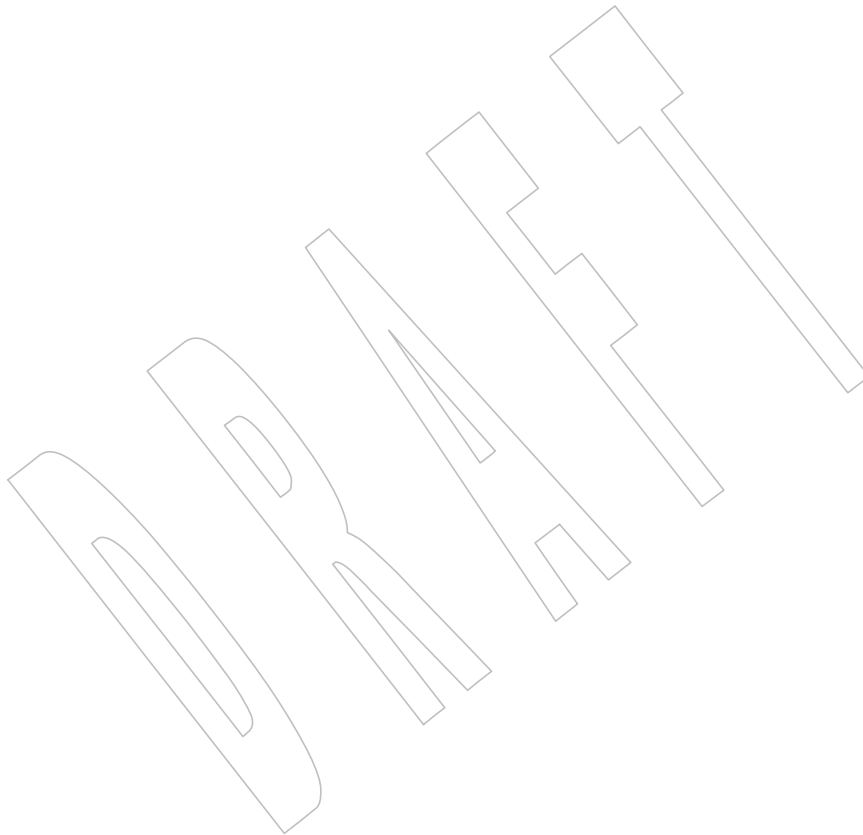
First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/107 is applied, updating the ERRORS section so that the [EDEADLK] error includes detection of a deadlock condition.

Issue 7

The `pthread_spin_lock()` and `pthread_spin_trylock()` functions are moved from the Spin Locks option to the Base.



53627 **NAME**

53628 pthread_spin_unlock — unlock a spin lock object

53629 **SYNOPSIS**

53630 #include <pthread.h>

53631 int pthread_spin_unlock(pthread_spinlock_t *lock);

53632 **DESCRIPTION**

53633 The *pthread_spin_unlock()* function shall release the spin lock referenced by *lock* which was
 53634 locked via the *pthread_spin_lock()* or *pthread_spin_trylock()* functions. The results are undefined if
 53635 the lock is not held by the calling thread. If there are threads spinning on the lock when
 53636 *pthread_spin_unlock()* is called, the lock becomes available and an unspecified spinning thread
 53637 shall acquire the lock.

53638 The results are undefined if this function is called with an uninitialized thread spin lock.

53639 **RETURN VALUE**

53640 Upon successful completion, the *pthread_spin_unlock()* function shall return zero; otherwise, an
 53641 error number shall be returned to indicate the error.

53642 **ERRORS**53643 The *pthread_spin_unlock()* function may fail if:

53644 [EINVAL] An invalid argument was specified.

53645 [EPERM] The calling thread does not hold the lock.

53646 This function shall not return an error code of [EINTR].

53647 **EXAMPLES**

53648 None.

53649 **APPLICATION USAGE**

53650 None.

53651 **RATIONALE**

53652 None.

53653 **FUTURE DIRECTIONS**

53654 None.

53655 **SEE ALSO**53656 [pthread_spin_destroy\(\)](#), [pthread_spin_lock\(\)](#)53657 XBD Section 4.11 (on page 98), [<pthread.h>](#)53658 **CHANGE HISTORY**

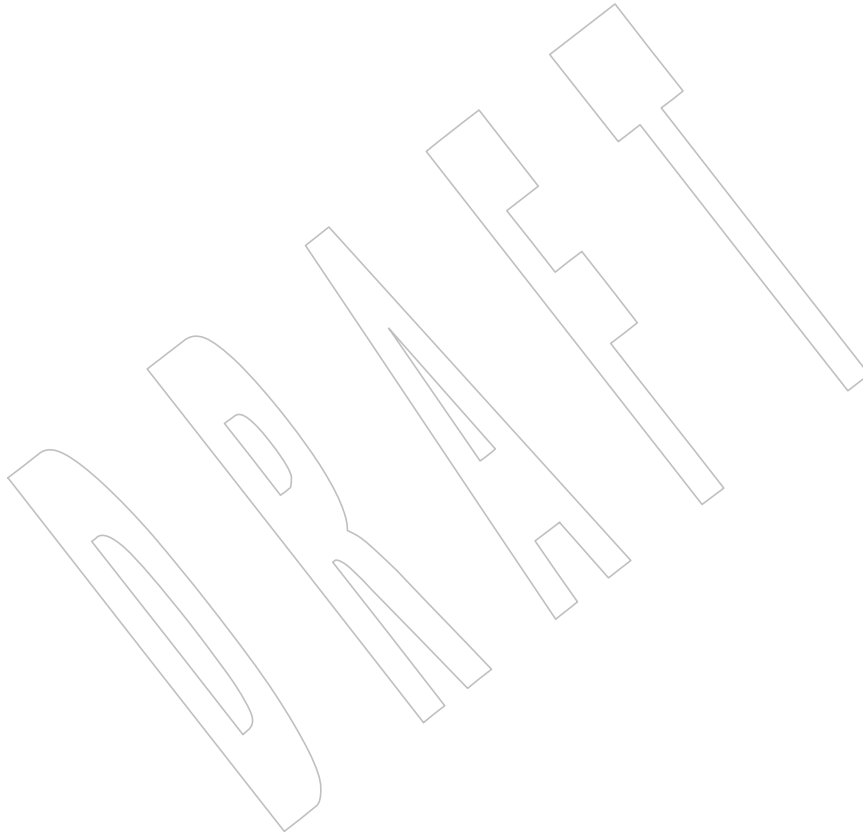
53659 First released in Issue 6. Derived from IEEE Std 1003.1j-2000.

53660 In the SYNOPSIS, the inclusion of [<sys/types.h>](#) is no longer required.53661 **Issue 7**53662 The *pthread_spin_unlock()* function is moved from the Spin Locks option to the Base.

53663 **NAME**
53664 pthread_testcancel — set cancelability state

53665 **SYNOPSIS**
53666 #include <pthread.h>
53667 void pthread_testcancel(void);

53668 **DESCRIPTION**
53669 Refer to *pthread_setcancelstate()*.



53670 **NAME**

53671 ptsname — get name of the slave pseudo-terminal device

53672 **SYNOPSIS**

```
53673 XSI #include <stdlib.h>
53674 char *ptsname(int fildes);
```

53675 **DESCRIPTION**

53676 The *ptsname()* function shall return the name of the slave pseudo-terminal device associated
 53677 with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the
 53678 master device. The *ptsname()* function shall return a pointer to a string containing the pathname
 53679 of the corresponding slave device.

53680 The *ptsname()* function need not be thread-safe. A function that is not required to be thread-safe
 53681 is not required to be reentrant.

53682 **RETURN VALUE**

53683 Upon successful completion, *ptsname()* shall return a pointer to a string which is the name of the
 53684 pseudo-terminal slave device. Upon failure, *ptsname()* shall return a null pointer. This could
 53685 occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file
 53686 system.

53687 **ERRORS**

53688 No errors are defined.

53689 **EXAMPLES**

53690 None.

53691 **APPLICATION USAGE**53692 The value returned may point to a static data area that is overwritten by each call to *ptsname()*.53693 **RATIONALE**

53694 None.

53695 **FUTURE DIRECTIONS**

53696 None.

53697 **SEE ALSO**53698 *grantpt()*, *open()*, *ttyname()*, *unlockpt()*

53699 XBD <stdlib.h>

53700 **CHANGE HISTORY**

53701 First released in Issue 4, Version 2.

53702 **Issue 5**

53703 Moved from X/OPEN UNIX extension to BASE.

53704 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

53705 **NAME**

53706 putc — put a byte on a stream

53707 **SYNOPSIS**

53708 #include <stdio.h>

53709 int putc(int *c*, FILE **stream*);53710 **DESCRIPTION**53711 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
53712 conflict between the requirements described here and the ISO C standard is unintentional. This
53713 volume of POSIX.1-200x defers to the ISO C standard.53714 The *putc()* function shall be equivalent to *fputc()*, except that if it is implemented as a macro it
53715 may evaluate *stream* more than once, so the argument should never be an expression with side
53716 effects.53717 **RETURN VALUE**53718 Refer to *fputc()*.53719 **ERRORS**53720 Refer to *fputc()*.53721 **EXAMPLES**

53722 None.

53723 **APPLICATION USAGE**53724 Since it may be implemented as a macro, *putc()* may treat a *stream* argument with side effects
53725 incorrectly. In particular, *putc(c,*f++)* does not necessarily work correctly. Therefore, use of this
53726 function is not recommended in such situations; *fputc()* should be used instead.53727 **RATIONALE**

53728 None.

53729 **FUTURE DIRECTIONS**

53730 None.

53731 **SEE ALSO**53732 *fputc()*

53733 XBD <stdio.h>

53734 **CHANGE HISTORY**

53735 First released in Issue 1. Derived from Issue 1 of the SVID.

putc_unlocked()

53736 **NAME**
53737 putc_unlocked — stdio with explicit client locking

53738 **SYNOPSIS**
53739 CX #include <stdio.h>
53740 int putc_unlocked(int *c*, FILE **stream*);

53741 **DESCRIPTION**
53742 Refer to [getc_unlocked\(\)](#).

53743
53744
53745
53746
53747
53748
53749
53750
53751
53752
53753
53754
53755
53756
53757
53758
53759
53760
53761
53762
53763
53764
53765
53766
53767
53768
53769

NAME

putchar — put a byte on a stdout stream

SYNOPSIS

```
#include <stdio.h>

int putchar(int c);
```

DESCRIPTION

CX The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

The function call *putchar(c)* shall be equivalent to *putc(c,stdout)*.

RETURN VALUE

Refer to *fputc()*.

ERRORS

Refer to *fputc()*.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

putc()
XBD [<stdio.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

putchar_unlocked()*System Interfaces*

53770 **NAME**
53771 putchar_unlocked — stdio with explicit client locking

SYNOPSIS

53772 CX `#include <stdio.h>`
53773 `int putchar_unlocked(int c);`

DESCRIPTION

53775 Refer to [getc_unlocked\(\)](#).
53776

53777 **NAME**
 53778 putenv — change or add a value to an environment

53779 SYNOPSIS

```
53780 XSI #include <stdlib.h>
53781 int putenv(char *string);
```

53782 DESCRIPTION

53783 The *putenv()* function shall use the *string* argument to set environment variable values. The
 53784 *string* argument should point to a string of the form "*name=value*". The *putenv()* function shall
 53785 make the value of the environment variable *name* equal to *value* by altering an existing variable
 53786 or creating a new one. In either case, the string pointed to by *string* shall become part of the
 53787 environment, so altering the string shall change the environment. The space used by *string* is no
 53788 longer used once a new string which defines *name* is passed to *putenv()*.

53789 The *putenv()* function need not be thread-safe. A function that is not required to be thread-safe is
 53790 not required to be reentrant.

53791 RETURN VALUE

53792 Upon successful completion, *putenv()* shall return 0; otherwise, it shall return a non-zero value
 53793 and set *errno* to indicate the error.

53794 ERRORS

53795 The *putenv()* function may fail if:
 53796 [ENOMEM] Insufficient memory was available.

53797 EXAMPLES

53798 Changing the Value of an Environment Variable

53799 The following example changes the value of the *HOME* environment variable to the value
 53800 */usr/home*.

```
53801 #include <stdlib.h>
53802 ...
53803 static char *var = "HOME=/usr/home";
53804 int ret;
53805 ret = putenv(var);
```

53806 APPLICATION USAGE

53807 The *putenv()* function manipulates the environment pointed to by *environ*, and can be used in
 53808 conjunction with *getenv()*.

53809 See *exec*, for restrictions on changing the environment in multi-threaded applications.

53810 This routine may use *malloc()* to enlarge the environment.

53811 A potential error is to call *putenv()* with an automatic variable as the argument, then return from
 53812 the calling function while *string* is still part of the environment.

53813 The *setenv()* function is preferred over this function.

putenv()53814
53815
53816

53817
53818

53819
53820
53821

53822
53823

53824
53825
53826

53827

53828
53829
53830
53831**RATIONALE**

The standard developers noted that *putenv()* is the only function available to add to the environment without permitting memory leaks.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getenv()*, *malloc()*, *setenv()*

XBD [<stdlib.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The type of the argument to this function is changed from **const char *** to **char ***. This was indicated as a FUTURE DIRECTION in previous issues.

A note indicating that this function need not be reentrant is added to the DESCRIPTION.

Issue 6

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/48 is applied, clarifying wording in the DESCRIPTION and adding a new paragraph into APPLICATION USAGE referring readers to *exec*.

53832 **NAME**53833 putmsg, putpmsg — send a message on a STREAM (**STREAMS**)53834 **SYNOPSIS**

```
53835 OB XSR #include <stropts.h>
53836
53836 int putmsg(int fildev, const struct strbuf *ctlptr,
53837           const struct strbuf *dataptr, int flags);
53838 int putpmsg(int fildev, const struct strbuf *ctlptr,
53839           const struct strbuf *dataptr, int band, int flags);
```

53840 **DESCRIPTION**

53841 The *putmsg()* function shall create a message from a process buffer(s) and send the message to a
 53842 STREAMS file. The message may contain either a data part, a control part, or both. The data and
 53843 control parts are distinguished by placement in separate buffers, as described below. The
 53844 semantics of each part are defined by the STREAMS module that receives the message.

53845 The *putpmsg()* function is equivalent to *putmsg()*, except that the process can send messages in
 53846 different priority bands. Except where noted, all requirements on *putmsg()* also pertain to
 53847 *putpmsg()*.

53848 The *fildev* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and
 53849 *dataptr* arguments each point to a **strbuf** structure.

53850 The *ctlptr* argument points to the structure describing the control part, if any, to be included in
 53851 the message. The *buf* member in the **strbuf** structure points to the buffer where the control
 53852 information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen*
 53853 member is not used by *putmsg()*. In a similar manner, the argument *dataptr* specifies the data, if
 53854 any, to be included in the message. The *flags* argument indicates what type of message should be
 53855 sent and is described further below.

53856 To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer
 53857 and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the
 53858 application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part
 53859 shall be sent if either *dataptr(ctlptr)* is a null pointer or the *len* member of *dataptr(ctlptr)* is set to
 53860 -1.

53861 For *putmsg()*, if a control part is specified and *flags* is set to RS_HIPRI, a high priority message
 53862 shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg()* shall fail and
 53863 set *errno* to [EINVAL]. If *flags* is set to 0, a normal message (priority band equal to 0) shall be
 53864 sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be
 53865 sent and 0 shall be returned.

53866 For *putpmsg()*, the flags are different. The *flags* argument is a bitmask with the following
 53867 mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg()*
 53868 shall fail and set *errno* to [EINVAL]. If a control part is specified and *flags* is set to MSG_HIPRI
 53869 and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either
 53870 no control part is specified or *band* is set to a non-zero value, *putpmsg()* shall fail and set *errno* to
 53871 [EINVAL]. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band
 53872 specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND,
 53873 no message shall be sent and 0 shall be returned.

53874 The *putmsg()* function shall block if the STREAM write queue is full due to internal flow control
 53875 conditions, with the following exceptions:

putmsg()

- 53876 • For high-priority messages, *putmsg()* shall not block on this condition and continues
53877 processing the message.
- 53878 • For other messages, *putmsg()* shall not block but shall fail when the write queue is full and
53879 O_NONBLOCK is set.

53880 The *putmsg()* function shall also block, unless prevented by lack of internal resources, while
53881 waiting for the availability of message blocks in the STREAM, regardless of priority or whether
53882 O_NONBLOCK has been specified. No partial message shall be sent.

RETURN VALUE

53883 Upon successful completion, *putmsg()* and *putpmsg()* shall return 0; otherwise, they shall return
53884 -1 and set *errno* to indicate the error.
53885

ERRORS

53886 The *putmsg()* and *putpmsg()* functions shall fail if:
53887

53888 [EAGAIN] A non-priority message was specified, the O_NONBLOCK flag is set, and the
53889 STREAM write queue is full due to internal flow control conditions; or buffers
53890 could not be allocated for the message that was to be created.

53891 [EBADF] *fildev* is not a valid file descriptor open for writing.

53892 [EINTR] A signal was caught during *putmsg()*.

53893 [EINVAL] An undefined value is specified in *flags*, or *flags* is set to RS_HIPRI or
53894 MSG_HIPRI and no control part is supplied, or the STREAM or multiplexer
53895 referenced by *fildev* is linked (directly or indirectly) downstream from a
53896 multiplexer, or *flags* is set to MSG_HIPRI and *band* is non-zero (for *putpmsg()*
53897 only).

53898 [ENOSR] Buffers could not be allocated for the message that was to be created due to
53899 insufficient STREAMS memory resources.

53900 [ENOSTR] A STREAM is not associated with *fildev*.

53901 [ENXIO] A hangup condition was generated downstream for the specified STREAM.

53902 [EPIPE] or [EIO] The *fildev* argument refers to a STREAMS-based pipe and the other end of the
53903 pipe is closed. A SIGPIPE signal is generated for the calling thread.

53904 [ERANGE] The size of the data part of the message does not fall within the range
53905 specified by the maximum and minimum packet sizes of the topmost
53906 STREAM module. This value is also returned if the control part of the message
53907 is larger than the maximum configured size of the control part of a message,
53908 or if the data part of a message is larger than the maximum configured size of
53909 the data part of a message.

53910 In addition, *putmsg()* and *putpmsg()* shall fail if the STREAM head had processed an
53911 asynchronous error before the call. In this case, the value of *errno* does not reflect the result of
53912 *putmsg()* or *putpmsg()*, but reflects the prior error.

53913 **EXAMPLES**53914 **Sending a High-Priority Message**

53915 The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg()* does the
 53916 following:

- 53917 1. Creates a high-priority message with a control part and a data part, using the buffers
 53918 pointed to by *ctrlbuf* and *databuf*, respectively.
- 53919 2. Sends the message to the STREAMS file identified by *fd*.

```

53920 #include <stropts.h>
53921 #include <string.h>
53922 ...
53923 int fd;
53924 char *ctrlbuf = "This is the control part";
53925 char *databuf = "This is the data part";
53926 struct strbuf ctrl;
53927 struct strbuf data;
53928 int ret;

53929 ctrl.buf = ctrlbuf;
53930 ctrl.len = strlen(ctrlbuf);

53931 data.buf = databuf;
53932 data.len = strlen(databuf);

53933 ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);

```

53934 **Using putpmsg()**

53935 This example has the same effect as the previous example. In this example, however, the
 53936 *putpmsg()* function creates and sends the message to the STREAMS file.

```

53937 #include <stropts.h>
53938 #include <string.h>
53939 ...
53940 int fd;
53941 char *ctrlbuf = "This is the control part";
53942 char *databuf = "This is the data part";
53943 struct strbuf ctrl;
53944 struct strbuf data;
53945 int ret;

53946 ctrl.buf = ctrlbuf;
53947 ctrl.len = strlen(ctrlbuf);

53948 data.buf = databuf;
53949 data.len = strlen(databuf);

53950 ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);

```

53951 **APPLICATION USAGE**

53952 None.

53953 **RATIONALE**

53954 None.

53955

FUTURE DIRECTIONS

53956

The *putmsg()* and *putpmsg()* functions may be removed in a future version.

53957

SEE ALSO

53958

[Section 2.6](#) (on page 473), *getmsg()*, *poll()*, *read*, *write*

53959

XBD [<stropts.h>](#)

53960

CHANGE HISTORY

53961

First released in Issue 4, Version 2.

53962

Issue 5

53963

Moved from X/OPEN UNIX extension to BASE.

53964

53965

The following text is removed from the DESCRIPTION: “The STREAM head guarantees that the control part of a message generated by *putmsg()* is at least 64 bytes in length”.

53966

Issue 6

53967

This function is marked as part of the XSI STREAMS Option Group.

53968

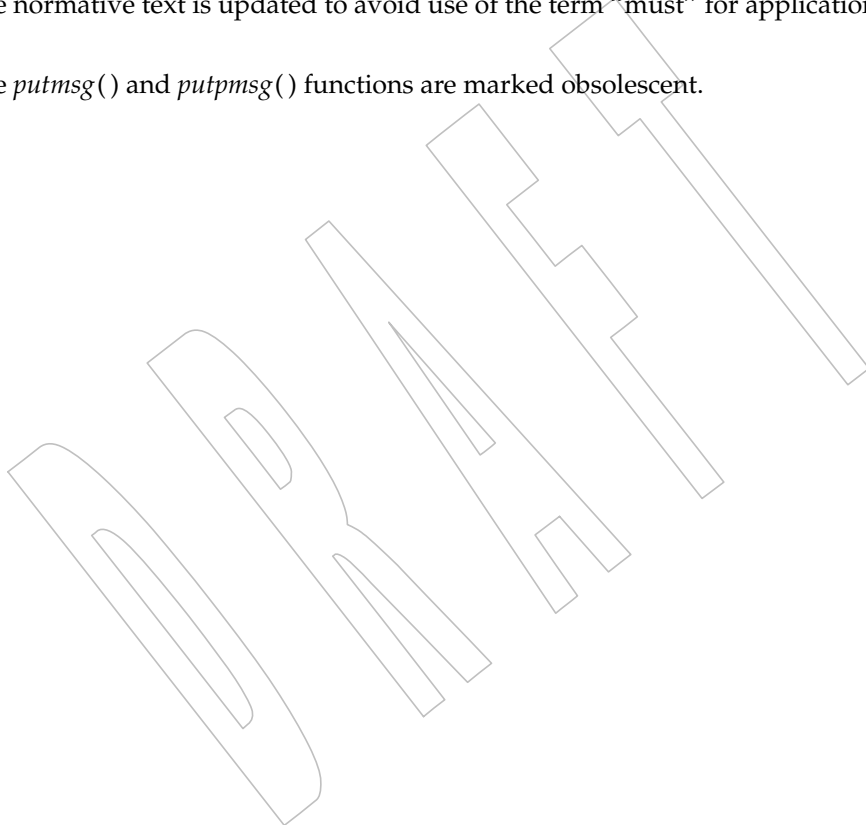
The normative text is updated to avoid use of the term “must” for application requirements.

53969

Issue 7

53970

The *putmsg()* and *putpmsg()* functions are marked obsolescent.



53971 **NAME**
 53972 puts — put a string on standard output

53973 **SYNOPSIS**
 53974 #include <stdio.h>

53975 int puts(const char *s);

53976 DESCRIPTION

53977 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 53978 conflict between the requirements described here and the ISO C standard is unintentional. This
 53979 volume of POSIX.1-200x defers to the ISO C standard.

53980 The *puts()* function shall write the string pointed to by *s*, followed by a <newline>, to the
 53981 standard output stream *stdout*. The terminating null byte shall not be written.

53982 CX The last data modification and last file status change timestamps of the file shall be marked for
 53983 update between the successful execution of *puts()* and the next successful completion of a call to
 53984 *fflush()* or *fclose()* on the same stream or a call to *exit()* or *abort()*.

53985 RETURN VALUE

53986 Upon successful completion, *puts()* shall return a non-negative number. Otherwise, it shall
 53987 CX return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

53988 ERRORS

53989 Refer to *fputc()*.

53990 EXAMPLES

53991 Printing to Standard Output

53992 The following example gets the current time, converts it to a string using *localtime()* and
 53993 *asctime()*, and prints it to standard output using *puts()*. It then prints the number of minutes to
 53994 an event for which it is waiting.

```
53995 #include <time.h>
53996 #include <stdio.h>
53997 ...
53998 time_t now;
53999 int minutes_to_event;
54000 ...
54001 time(&now);
54002 printf("The time is ");
54003 puts(asctime(localtime(&now)));
54004 printf("There are %d minutes to the event.\n",
54005        minutes_to_event);
54006 ...
```

54007 APPLICATION USAGE

54008 The *puts()* function appends a <newline>, while *fputs()* does not.

54009 RATIONALE

54010 None.

puts()54011 **FUTURE DIRECTIONS**

54012 None.

54013 **SEE ALSO**54014 *fopen()*, *fputs()*, *putc()*54015 XBD [<stdio.h>](#) |54016 **CHANGE HISTORY**

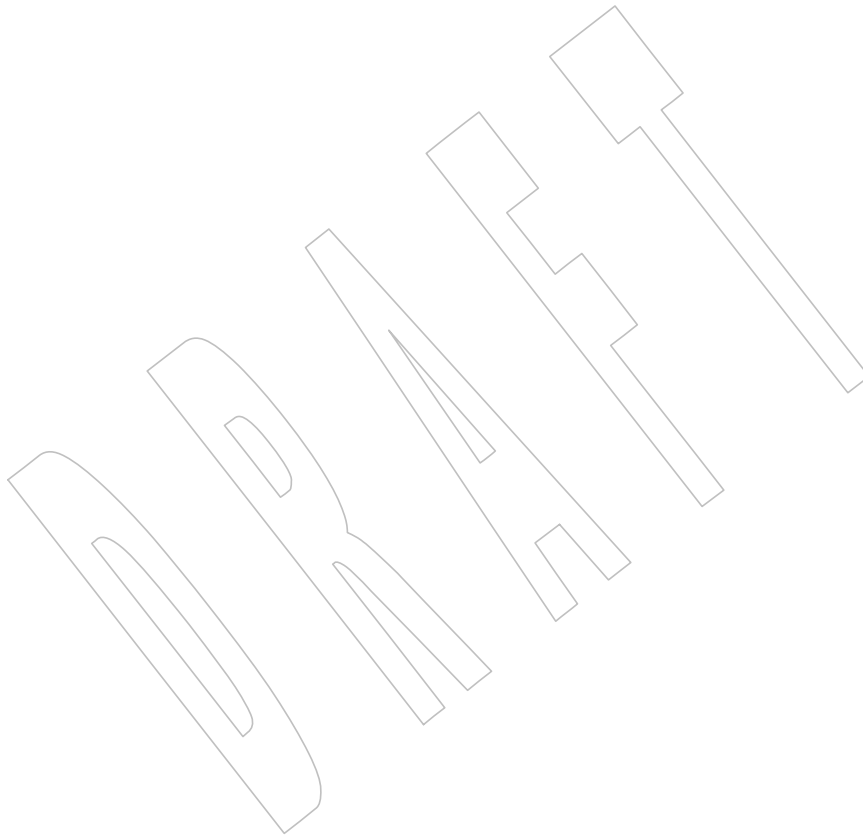
54017 First released in Issue 1. Derived from Issue 1 of the SVID.

54018 **Issue 6**

54019 Extensions beyond the ISO C standard are marked.

54020 **Issue 7**

54021 Changes are made related to support for finegrained timestamps. +



54022 **NAME**
54023 pututxline — put an entry into the user accounting database

54024 **SYNOPSIS**

54025 XSI `#include <utmpx.h>`
54026 `struct utmpx *pututxline(const struct utmpx *utmpx);`

54027 **DESCRIPTION**

54028 Refer to *endutxent()*.

54029 **NAME**

54030 putwc — put a wide character on a stream

54031 **SYNOPSIS**

54032 #include <stdio.h>

54033 #include <wchar.h>

54034 wint_t putwc(wchar_t *wc*, FILE **stream*);54035 **DESCRIPTION**

54036 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 54037 conflict between the requirements described here and the ISO C standard is unintentional. This
 54038 volume of POSIX.1-200x defers to the ISO C standard.

54039 The *putwc()* function shall be equivalent to *fputwc()*, except that if it is implemented as a macro
 54040 it may evaluate *stream* more than once, so the argument should never be an expression with side
 54041 effects.

54042 **RETURN VALUE**54043 Refer to *fputwc()*.54044 **ERRORS**54045 Refer to *fputwc()*.54046 **EXAMPLES**

54047 None.

54048 **APPLICATION USAGE**

54049 Since it may be implemented as a macro, *putwc()* may treat a *stream* argument with side effects
 54050 incorrectly. In particular, *putwc(wc,*f++)* need not work correctly. Therefore, use of this function
 54051 is not recommended; *fputwc()* should be used instead.

54052 **RATIONALE**

54053 None.

54054 **FUTURE DIRECTIONS**

54055 None.

54056 **SEE ALSO**54057 *fputwc()*

54058 XBD <stdio.h>, <wchar.h>

54059 **CHANGE HISTORY**

54060 First released as a World-wide Portability Interface in Issue 4.

54061 **Issue 5**

54062 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
 54063 is changed from **wint_t** to **wchar_t**.

54064 The Optional Header (OH) marking is removed from <stdio.h>.

54065 **NAME**
 54066 putwchar — put a wide character on a stdout stream

54067 **SYNOPSIS**
 54068 #include <wchar.h>

54069 wint_t putwchar(wchar_t wc);

54070 **DESCRIPTION**

54071 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 54072 conflict between the requirements described here and the ISO C standard is unintentional. This
 54073 volume of POSIX.1-200x defers to the ISO C standard.

54074 The function call *putwchar(wc)* shall be equivalent to *putwc(wc,stdout)*.

54075 **RETURN VALUE**

54076 Refer to *fputwc()*.

54077 **ERRORS**

54078 Refer to *fputwc()*.

54079 **EXAMPLES**

54080 None.

54081 **APPLICATION USAGE**

54082 None.

54083 **RATIONALE**

54084 None.

54085 **FUTURE DIRECTIONS**

54086 None.

54087 **SEE ALSO**

54088 *fputwc()*, *putwc()*

54089 XBD <wchar.h>

54090 **CHANGE HISTORY**

54091 First released in Issue 4.

54092 **Issue 5**

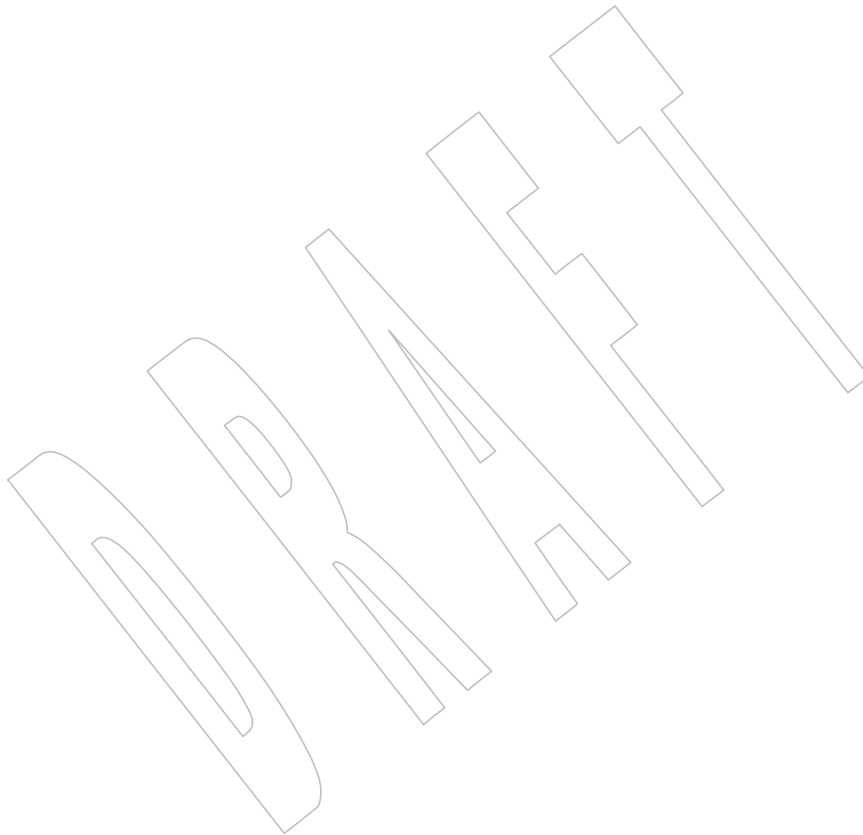
54093 Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of argument *wc*
 54094 is changed from **wint_t** to **wchar_t**.

54095 **NAME**
54096 pwrite — write on a file

54097 **SYNOPSIS**
54098 #include <unistd.h>

54099 ssize_t pwrite(int *fildes*, const void **buf*, size_t *nbyte*,
54100 off_t *offset*);

54101 **DESCRIPTION**
54102 Refer to *write*.



54103 **NAME**

54104 qsort — sort a table of data

54105 **SYNOPSIS**

54106 #include <stdlib.h>

54107 void qsort(void *base, size_t nel, size_t width,
54108 int (*compar)(const void *, const void *));54109 **DESCRIPTION**54110 CX The functionality described on this reference page is aligned with the ISO C standard. Any
54111 conflict between the requirements described here and the ISO C standard is unintentional. This
54112 volume of POSIX.1-200x defers to the ISO C standard.54113 The *qsort()* function shall sort an array of *nel* objects, the initial element of which is pointed to by
54114 *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has
54115 the value zero, the comparison function pointed to by *compar* shall not be called and no
54116 rearrangement shall take place.54117 The application shall ensure that the comparison function pointed to by *compar* does not alter the
54118 contents of the array. The implementation may reorder elements of the array between calls to the
54119 comparison function, but shall not alter the contents of any individual element.54120 When the same objects (consisting of *width* bytes, irrespective of their current positions in the
54121 array) are passed more than once to the comparison function, the results shall be consistent with
54122 one another. That is, they shall define a total ordering on the array.54123 The contents of the array shall be sorted in ascending order according to a comparison function.
54124 The *compar* argument is a pointer to the comparison function, which is called with two
54125 arguments that point to the elements being compared. The application shall ensure that the
54126 function returns an integer less than, equal to, or greater than 0, if the first argument is
54127 considered respectively less than, equal to, or greater than the second. If two members compare
54128 as equal, their order in the sorted array is unspecified.54129 **RETURN VALUE**54130 The *qsort()* function shall not return a value.54131 **ERRORS**

54132 No errors are defined.

54133 **EXAMPLES**

54134 None.

54135 **APPLICATION USAGE**54136 The comparison function need not compare every byte, so arbitrary data may be contained in
54137 the elements in addition to the values being compared.54138 **RATIONALE**54139 The requirement that each argument (hereafter referred to as *p*) to the comparison function is a
54140 pointer to elements of the array implies that for every call, for each argument separately, all of
54141 the following expressions are non-zero:54142 ((char *)p - (char *)base) % width == 0
54143 (char *)p >= (char *)base
54144 (char *)p < (char *)base + nel * width

54145 **FUTURE DIRECTIONS**

54146 None.

54147 **SEE ALSO**54148 *alphasort()*

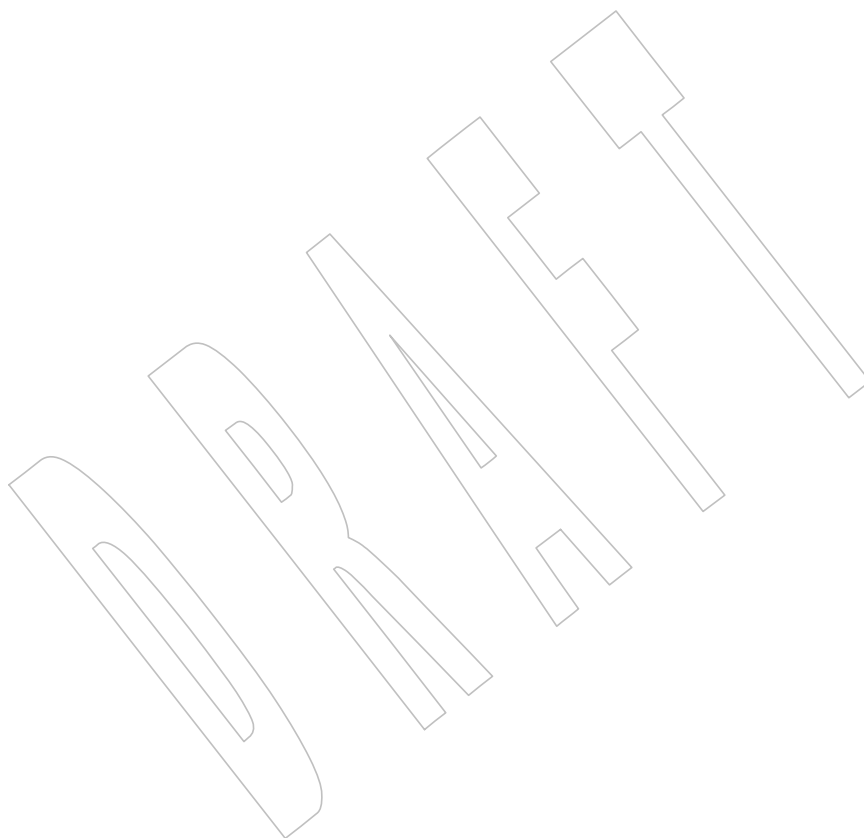
54149 XBD <stdlib.h>

54150 **CHANGE HISTORY**

54151 First released in Issue 1. Derived from Issue 1 of the SVID.

54152 **Issue 6**

54153 The normative text is updated to avoid use of the term “must” for application requirements.

54154 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/49 is applied, adding the last sentence to
54155 the first non-shaded paragraph in the DESCRIPTION, and the following two paragraphs. The
54156 RATIONALE is also updated. These changes are for alignment with the ISO C standard.

54157 **NAME**

54158 raise — send a signal to the executing process

54159 **SYNOPSIS**

54160 #include <signal.h>

54161 int raise(int sig);

54162 **DESCRIPTION**54163 CX The functionality described on this reference page is aligned with the ISO C standard. Any
54164 conflict between the requirements described here and the ISO C standard is unintentional. This
54165 volume of POSIX.1-200x defers to the ISO C standard.54166 CX The *raise()* function shall send the signal *sig* to the executing thread or process. If a signal
54167 handler is called, the *raise()* function shall not return until after the signal handler does.54168 CX The effect of the *raise()* function shall be equivalent to calling:

54169 pthread_kill(pthread_self(), sig);

54170 **RETURN VALUE**54171 CX Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned
54172 and *errno* shall be set to indicate the error.54173 **ERRORS**54174 The *raise()* function shall fail if:54175 CX [EINVAL] The value of the *sig* argument is an invalid signal number.54176 **EXAMPLES**

54177 None.

54178 **APPLICATION USAGE**

54179 None.

54180 **RATIONALE**

54181 The term “thread” is an extension to the ISO C standard.

54182 **FUTURE DIRECTIONS**

54183 None.

54184 **SEE ALSO**54185 *kill*, *sigaction()*

54186 XBD <signal.h>, <sys/types.h>

54187 **CHANGE HISTORY**

54188 First released in Issue 4. Derived from the ANSI C standard.

54189 **Issue 5**

54190 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

54191 **Issue 6**

54192 Extensions beyond the ISO C standard are marked.

54193 The following new requirements on POSIX implementations derive from alignment with the
54194 Single UNIX Specification:

raise()

54195

- In the RETURN VALUE section, the requirement to set *errno* on error is added.

54196

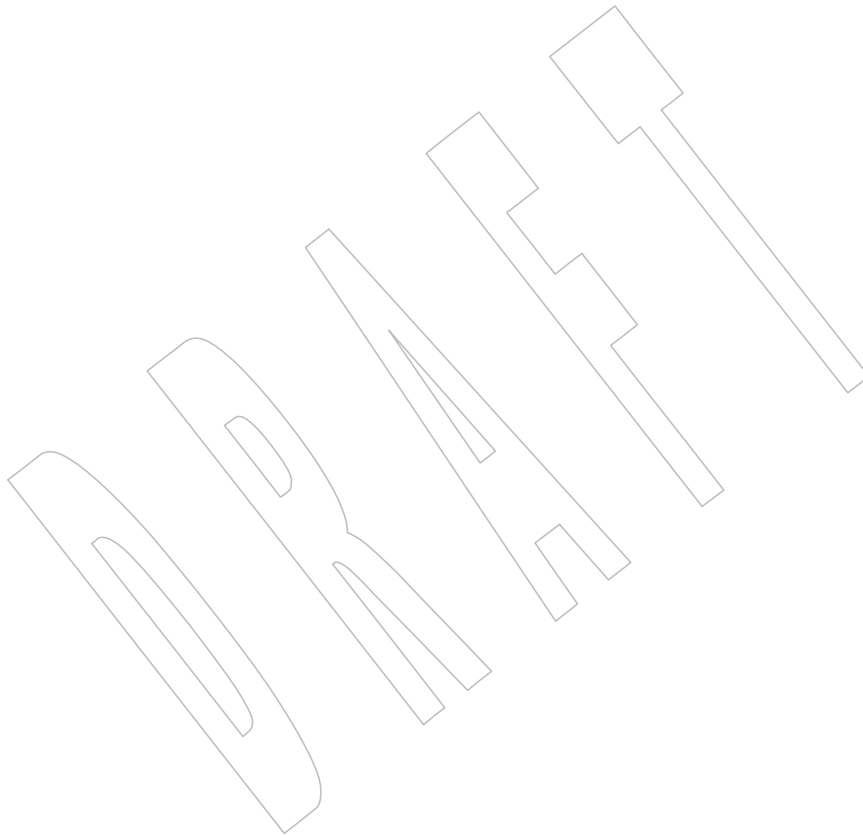
- The [EINVAL] error condition is added.

54197

Issue 7

54198

Functionality relating to the Threads option is moved to the Base.



54199 **NAME**
 54200 rand, rand_r, srand — pseudo-random number generator

54201 **SYNOPSIS**

54202 #include <stdlib.h>
 54203 int rand(void);
 54204 OB CX int rand_r(unsigned *seed);
 54205 void srand(unsigned seed);

54206 **DESCRIPTION**

54207 CX For *rand()* and *srand()*: The functionality described on this reference page is aligned with the
 54208 ISO C standard. Any conflict between the requirements described here and the ISO C standard is
 54209 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

54210 The *rand()* function shall compute a sequence of pseudo-random integers in the range
 54211 XSI [0,{RAND_MAX}] with a period of at least 2^{32} .

54212 CX The *rand()* function need not be thread-safe. A function that is not required to be thread-safe is
 54213 not required to be reentrant.

54214 OB CX The *rand_r()* function shall compute a sequence of pseudo-random integers in the range
 54215 [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro shall be at least 32767.)

54216 If *rand_r()* is called with the same initial value for the object pointed to by *seed* and that object is
 54217 not modified between successive returns and calls to *rand_r()*, the same sequence shall be
 54218 generated.

54219 The *srand()* function uses the argument as a seed for a new sequence of pseudo-random
 54220 numbers to be returned by subsequent calls to *rand()*. If *srand()* is then called with the same
 54221 seed value, the sequence of pseudo-random numbers shall be repeated. If *rand()* is called before
 54222 any calls to *srand()* are made, the same sequence shall be generated as when *srand()* is first
 54223 called with a seed value of 1.

54224 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 54225 *rand()* or *srand()*.

54226 **RETURN VALUE**

54227 The *rand()* function shall return the next pseudo-random number in the sequence.

54228 OB CX The *rand_r()* function shall return a pseudo-random integer.

54229 The *srand()* function shall not return a value.

54230 **ERRORS**

54231 No errors are defined.

54232 **EXAMPLES**

54233 **Generating a Pseudo-Random Number Sequence**

54234 The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
54235 #include <stdio.h>
54236 #include <stdlib.h>
54237 ...
54238     long count, i;
54239     char *keystri;
54240     int elementlen, len;
54241     char c;
```

```

54242     ...
54243     /* Initial random number generator. */
54244         srand(1);

54245         /* Create keys using only lowercase characters */
54246         len = 0;
54247         for (i=0; i<count; i++) {
54248             while (len < elementlen) {
54249                 c = (char) (rand() % 128);
54250                 if (islower(c))
54251                     keystr[len++] = c;
54252             }

54253             keystr[len] = '\0';
54254             printf("%s Element%0*ld\n", keystr, elementlen, i);
54255             len = 0;
54256         }

```

54257 **Generating the Same Sequence on Different Machines**

54258 The following code defines a pair of functions that could be incorporated into applications
54259 wishing to ensure that the same sequence of numbers is generated across different machines.

```

54260     static unsigned long next = 1;
54261     int myrand(void) /* RAND_MAX assumed to be 32767. */
54262     {
54263         next = next * 1103515245 + 12345;
54264         return((unsigned)(next/65536) % 32768);
54265     }

54266     void myrand(unsigned seed)
54267     {
54268         next = seed;
54269     }

```

54270 **APPLICATION USAGE**

54271 The *drand48()* function provides a much more elaborate random number generator.

54272 The limitations on the amount of state that can be carried between one function call and another
54273 mean the *rand_r()* function can never be implemented in a way which satisfies all of the
54274 requirements on a pseudo-random number generator. Therefore this function should be avoided
54275 whenever non-trivial requirements (including safety) have to be fulfilled.

54276 **RATIONALE**

54277 The ISO C standard *rand()* and *srand()* functions allow per-process pseudo-random streams
54278 shared by all threads. Those two functions need not change, but there has to be mutual-
54279 exclusion that prevents interference between two threads concurrently accessing the random
54280 number generator.

54281 With regard to *rand()*, there are two different behaviors that may be wanted in a multi-threaded
54282 program:

- 54283 1. A single per-process sequence of pseudo-random numbers that is shared by all threads
54284 that call *rand()*
- 54285 2. A different sequence of pseudo-random numbers for each thread that calls *rand()*

54286 This is provided by the modified thread-safe function based on whether the seed value is global
54287 to the entire process or local to each thread.

54288 This does not address the known deficiencies of the *rand()* function implementations, which
 54289 have been approached by maintaining more state. In effect, this specifies new thread-safe forms
 54290 of a deficient function.

54291 FUTURE DIRECTIONS

54292 The *rand_r()* function may be removed in a future version. |

54293 SEE ALSO

54294 *drand48()*

54295 XBD <stdlib.h> |

54296 CHANGE HISTORY

54297 First released in Issue 1. Derived from Issue 1 of the SVID.

54298 Issue 5

54299 The *rand_r()* function is included for alignment with the POSIX Threads Extension.

54300 A note indicating that the *rand()* function need not be reentrant is added to the DESCRIPTION.

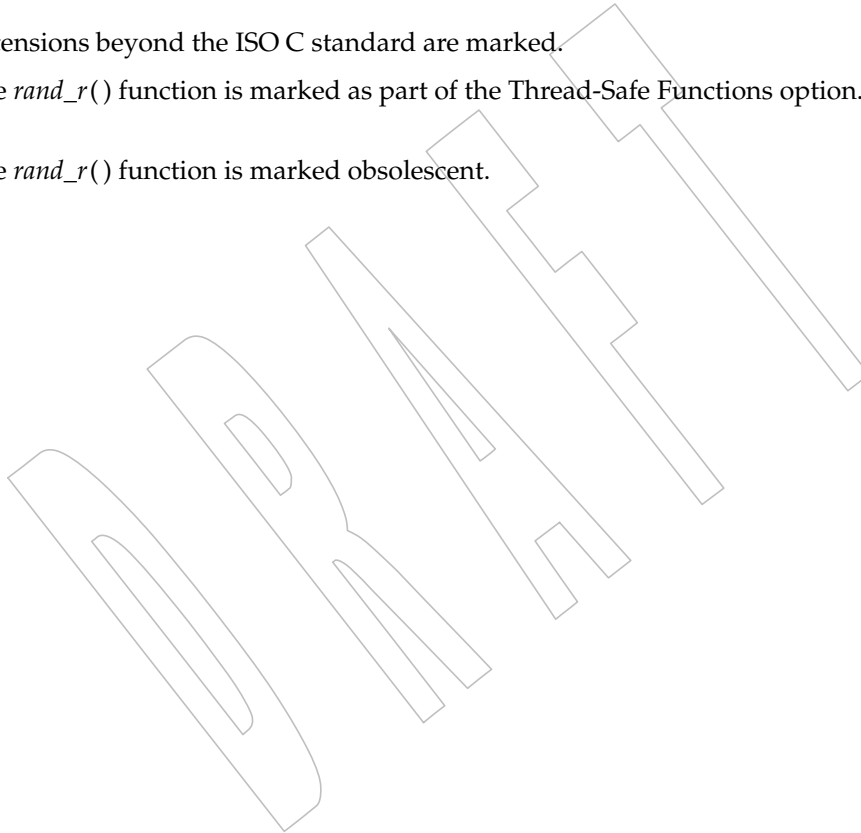
54301 Issue 6

54302 Extensions beyond the ISO C standard are marked.

54303 The *rand_r()* function is marked as part of the Thread-Safe Functions option.

54304 Issue 7

54305 The *rand_r()* function is marked obsolescent. |



random()

54306

NAME

54307

random — generate pseudo-random number

54308

SYNOPSIS

54309

XSI `#include <stdlib.h>`

54310

`long random(void);`

54311

DESCRIPTION

54312

Refer to *initstate()*.

54313 **NAME**54314 `pread, read` — read from a file54315 **SYNOPSIS**54316 `#include <unistd.h>`54317 `ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);`54318 `ssize_t read(int fildes, void *buf, size_t nbyte);`54319 **DESCRIPTION**54320 The `read()` function shall attempt to read *nbyte* bytes from the file associated with the open file
54321 descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on
54322 the same pipe, FIFO, or terminal device is unspecified.54323 Before any action described below is taken, and if *nbyte* is zero, the `read()` function may detect
54324 and return errors as described below. In the absence of errors, or if error detection is not
54325 performed, the `read()` function shall return zero and have no other results.54326 On files that support seeking (for example, a regular file), the `read()` shall start at a position in
54327 the file given by the file offset associated with *fildes*. The file offset shall be incremented by the
54328 number of bytes actually read.54329 Files that do not support seeking—for example, terminals—always read from the current
54330 position. The value of a file offset associated with such a file is undefined.54331 No data transfer shall occur past the current end-of-file. If the starting position is at or after the
54332 end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent
54333 `read()` requests is implementation-defined.54334 If the value of *nbyte* is greater than `{SSIZE_MAX}`, the result is implementation-defined.

54335 When attempting to read from an empty pipe or FIFO:

- 54336
- If no process has the pipe open for writing, `read()` shall return 0 to indicate end-of-file.
 - 54337 • If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` shall return
54338 `-1` and set *errno* to `[EAGAIN]`.
 - 54339 • If some process has the pipe open for writing and `O_NONBLOCK` is clear, `read()` shall
54340 block the calling thread until some data is written or the pipe is closed by all processes that
54341 had the pipe open for writing.

54342 When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and
54343 has no data currently available:

- 54344
- If `O_NONBLOCK` is set, `read()` shall return `-1` and set *errno* to `[EAGAIN]`.
 - 54345 • If `O_NONBLOCK` is clear, `read()` shall block the calling thread until some data becomes
54346 available.
 - 54347 • The use of the `O_NONBLOCK` flag has no effect if there is some data available.

54348 The `read()` function reads data previously written to a file. If any portion of a regular file prior to
54349 the end-of-file has not been written, `read()` shall return bytes with value 0. For example, `lseek()`
54350 allows the file offset to be set beyond the end of existing data in the file. If data is later written at
54351 this point, subsequent reads in the gap between the previous end of data and the newly written
54352 data shall return bytes with value 0 until data is written into the gap.54353 Upon successful completion, where *nbyte* is greater than 0, `read()` shall mark for update the last
54354 data access timestamp of the file, and shall return the number of bytes read. This number shall
54355 never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes

read()

54356		left in the file is less than <i>nbyte</i> , if the <i>read()</i> request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than <i>nbyte</i> bytes immediately available for reading. For example, a <i>read()</i> from a file associated with a terminal may return one typed line of data.
54357		
54358		
54359		If a <i>read()</i> is interrupted by a signal before it reads any data, it shall return -1 with <i>errno</i> set to [EINTR].
54360		
54361		If a <i>read()</i> is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.
54362		
54363		For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with <i>filides</i> .
54364		
54365		If <i>filides</i> refers to a socket, <i>read()</i> shall be equivalent to <i>recv()</i> with no flags set.
54366	SIO	If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.
54367		
54368		
54369		
54370	SHM	If <i>filides</i> refers to a shared memory object, the result of the <i>read()</i> function is unspecified.
54371	TYM	If <i>filides</i> refers to a typed memory object, the result of the <i>read()</i> function is unspecified.
54372	OB XSR	A <i>read()</i> from a STREAMS file can read data in three different modes: <i>byte-stream</i> mode, <i>message-nondiscard</i> mode, and <i>message-discard</i> mode. The default shall be byte-stream mode. This can be changed using the I_SRDOPT <i>ioctl()</i> request, and can be tested with I_GRDOPT <i>ioctl()</i> . In byte-stream mode, <i>read()</i> shall retrieve data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.
54373		
54374		
54375		
54376		
54377		
54378		In STREAMS message-nondiscard mode, <i>read()</i> shall retrieve data until as many bytes as were requested are transferred, or until a message boundary is reached. If <i>read()</i> does not retrieve all the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by the next <i>read()</i> call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the <i>read()</i> returns shall be discarded, and shall not be available for a subsequent <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> call.
54379		
54380		
54381		
54382		
54383		
54384		
54385		How <i>read()</i> handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, <i>read()</i> shall accept data until it has read <i>nbyte</i> bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The <i>read()</i> function shall then return the number of bytes read, and place the zero-byte message back on the STREAM to be retrieved by the next <i>read()</i> , <i>getmsg()</i> , or <i>getpmsg()</i> . In message-nondiscard mode or message-discard mode, a zero-byte message shall return 0 and the message shall be removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless of the read mode.
54386		
54387		
54388		
54389		
54390		
54391		
54392		
54393		
54394		A <i>read()</i> from a STREAMS file shall return the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.
54395		
54396		By default, STREAMS are in control-normal mode, in which a <i>read()</i> from a STREAMS file can only process messages that contain a data part but do not contain a control part. The <i>read()</i> shall fail if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the I_SRDOPT <i>ioctl()</i> command. In control-data mode, <i>read()</i> shall convert any control part to data and pass it to the application before passing any data part originally present in the same message. In control-discard mode, <i>read()</i> shall discard message control parts but return to the process any data part in the message.
54397		
54398		
54399		
54400		
54401		
54402		
54403		

54404 In addition, `read()` shall fail if the STREAM head had processed an asynchronous error before
 54405 the call. In this case, the value of `errno` shall not reflect the result of `read()`, but reflect the prior
 54406 error. If a hangup occurs on the STREAM being read, `read()` shall continue to operate normally
 54407 until the STREAM head read queue is empty. Thereafter, it shall return 0.

54408 The `pread()` function shall be equivalent to `read()`, except that it shall read from a given position
 54409 in the file without changing the file pointer. The first three arguments to `pread()` are the same as
 54410 `read()` with the addition of a fourth argument `offset` for the desired position inside the file. An
 54411 attempt to perform a `pread()` on a file that is incapable of seeking shall result in an error.

RETURN VALUE

54412 Upon successful completion, these functions shall return a non-negative integer indicating the
 54413 number of bytes actually read. Otherwise, the functions shall return `-1` and set `errno` to indicate
 54414 the error.
 54415

ERRORS

54416 These functions shall fail if:

54417 [EAGAIN] The `O_NONBLOCK` flag is set for the file descriptor and the thread would be
 54418 delayed.
 54419

54420 [EBADF] The `fildev` argument is not a valid file descriptor open for reading.

54421 OB XSR [EBADMSG] The file is a STREAM file that is set to control-normal mode and the message
 54422 waiting to be read includes a control part.

54423 [EINTR] The read operation was terminated due to the receipt of a signal, and no data
 54424 was transferred.

54425 OB XSR [EINVAL] The STREAM or multiplexer referenced by `fildev` is linked (directly or
 54426 indirectly) downstream from a multiplexer.

54427 [EIO] The process is a member of a background process attempting to read from its
 54428 controlling terminal, the process is ignoring or blocking the SIGTTIN signal,
 54429 or the process group is orphaned. This error may also be generated for
 54430 implementation-defined reasons.

54431 XSI [EISDIR] The `fildev` argument refers to a directory and the implementation does not
 54432 allow the directory to be read using `read()` or `pread()`. The `readdir()` function
 54433 should be used instead.

54434 [EOVERFLOW] The file is a regular file, `nbyte` is greater than 0, the starting position is before
 54435 the end-of-file, and the starting position is greater than or equal to the offset
 54436 maximum established in the open file description associated with `fildev`.

54437 The `read()` function shall fail if:

54438 [EAGAIN] or [EWOULDBLOCK]

54439 The file descriptor is for a socket, is marked `O_NONBLOCK`, and no data is
 54440 waiting to be received.

54441 [ECONNRESET] A read was attempted on a socket and the connection was forcibly closed by
 54442 its peer.

54443 [ENOTCONN] A read was attempted on a socket that is not connected.

54444 [ETIMEDOUT] A read was attempted on a socket and a transmission timeout occurred.

54445 These functions may fail if:

54446 [EIO] A physical I/O error has occurred.

54447	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
54448	[ENOMEM]	Insufficient memory was available to fulfill the request.
54449	[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
54450		
54451		The <i>pread()</i> function shall fail, and the file pointer shall remain unchanged, if:
54452	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.
54453	[EOVERFLOW]	The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the file.
54454		
54455	[ENXIO]	A request was outside the capabilities of the device.
54456	[ESPIPE]	<i>fildev</i> is associated with a pipe or FIFO.

EXAMPLES**Reading Data into a Buffer**

The following example reads data from the file associated with the file descriptor *fd* into the buffer pointed to by *buf*.

```

54461 #include <sys/types.h>
54462 #include <unistd.h>
54463 ...
54464 char buf[20];
54465 size_t nbytes;
54466 ssize_t bytes_read;
54467 int fd;
54468 ...
54469 nbytes = sizeof(buf);
54470 bytes_read = read(fd, buf, nbytes);
54471 ...

```

APPLICATION USAGE

None.

RATIONALE

This volume of POSIX.1-200x does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

Note that a *read()* of zero bytes does not modify the last data access timestamp. A *read()* that requests more than zero bytes, but returns zero, is required to modify the last data access timestamp.

Implementations are allowed, but not required, to perform error checking for *read()* requests of zero bytes.

54485

Input and Output54486
54487
54488
54489
54490
54491
54492
54493
54494

The use of I/O with large byte counts has always presented problems. Ideas such as *lread()* and *lwrite()* (using and returning **longs**) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read()* and *write()*. The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of POSIX.1-200x also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

54495
54496
54497

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

54498
54499
54500
54501
54502

This volume of POSIX.1-200x requires that no action be taken for *read()* or *write()* when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-200x, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

54503
54504
54505
54506
54507
54508

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

54509
54510
54511
54512
54513
54514

There were recommendations to add format parameters to *read()* and *write()* in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of POSIX.1-200x. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

54515
54516
54517
54518
54519
54520

In 4.3 BSD, a *read()* or *write()* that is interrupted by a signal before transferring any data does not by default return an [EINTR] error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select()*, whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select()* to be used before *read()* in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

54521
54522

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

54523
54524
54525

There are no references to actions taken following an “unrecoverable error”. It is considered beyond the scope of this volume of POSIX.1-200x to describe what happens in the case of hardware errors.

54526
54527
54528
54529
54530
54531
54532
54533
54534

Earlier versions of this standard allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that POSIX.1-200x should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read()* and *write()* return -1 and set *errno* to [EINTR] when interrupted after some, but not all, of the data requested had been transferred. However, the U.S. Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read()* and *write()* return the number of bytes actually transferred before the interrupt. If -1 is returned when any data is transferred, it is difficult to recover from the error on a seekable device and impossible on a non-seekable device. Most new implementations support this behavior. The

54535 behavior required by POSIX.1-200x is to return the number of bytes transferred.

54536 POSIX.1-200x does not specify when an implementation that buffers *read()*s actually moves the
54537 data into the user-supplied buffer, so an implementation may choose to do this at the latest
54538 possible moment. Therefore, an interrupt arriving earlier may not cause *read()* to return a
54539 partial byte count, but rather to return -1 and set *errno* to [EINTR].

54540 Consideration was also given to combining the two previous options, and setting *errno* to
54541 [EINTR] while returning a short count. However, not only is there no existing practice that
54542 implements this, it is also contradictory to the idea that when *errno* is set, the function
54543 responsible shall return -1 .

54544 FUTURE DIRECTIONS

54545 None.

54546 SEE ALSO

54547 *fcntl()*, *ioctl()*, *lseek()*, *open()*, *pipe()*, *readv()*

54548 XBD Chapter 11 (on page 185), [<stropts.h>](#), [<sys/uio.h>](#), [<unistd.h>](#)

54549 CHANGE HISTORY

54550 First released in Issue 1. Derived from Issue 1 of the SVID.

54551 Issue 5

54552 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
54553 Threads Extension.

54554 Large File Summit extensions are added.

54555 The *pread()* function is added.

54556 Issue 6

54557 The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are
54558 marked as part of the XSI STREAMS Option Group.

54559 The following new requirements on POSIX implementations derive from alignment with the
54560 Single UNIX Specification:

- 54561 • The DESCRIPTION now states that if *read()* is interrupted by a signal after it has
54562 successfully read some data, it returns the number of bytes read. In Issue 3, it was optional
54563 whether *read()* returned the number of bytes read, or whether it returned -1 with *errno* set
54564 to [EINTR]. This is a FIPS requirement.
- 54565 • In the DESCRIPTION, text is added to indicate that for regular files, no data transfer
54566 occurs past the offset maximum established in the open file description associated with
54567 *files*. This change is to support large files.
- 54568 • The [Eoverflow] mandatory error condition is added.
- 54569 • The [ENxio] optional error condition is added.

54570 Text referring to sockets is added to the DESCRIPTION.

54571 The following changes were made to align with the IEEE P1003.1a draft standard:

- 54572 • The effect of reading zero bytes is clarified.

54573 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that
54574 *read()* results are unspecified for typed memory objects.

54575 New RATIONALE is added to explain the atomicity requirements for input and output
54576 operations.

54577 The following error conditions are added for operations on sockets: [EAGAIN],
54578 [ECONNRESET], [ENOTCONN], and [ETIMEDOUT].

54579

The [EIO] error is made optional.

54580

54581

The following error conditions are added for operations on sockets: [ENOBUFS] and [ENOMEM].

54582

The *readv()* function is split out into a separate reference page.

54583

54584

54585

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/108 is applied, updating the [EAGAIN] error in the ERRORS section from “the process would be delayed” to “the thread would be delayed”.

54586

54587

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/109 is applied, making an editorial correction in the RATIONALE section.

54588

Issue 7

54589

The *pread()* function is moved from the XSI option to the Base.

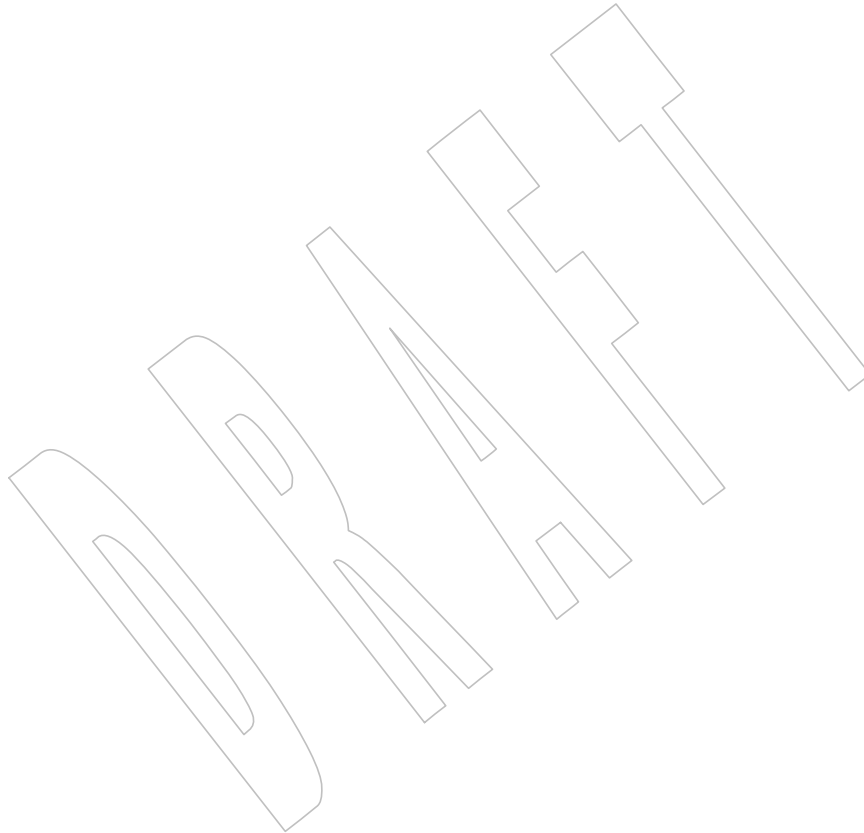
54590

Functionality relating to the XSI STREAMS option is marked obsolescent.

54591

Changes are made related to support for finegrained timestamps.

+



54592 NAME

54593 readdir, readdir_r — read a directory

54594 SYNOPSIS

54595 #include <dirent.h>

54596 struct dirent *readdir(DIR *dirp);

54597 int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,

54598 struct dirent **restrict result);

54599 DESCRIPTION

54600 The type **DIR**, which is defined in the **<dirent.h>** header, represents a *directory stream*, which is
 54601 an ordered sequence of all the directory entries in a particular directory. Directory entries
 54602 represent files; files may be removed from a directory or added to a directory asynchronously to
 54603 the operation of *readdir()*.

54604 The *readdir()* function shall return a pointer to a structure representing the directory entry at the
 54605 current position in the directory stream specified by the argument *dirp*, and position the
 54606 directory stream at the next entry. It shall return a null pointer upon reaching the end of the
 54607 directory stream. The structure **dirent** defined in the **<dirent.h>** header describes a directory
 54608 entry.

54609 The *readdir()* function shall not return directory entries containing empty names. If entries for
 54610 dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-
 54611 dot; otherwise, they shall not be returned.

54612 The pointer returned by *readdir()* points to data which may be overwritten by another call to
 54613 *readdir()* on the same directory stream. This data is not overwritten by another call to *readdir()*
 54614 on a different directory stream.

54615 If a file is removed from or added to the directory after the most recent call to *opendir()* or
 54616 *rewinddir()*, whether a subsequent call to *readdir()* returns an entry for that file is unspecified.

54617 The *readdir()* function may buffer several directory entries per actual read operation; *readdir()* |
 54618 shall mark for update the last data access timestamp of the directory each time the directory is |
 54619 actually read.

54620 After a call to *fork()*, either the parent or child (but not both) may continue processing the
 54621 directory stream using *readdir()*, *rewinddir()*, or *seekdir()*. If both the parent and child processes
 54622 use these functions, the result is undefined.

54623 If the entry names a symbolic link, the value of the *d_ino* member is unspecified.

54624 The *readdir()* function need not be thread-safe. A function that is not required to be thread-safe is
 54625 not required to be reentrant.

54626 The *readdir_r()* function shall initialize the **dirent** structure referenced by *entry* to represent the
 54627 directory entry at the current position in the directory stream referred to by *dirp*, store a pointer
 54628 to this structure at the location referenced by *result*, and position the directory stream at the next
 54629 entry.

54630 The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name*
 54631 members containing at least {NAME_MAX}+1 elements.

54632 Upon successful return, the pointer returned at **result* shall have the same value as the argument
 54633 *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

54634 The *readdir_r()* function shall not return directory entries containing empty names.

54635 If a file is removed from or added to the directory after the most recent call to *opendir()* or

54636 *rewinddir()*, whether a subsequent call to *readdir_r()* returns an entry for that file is unspecified.

54637 The *readdir_r()* function may buffer several directory entries per actual read operation; |
 54638 *readdir_r()* shall mark for update the last data access timestamp of the directory each time the |
 54639 directory is actually read.

54640 Applications wishing to check for error situations should set *errno* to 0 before calling *readdir()*. If
 54641 *errno* is set to non-zero on return, an error occurred.

54642 RETURN VALUE

54643 Upon successful completion, *readdir()* shall return a pointer to an object of type **struct dirent**.
 54644 When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate
 54645 the error. When the end of the directory is encountered, a null pointer shall be returned and
 54646 *errno* is not changed.

54647 If successful, the *readdir_r()* function shall return zero; otherwise, an error number shall be
 54648 returned to indicate the error.

54649 ERRORS

54650 The *readdir()* and *readdir_r()* functions shall fail if:

54651 [EOVERFLOW] One of the values in the structure to be returned cannot be represented
 54652 correctly.

54653 The *readdir()* and *readdir_r()* functions may fail if:

54654 [EBADF] The *dirp* argument does not refer to an open directory stream.

54655 [ENOENT] The current position of the directory stream is invalid.

54656 EXAMPLES

54657 The following sample program searches the current directory for each of the arguments supplied
 54658 on the command line.

```
54659 #include <dirent.h>
54660 #include <errno.h>
54661 #include <stdio.h>
54662 #include <string.h>
54663 static void lookup(const char *arg)
54664 {
54665     DIR *dirp;
54666     struct dirent *dp;
54667     if ((dirp = opendir(".")) == NULL) {
54668         perror("couldn't open '.'");
54669         return;
54670     }
54671     do {
54672         errno = 0;
54673         if ((dp = readdir(dirp)) != NULL) {
54674             if (strcmp(dp->d_name, arg) != 0)
54675                 continue;
54676             (void) printf("found %s\n", arg);
54677             (void) closedir(dirp);
54678             return;
54679         }
54680     } while (dp != NULL);
54681     if (errno != 0)
```

```

54682         perror("error reading directory");
54683     else
54684         (void) printf("failed to find %s\n", arg);
54685     (void) closedir(dirp);
54686     return;
54687 }
54688 int main(int argc, char *argv[])
54689 {
54690     int i;
54691     for (i = 1; i < argc; i++)
54692         lookup(argv[i]);
54693     return (0);
54694 }

```

APPLICATION USAGE

The *readdir()* function should be used in conjunction with *opendir()*, *closedir()*, and *rewinddir()* to examine the contents of the directory.

The *readdir_r()* function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The returned value of *readdir()* merely *represents* a directory entry. No equivalence should be inferred.

Historical implementations of *readdir()* obtain multiple directory entries on a single read operation, which permits subsequent *readdir()* operations to operate from the buffered information. Any wording that required each successful *readdir()* operation to mark the directory last data access timestamp for update would disallow such historical performance-oriented implementations.

Since *readdir()* returns NULL when it detects an error and when the end of the directory is encountered, an application that needs to tell the difference must set *errno* to zero before the call and check it if NULL is returned. Since the function must not change *errno* in the second case and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```

54714 int derror (dirp)
54715 DIR *dirp;
54716
54717 void clearerr (dirp)
54718 DIR *dirp;

```

The first would indicate whether an error had occurred, and the second would clear the error indication. The simpler method involving *errno* was adopted instead by requiring that *readdir()* not change *errno* when end-of-directory is encountered.

An error or signal indicating that a directory has changed while open was considered but rejected.

The thread-safe version of the directory reading function returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call. Either the {NAME_MAX} compile-time constant or the corresponding *pathconf()* option can be used to determine the maximum sizes of returned pathnames.

54727 **FUTURE DIRECTIONS**

54728 None.

54729 **SEE ALSO**54730 *closedir()*, *dirfd()*, *exec*, *fdopendir()*, *fstatat()*, *rewinddir()*, *symlink()*54731 XBD **<dirent.h>**, **<sys/types.h>**54732 **CHANGE HISTORY**

54733 First released in Issue 2.

54734 **Issue 5**

54735 Large File Summit extensions are added.

54736 The *readdir_r()* function is included for alignment with the POSIX Threads Extension.54737 A note indicating that the *readdir()* function need not be reentrant is added to the
54738 DESCRIPTION.54739 **Issue 6**54740 The *readdir_r()* function is marked as part of the Thread-Safe Functions option.54741 The Open Group Corrigendum U026/7 is applied, correcting the prototype for *readdir_r()*.54742 The Open Group Corrigendum U026/8 is applied, clarifying the wording of the successful
54743 return for the *readdir_r()* function.54744 The following new requirements on POSIX implementations derive from alignment with the
54745 Single UNIX Specification:

- 54746 • The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was
54747 required for conforming implementations of previous POSIX specifications, it was not
54748 required for UNIX applications.
- 54749 • A statement is added to the DESCRIPTION indicating the disposition of certain fields in
54750 **struct dirent** when an entry refers to a symbolic link.
- 54751 • The [EOVERFLOW] mandatory error condition is added. This change is to support large
54752 files.
- 54753 • The [ENOENT] optional error condition is added.

54754 The APPLICATION USAGE section is updated to include a note on the thread-safe function and
54755 its avoidance of possibly using a static data area.54756 The **restrict** keyword is added to the *readdir_r()* prototype for alignment with the
54757 ISO/IEC 9899:1999 standard.54758 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/50 is applied, replacing the EXAMPLES
54759 section with a new example.54760 **Issue 7**

54761 Austin Group Interpretation 1003.1-2001 #059 is applied, updating the ERRORS section.

54762 The *readdir_r()* function is moved from the Thread-Safe Functions option to the Base.

54763 Changes are made related to support for finegrained timestamps.

54764 **NAME**

54765 readlink, readlinkat — read the contents of a symbolic link relative to a directory file descriptor

54766 **SYNOPSIS**

54767 #include <unistd.h>

54768 ssize_t readlink(const char *restrict path, char *restrict buf,
54769 size_t bufsize);54770 ssize_t readlinkat(int fd, const char *restrict path,
54771 char *restrict buf, size_t bufsize);54772 **DESCRIPTION**54773 The *readlink()* function shall place the contents of the symbolic link referred to by *path* in the
54774 buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*,
54775 the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to
54776 contain the link content, the first *bufsize* bytes shall be placed in *buf*.54777 If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.54778 The *readlinkat()* function shall be equivalent to the *readlink()* function except in the case where
54779 *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the
54780 directory associated with the file descriptor *fd* instead of the current working directory. It is
54781 unspecified whether directory searches are permitted based on whether the file was opened
54782 with search permission or on the current permissions of the directory underlying the file
54783 descriptor.54784 If *readlinkat()* is passed the special value AT_FDCWD in the *fd* parameter, the current working
54785 directory is used and the behavior shall be identical to a call to *readlink()*.54786 **RETURN VALUE**54787 Upon successful completion, *readlink()* shall return the count of bytes placed in the buffer.
54788 Otherwise, it shall return a value of -1 , leave the buffer unchanged, and set *errno* to indicate the
54789 error.54790 Upon successful completion, the *readlinkat()* function shall return 0. Otherwise, it shall return -1
54791 and set *errno* to indicate the error.54792 **ERRORS**

54793 These functions shall fail if:

54794 [EACCES] Search permission is denied for a component of the path prefix of *path*.54795 [EINVAL] The *path* argument names a file that is not a symbolic link.

54796 [EIO] An I/O error occurred while reading from the file system.

54797 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
54798 argument.

54799 [ENAMETOOLONG]

54800 The length of the *path* argument exceeds {PATH_MAX} or a pathname
54801 component is longer than {NAME_MAX}.54802 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.

54803 [ENOTDIR] A component of the path prefix is not a directory.

54804 The *readlinkat()* function shall fail if:

54805 [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is
 54806 neither AT_FDCWD nor a valid file descriptor open for reading.

54807 These functions may fail if:

54808 [EACCES] Read permission is denied for the symbolic link referred to by *path*.

54809 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
 54810 resolution of the *path* argument.

54811 [ENAMETOOLONG]

54812 As a result of encountering a symbolic link in resolution of the *path* argument,
 54813 the length of the substituted pathname string exceeded {PATH_MAX}.

54814 The *readlinkat*() function may fail if:

54815 [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
 54816 file descriptor associated with a directory.

54817 EXAMPLES

54818 Reading the Name of a Symbolic Link

54819 The following example shows how to read the name of a symbolic link named `/modules/pass1`.

```
54820 #include <unistd.h>
54821 char buf[1024];
54822 ssize_t len;
54823 ...
54824 if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
54825     buf[len] = '\0';
```

54826 APPLICATION USAGE

54827 Conforming applications should not assume that the returned contents of the symbolic link are
 54828 null-terminated.

54829 RATIONALE

54830 Since POSIX.1-200x does not require any association of file times with symbolic links, there is no
 54831 requirement that file times be updated by *readlink*(). The type associated with *bufsiz* is a `size_t`
 54832 in order to be consistent with both the ISO C standard and the definition of *read*(). The behavior
 54833 specified for *readlink*() when *bufsiz* is zero represents historical practice. For this case, the
 54834 standard developers considered a change whereby *readlink*() would return the number of non-
 54835 null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however,
 54836 since the `stat` structure member *st_size* value can be used to determine the size of buffer
 54837 necessary to contain the contents of the symbolic link as returned by *readlink*(), this proposal
 54838 was rejected, and the historical practice retained.

54839 The purpose of the *readlinkat*() function is to read the content of symbolic links in directories
 54840 other than the current working directory without exposure to race conditions. Any part of the
 54841 path of a file could be changed in parallel to a call to *readlink*(), resulting in unspecified behavior.
 54842 By opening a file descriptor for the target directory and using the *readlinkat*() function it can be
 54843 guaranteed that the symbolic link read is located relative to the desired directory.

54844 FUTURE DIRECTIONS

54845 None.

54846 SEE ALSO

54847 [*fstatat*\(\)](#), [*symlink*\(\)](#)

54848 XBD [`<unistd.h>`](#)

54849
54850
54851
54852
54853
54854
54855
54856
54857
54858
54859
54860
54861
54862
54863
54864
54865
54866
54867
54868
54869
54870**CHANGE HISTORY**

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The return type is changed to **ssize_t**, to align with the IEEE P1003.1a draft standard.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- This function is made mandatory.
- In this function it is possible for the return value to exceed the range of the type **ssize_t** (since **size_t** has a larger range of positive values than **ssize_t**). A sentence restricting the size of the **size_t** object is added to the description to resolve this conflict.

The following changes are made for alignment with the ISO POSIX-1: 1996 standard:

- The FUTURE DIRECTIONS section is changed to None.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

The **restrict** keyword is added to the *readlink()* prototype for alignment with the ISO/IEC 9899: 1999 standard.

Issue 7

SD5-XSH-ERN-189 is applied, updating the ERRORS section. +

The *readlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

54871 **NAME**

54872 readv — read a vector

54873 **SYNOPSIS**

```
54874 XSI #include <sys/uio.h>
54875 ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
```

54876 **DESCRIPTION**

54877 The *readv()* function shall be equivalent to *read()*, except as described below. The *readv()*
 54878 function shall place the input data into the *iovcnt* buffers specified by the members of the *iov*
 54879 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than
 54880 or equal to {IOV_MAX}.

54881 Each *iovec* entry specifies the base address and length of an area in memory where data should
 54882 be placed. The *readv()* function shall always fill an area completely before proceeding to the
 54883 next.

54884 Upon successful completion, *readv()* shall mark for update the last data access timestamp of the
 54885 file.

54886 **RETURN VALUE**54887 Refer to *read*.54888 **ERRORS**54889 Refer to *read*.54890 In addition, the *readv()* function shall fail if:54891 [EINVAL] The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.54892 The *readv()* function may fail if:54893 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.54894 **EXAMPLES**54895 **Reading Data into an Array**

54896 The following example reads data from the file associated with the file descriptor *fd* into the
 54897 buffers specified by members of the *iov* array.

```
54898 #include <sys/types.h>
54899 #include <sys/uio.h>
54900 #include <unistd.h>
54901 ...
54902 ssize_t bytes_read;
54903 int fd;
54904 char buf0[20];
54905 char buf1[30];
54906 char buf2[40];
54907 int iovcnt;
54908 struct iovec iov[3];

54909 iov[0].iov_base = buf0;
54910 iov[0].iov_len = sizeof(buf0);
54911 iov[1].iov_base = buf1;
54912 iov[1].iov_len = sizeof(buf1);
```

readv()

```

54913         iov[2].iov_base = buf2;
54914         iov[2].iov_len = sizeof(buf2);
54915         ...
54916         iovcnt = sizeof(iov) / sizeof(struct iovec);
54917         bytes_read = readv(fd, iov, iovcnt);
54918         ...

```

APPLICATION USAGE

None.

RATIONALE

Refer to *read*.

FUTURE DIRECTIONS

None.

SEE ALSO

read, *writev()*

XBD <[sys/uio.h](#)>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 6

Split out from the *read()* reference page.

Issue 7

Changes are made related to support for finegrained timestamps.

54934 **NAME**
 54935 realloc — memory reallocator

54936 **SYNOPSIS**
 54937 #include <stdlib.h>
 54938 void *realloc(void *ptr, size_t size);

54939 DESCRIPTION

54940 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 54941 conflict between the requirements described here and the ISO C standard is unintentional. This
 54942 volume of POSIX.1-200x defers to the ISO C standard.

54943 The *realloc()* function shall change the size of the memory object pointed to by *ptr* to the size
 54944 specified by *size*. The contents of the object shall remain unchanged up to the lesser of the new
 54945 and old sizes. If the new size of the memory object would require movement of the object, the
 54946 space for the previous instantiation of the object is freed. If the new size is larger, the contents of
 54947 the newly allocated portion of the object are unspecified. If *size* is 0 and *ptr* is not a null pointer,
 54948 the object pointed to is freed. If the space cannot be allocated, the object shall remain unchanged.

54949 If *ptr* is a null pointer, *realloc()* shall be equivalent to *malloc()* for the specified size.

54950 If *ptr* does not match a pointer returned earlier by *calloc()*, *malloc()*, or *realloc()* or if the space has
 54951 previously been deallocated by a call to *free()* or *realloc()*, the behavior is undefined.

54952 The order and contiguity of storage allocated by successive calls to *realloc()* is unspecified. The
 54953 pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to
 54954 a pointer to any type of object and then used to access such an object in the space allocated (until
 54955 the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object
 54956 disjoint from any other object. The pointer returned shall point to the start (lowest byte address)
 54957 of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

54958 RETURN VALUE

54959 Upon successful completion with a size not equal to 0, *realloc()* shall return a pointer to the
 54960 (possibly moved) allocated space. If *size* is 0, either a null pointer or a unique pointer that can be
 54961 successfully passed to *free()* shall be returned. If there is not enough available memory, *realloc()*
 54962 shall return a null pointer and set *errno* to [ENOMEM].

54963 ERRORS

54964 The *realloc()* function shall fail if:

54965 CX [ENOMEM] Insufficient memory is available.

54966 EXAMPLES

54967 None.

54968 APPLICATION USAGE

54969 None.

54970 RATIONALE

54971 None.

54972 FUTURE DIRECTIONS

54973 None.

realloc()54974
54975
54976
54977
54978
54979
54980
54981
54982
54983
54984
54985**SEE ALSO***calloc()*, *free()*, *malloc()*

XBD <stdlib.h>

CHANGE HISTORY

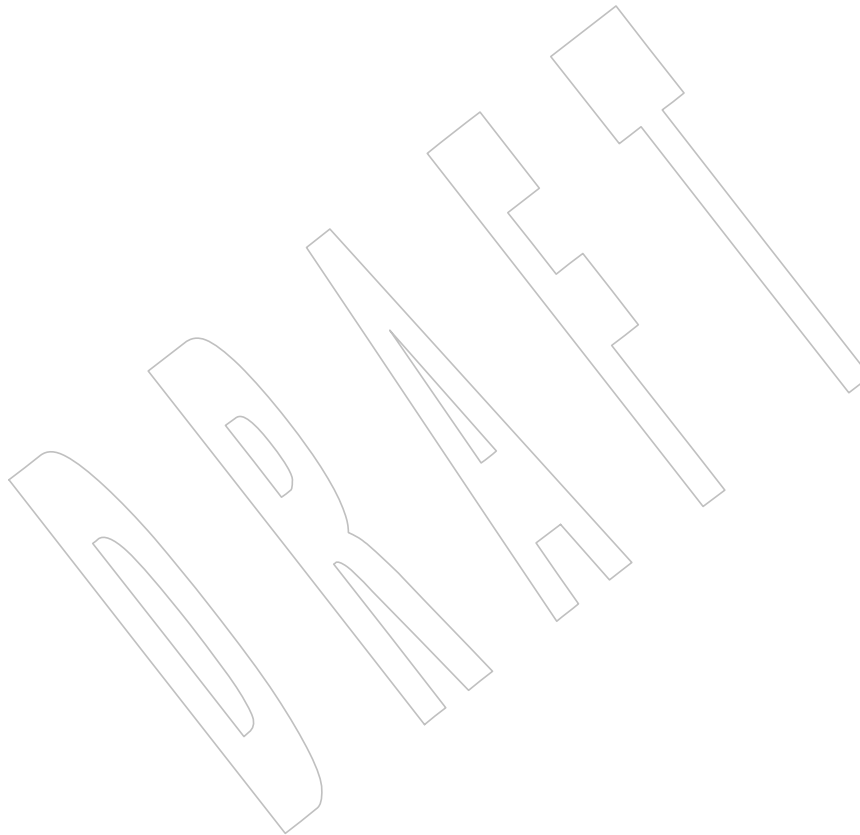
First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE section, if there is not enough available memory, the setting of *errno* to [ENOMEM] is added.
- The [ENOMEM] error condition is added.



54986 **NAME**
 54987 `realpath` — resolve a pathname

54988 **SYNOPSIS**

```
54989 XSI #include <stdlib.h>
54990
54990 char *realpath(const char *restrict file_name,
54991               char *restrict resolved_name);
```

54992 **DESCRIPTION**

54993 The `realpath()` function shall derive, from the pathname pointed to by `file_name`, an absolute
 54994 pathname that names the same file, whose resolution does not involve '.', '..', or symbolic |
 54995 links. If `resolved_name` is a null pointer, the generated pathname shall be stored as a null- |
 54996 terminated string in a buffer allocated by `realpath()`. Otherwise, if {PATH_MAX} is defined as a |
 54997 constant in the <limits.h> header, then the generated pathname shall be stored as a null- |
 54998 terminated string, up to a maximum of {PATH_MAX} bytes, in the buffer pointed to by |
 54999 `resolved_name`.

55000 If `resolved_name` is not a null pointer and {PATH_MAX} is not defined as a constant in the |
 55001 <limits.h> header, the behavior is implementation-defined.

55002 **RETURN VALUE**

55003 Upon successful completion, `realpath()` shall return a pointer to the buffer containing the |
 55004 resolved name. Otherwise, `realpath()` shall return a null pointer and set `errno` to indicate the |
 55005 error.

55006 If the `resolved_name` argument is a null pointer, the pointer returned by `realpath()` can be passed |
 55007 to `free()`.

55008 If the `resolved_name` argument is not a null pointer and the `realpath()` function fails, the contents |
 55009 of the buffer pointed to by `resolved_name` are undefined.

55010 **ERRORS**

55011 The `realpath()` function shall fail if:

55012 [EACCES] Read or search permission was denied for a component of `file_name`.

55013 [EINVAL] The `file_name` argument is a null pointer.

55014 [EIO] An error occurred while reading from the file system.

55015 [ELOOP] A loop exists in symbolic links encountered during resolution of the `file_name` |
 55016 argument.

55017 [ENAMETOOLONG]

55018 The length of the `file_name` argument exceeds {PATH_MAX} or a pathname |
 55019 component is longer than {NAME_MAX}.

55020 [ENOENT] A component of `file_name` does not name an existing file or `file_name` points to |
 55021 an empty string.

55022 [ENOTDIR] A component of the path prefix is not a directory.

55023 The `realpath()` function may fail if:

55024 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during |
 55025 resolution of the `file_name` argument.

- 55026 [ENAMETOOLONG]
 55027 Pathname resolution of a symbolic link produced an intermediate result
 55028 whose length exceeds {PATH_MAX}.
 55029 [ENOMEM] Insufficient storage space is available.

EXAMPLES**Generating an Absolute Pathname**

The following example generates an absolute pathname for the file identified by the *symlinkpath* argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
55034 #include <stdlib.h>
55035 ...
55036 char *symlinkpath = "/tmp/symlink/file";
55037 char *actualpath;

55038 actualpath = realpath(symlinkpath, NULL);
55039 if (actualpath != NULL)
55040 {
55041     ... use actualpath ...
55042     free(actualpath);
55043 }
55044 else
55045 {
55046     ... handle error ...
55047 }
```

APPLICATION USAGE

None.

RATIONALE

Since *realpath()* has no *length* argument, if {PATH_MAX} is not defined as a constant in <limits.h>, applications have no way of determining how large a buffer they need to allocate for it to be safe to pass to *realpath()*. A {PATH_MAX} value obtained from a prior *pathconf()* call is out-of-date by the time *realpath()* is called. Hence the only reliable way to use *realpath()* when {PATH_MAX} is not defined in <limits.h> is to pass a null pointer for *resolved_name* so that *realpath()* will allocate a buffer of the necessary size.

FUTURE DIRECTIONS

None.

SEE ALSO

fpathconf(), *free()*, *getcwd()*, *sysconf()*

XBD <limits.h>, <stdlib.h>

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

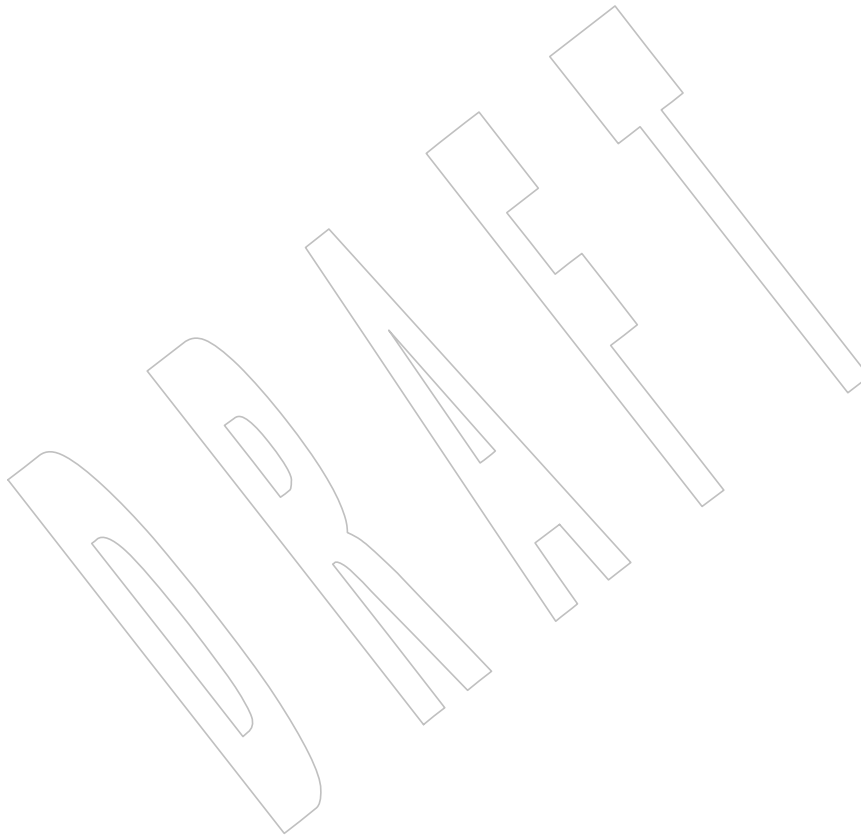
Moved from X/OPEN UNIX extension to BASE.

Issue 6

The **restrict** keyword is added to the *realpath()* prototype for alignment with the ISO/IEC 9899:1999 standard.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

- 55071 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/51 is applied, adding new text to the
55072 DESCRIPTION for the case when *resolved_name* is a null pointer, changing the [EINVAL] error
55073 text, adding text to the RATIONALE, and adding text to FUTURE DIRECTIONS.
- 55074 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/110 is applied, updating the ERRORS
55075 section to refer to the *file_name* argument, rather than a non-existent *path* argument.
- 55076 **Issue 7**
- 55077 This function is updated for passing a null pointer to *realpath()* for the *resolved_name* argument. +



55078 **NAME**

55079 recv — receive a message from a connected socket

55080 **SYNOPSIS**

55081 #include <sys/socket.h>

55082 ssize_t recv(int socket, void *buffer, size_t length, int flags);

55083 **DESCRIPTION**55084 The *recv()* function shall receive a message from a connection-mode or connectionless-mode
55085 socket. It is normally used with connected sockets because it does not permit the application to
55086 retrieve the source address of received data.55087 The *recv()* function takes the following arguments:55088 *socket* Specifies the socket file descriptor.55089 *buffer* Points to a buffer where the message should be stored.55090 *length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.55091 *flags* Specifies the type of message reception. Values of this argument are formed by
55092 logically OR'ing zero or more of the following values:55093 MSG_PEEK Peeks at an incoming message. The data is treated as unread and
55094 the next *recv()* or similar function shall still return this data.55095 MSG_OOB Requests out-of-band data. The significance and semantics of
55096 out-of-band data are protocol-specific.55097 MSG_WAITALL On SOCK_STREAM sockets this requests that the function block
55098 until the full amount of data can be returned. The function may
55099 return the smaller amount of data if the socket is a message-
55100 based socket, if a signal is caught, if the connection is terminated,
55101 if MSG_PEEK was specified, or if an error is pending for the
55102 socket.55103 The *recv()* function shall return the length of the message written to the buffer pointed to by the
55104 *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET,
55105 the entire message shall be read in a single operation. If a message is too long to fit in the
55106 supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be
55107 discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be
55108 ignored. In this case, data shall be returned to the user as soon as it becomes available, and no
55109 data shall be discarded.55110 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
55111 message.55112 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
55113 descriptor, *recv()* shall block until a message arrives. If no messages are available at the socket
55114 and O_NONBLOCK is set on the socket's file descriptor, *recv()* shall fail and set *errno* to
55115 [EAGAIN] or [EWOULDBLOCK].55116 **RETURN VALUE**55117 Upon successful completion, *recv()* shall return the length of the message in bytes. If no
55118 messages are available to be received and the peer has performed an orderly shutdown, *recv()*
55119 shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

55120 ERRORS

- 55121 The *recv()* function shall fail if:
- 55122 [EAGAIN] or [EWOULDBLOCK] The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.
- 55127 [EBADF] The *socket* argument is not a valid file descriptor.
- 55128 [ECONNRESET] A connection was forcibly closed by a peer.
- 55129 [EINTR] The *recv()* function was interrupted by a signal that was caught, before any data was available.
- 55130
- 55131 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.
- 55132 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.
- 55133 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 55134 [EOPNOTSUPP] The specified flags are not supported for this socket type or protocol.
- 55135 [ETIMEDOUT] The connection timed out during connection establishment, or due to a transmission timeout on active connection.
- 55136
- 55137 The *recv()* function may fail if:
- 55138 [EIO] An I/O error occurred while reading from or writing to the file system.
- 55139 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
- 55140 [ENOMEM] Insufficient memory was available to fulfill the request.

55141 EXAMPLES

55142 None.

55143 APPLICATION USAGE

55144 The *recv()* function is equivalent to *recvfrom()* with a zero *address_len* argument, and to *read()* if no flags are used.

55146 The *select()* and *poll()* functions can be used to determine when data is available to be received.

55147 RATIONALE

55148 None.

55149 FUTURE DIRECTIONS

55150 None.

55151 SEE ALSO

55152 *poll()*, *pselect()*, *read*, *recvmsg()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write* -

55153 XBD <sys/socket.h> |

55154 CHANGE HISTORY

55155 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

55156 **NAME**

55157 recvfrom — receive a message from a socket

55158 **SYNOPSIS**

```
55159 #include <sys/socket.h>
55160
55160 ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
55161                 int flags, struct sockaddr *restrict address,
55162                 socklen_t *restrict address_len);
```

55163 **DESCRIPTION**

55164 The *recvfrom()* function shall receive a message from a connection-mode or connectionless-mode
 55165 socket. It is normally used with connectionless-mode sockets because it permits the application
 55166 to retrieve the source address of received data.

55167 The *recvfrom()* function takes the following arguments:

55168	<i>socket</i>	Specifies the socket file descriptor.
55169	<i>buffer</i>	Points to the buffer where the message should be stored.
55170	<i>length</i>	Specifies the length in bytes of the buffer pointed to by the <i>buffer</i> argument.
55171	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by 55172 logically OR'ing zero or more of the following values:
55173	MSG_PEEK	Peeks at an incoming message. The data is treated as unread 55174 and the next <i>recvfrom()</i> or similar function shall still return 55175 this data.
55176	MSG_OOB	Requests out-of-band data. The significance and semantics 55177 of out-of-band data are protocol-specific.
55178	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function 55179 block until the full amount of data can be returned. The 55180 function may return the smaller amount of data if the socket 55181 is a message-based socket, if a signal is caught, if the 55182 connection is terminated, if MSG_PEEK was specified, or if 55183 an error is pending for the socket.
55184	<i>address</i>	A null pointer, or points to a sockaddr structure in which the sending address 55185 is to be stored. The length and format of the address depend on the address 55186 family of the socket.
55187	<i>address_len</i>	Specifies the length of the sockaddr structure pointed to by the <i>address</i> 55188 argument.

55189 The *recvfrom()* function shall return the length of the message written to the buffer pointed to by
 55190 RS the *buffer* argument. For message-based sockets, such as **SOCK_RAW**, **SOCK_DGRAM**, and
 55191 **SOCK_SEQPACKET**, the entire message shall be read in a single operation. If a message is too
 55192 long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess
 55193 bytes shall be discarded. For stream-based sockets, such as **SOCK_STREAM**, message
 55194 boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes
 55195 available, and no data shall be discarded.

55196 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 55197 message.

55198 Not all protocols provide the source address for messages. If the *address* argument is not a null
 55199 pointer and the protocol provides the source address of messages, the source address of the

55200 received message shall be stored in the **sockaddr** structure pointed to by the *address* argument,
 55201 and the length of this address shall be stored in the object pointed to by the *address_len*
 55202 argument.

55203 If the actual length of the address is greater than the length of the supplied **sockaddr** structure,
 55204 the stored address shall be truncated.

55205 If the *address* argument is not a null pointer and the protocol does not provide the source address
 55206 of messages, the value stored in the object pointed to by *address* is unspecified.

55207 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 55208 descriptor, *recvfrom()* shall block until a message arrives. If no messages are available at the
 55209 socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom()* shall fail and set *errno*
 55210 to [EAGAIN] or [EWOULDBLOCK].

55211 RETURN VALUE

55212 Upon successful completion, *recvfrom()* shall return the length of the message in bytes. If no
 55213 messages are available to be received and the peer has performed an orderly shutdown,
 55214 *recvfrom()* shall return 0. Otherwise, the function shall return -1 and set *errno* to indicate the
 55215 error.

55216 ERRORS

55217 The *recvfrom()* function shall fail if:

55218 [EAGAIN] or [EWOULDBLOCK]

55219 The socket's file descriptor is marked O_NONBLOCK and no data is waiting
 55220 to be received; or MSG_OOB is set and no out-of-band data is available and
 55221 either the socket's file descriptor is marked O_NONBLOCK or the socket does
 55222 not support blocking to await out-of-band data.

55223 [EBADF] The *socket* argument is not a valid file descriptor.

55224 [ECONNRESET] A connection was forcibly closed by a peer.

55225 [EINTR] A signal interrupted *recvfrom()* before any data was available.

55226 [EINVAL] The MSG_OOB flag is set and no out-of-band data is available.

55227 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

55228 [ENOTSOCK] The *socket* argument does not refer to a socket.

55229 [EOPNOTSUPP] The specified flags are not supported for this socket type.

55230 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 55231 transmission timeout on active connection.

55232 The *recvfrom()* function may fail if:

55233 [EIO] An I/O error occurred while reading from or writing to the file system.

55234 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

55235 [ENOMEM] Insufficient memory was available to fulfill the request.

recvfrom()

55236

EXAMPLES

55237

None.

55238

APPLICATION USAGE

55239

The *select()* and *poll()* functions can be used to determine when data is available to be received.

55240

RATIONALE

55241

None.

55242

FUTURE DIRECTIONS

55243

None.

55244

SEE ALSO

55245

poll(), *pselect()*, *read*, *recv()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*, *write* -

55246

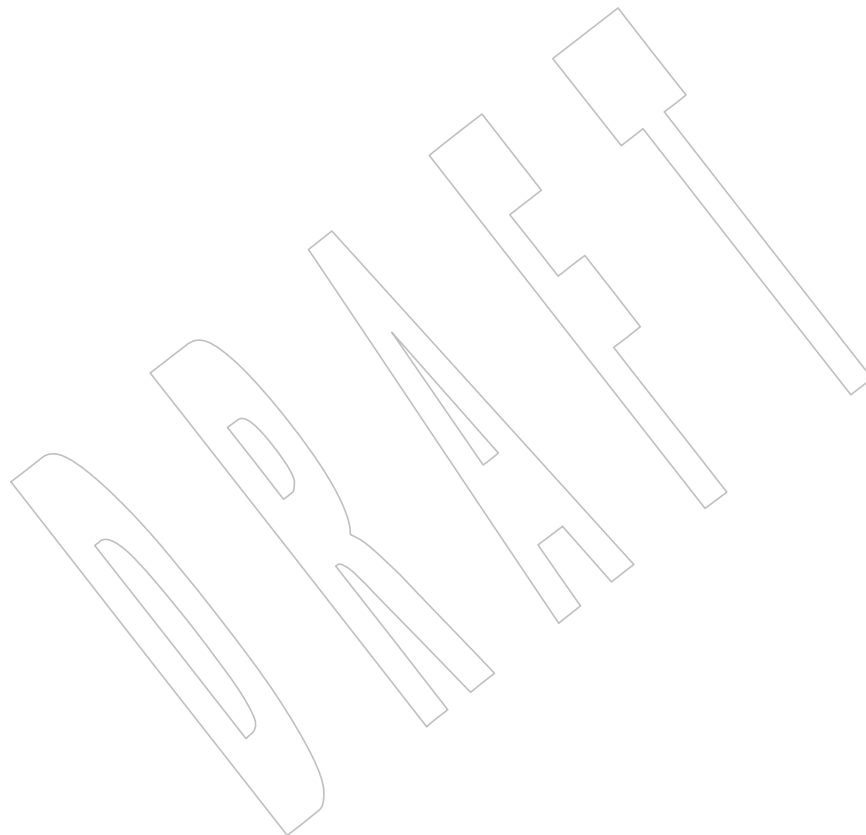
XBD [<sys/socket.h>](#) |

55247

CHANGE HISTORY

55248

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



55249 **NAME**55250 `recvmsg` — receive a message from a socket55251 **SYNOPSIS**55252 `#include <sys/socket.h>`55253 `ssize_t recvmsg(int socket, struct msghdr *message, int flags);`55254 **DESCRIPTION**

55255 The `recvmsg()` function shall receive a message from a connection-mode or connectionless-mode
 55256 socket. It is normally used with connectionless-mode sockets because it permits the application
 55257 to retrieve the source address of received data.

55258 The `recvmsg()` function takes the following arguments:

55259	<i>socket</i>	Specifies the socket file descriptor.
55260	<i>message</i>	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored on input, but may contain meaningful values on output.
55261		
55262		
55263		
55264	<i>flags</i>	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:
55265		
55266	MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
55267		
55268	MSG_PEEK	Peeks at the incoming message.
55269	MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.
55270		
55271		
55272		
55273		
55274		

55275 The `recvmsg()` function shall receive messages from unconnected or connected sockets and shall
 55276 return the length of the message.

55277 The `recvmsg()` function shall return the total length of the message. For message-based sockets,
 55278 such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single
 55279 operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the
 55280 *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the
 55281 *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM,
 55282 message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it
 55283 becomes available, and no data shall be discarded.

55284 If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first
 55285 message.

55286 If no messages are available at the socket and O_NONBLOCK is not set on the socket's file
 55287 descriptor, `recvmsg()` shall block until a message arrives. If no messages are available at the
 55288 socket and O_NONBLOCK is set on the socket's file descriptor, the `recvmsg()` function shall fail
 55289 and set *errno* to [EAGAIN] or [EWOULDBLOCK].

55290 In the **msghdr** structure, the *msg_name* and *msg_namelen* members specify the source address if
 55291 the socket is unconnected. If the socket is connected, the *msg_name* and *msg_namelen* members
 55292 shall be ignored. The *msg_name* member may be a null pointer if no names are desired or

55293 required. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be
 55294 stored. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of
 55295 this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field
 55296 gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn
 55297 until all of the received data is stored or all of the areas have been filled.

55298 Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-
 55299 inclusive OR of all of the following flags that indicate conditions detected for the received
 55300 message:

55301 **MSG_EOR** End-of-record was received (if supported by the protocol).
 55302 **MSG_OOB** Out-of-band data was received.
 55303 **MSG_TRUNC** Normal data was truncated.
 55304 **MSG_CTRUNC** Control data was truncated.

55305 RETURN VALUE

55306 Upon successful completion, *recvmsg()* shall return the length of the message in bytes. If no
 55307 messages are available to be received and the peer has performed an orderly shutdown,
 55308 *recvmsg()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

55309 ERRORS

55310 The *recvmsg()* function shall fail if:

55311 [EAGAIN] or [EWOULDBLOCK]
 55312 The socket's file descriptor is marked **O_NONBLOCK** and no data is waiting
 55313 to be received; or **MSG_OOB** is set and no out-of-band data is available and
 55314 either the socket's file descriptor is marked **O_NONBLOCK** or the socket does
 55315 not support blocking to await out-of-band data.

55316 [EBADF] The *socket* argument is not a valid open file descriptor.

55317 [ECONNRESET] A connection was forcibly closed by a peer.

55318 [EINTR] This function was interrupted by a signal before any data was available.

55319 [EINVAL] The sum of the *iov_len* values overflows a **ssize_t**, or the **MSG_OOB** flag is set
 55320 and no out-of-band data is available.

55321 [EMSGSIZE] The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less
 55322 than or equal to 0, or is greater than **{IOV_MAX}**.

55323 [ENOTCONN] A receive is attempted on a connection-mode socket that is not connected.

55324 [ENOTSOCK] The *socket* argument does not refer to a socket.

55325 [EOPNOTSUPP] The specified flags are not supported for this socket type.

55326 [ETIMEDOUT] The connection timed out during connection establishment, or due to a
 55327 transmission timeout on active connection.

55328 The *recvmsg()* function may fail if:

55329 [EIO] An I/O error occurred while reading from or writing to the file system.
 55330 [ENOBUFS] Insufficient resources were available in the system to perform the operation.
 55331 [ENOMEM] Insufficient memory was available to fulfill the request.

55332

EXAMPLES

55333

None.

55334

APPLICATION USAGE

55335

The *select()* and *poll()* functions can be used to determine when data is available to be received.

55336

RATIONALE

55337

None.

55338

FUTURE DIRECTIONS

55339

None.

55340

SEE ALSO

55341

poll(), *pselect()*, *recv()*, *recvfrom()*, *send()*, *sendmsg()*, *sendto()*, *shutdown()*, *socket()*

55342

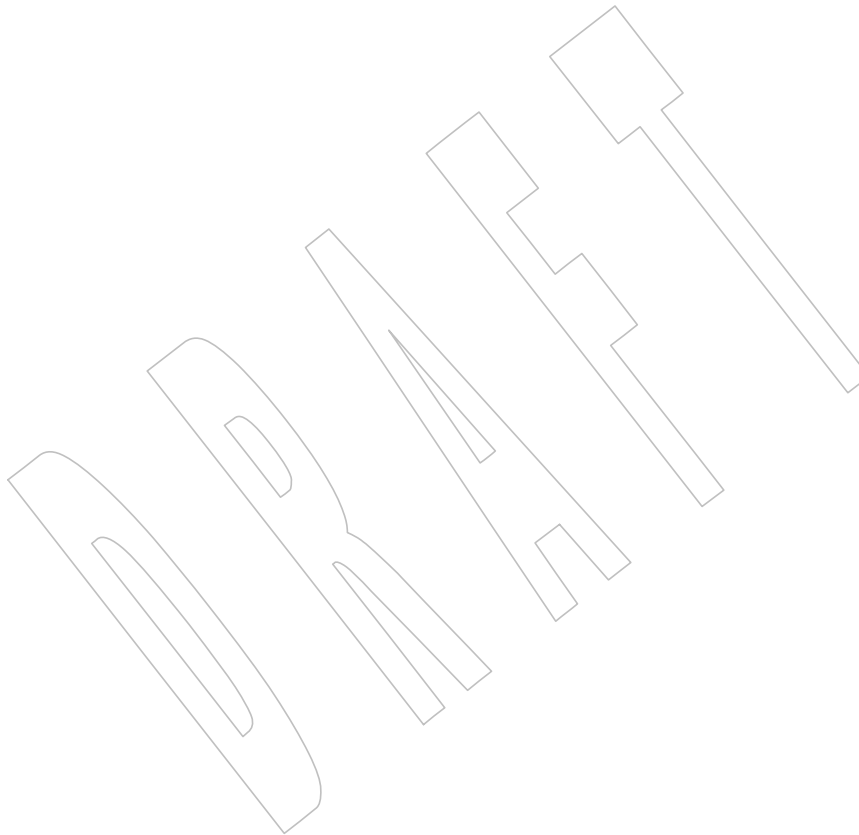
XBD <[sys/socket.h](#)>

55343

CHANGE HISTORY

55344

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



55345 **NAME**

55346 regcomp, regerror, regex, regfree — regular expression matching

55347 **SYNOPSIS**

```
55348 #include <regex.h>
55349
55349 int regcomp(regex_t *restrict preg, const char *restrict pattern,
55350             int cflags);
55351 size_t regerror(int errcode, const regex_t *restrict preg,
55352               char *restrict errbuf, size_t errbuf_size);
55353 int regex(const regex_t *restrict preg, const char *restrict string,
55354           size_t nmatch, regmatch_t pmatch[restrict], int eflags);
55355 void regfree(regex_t *preg);
```

55356 **DESCRIPTION**55357 These functions interpret *basic* and *extended* regular expressions as described in XBD [Chapter 9](#) |
55358 (on page 167).55359 The `regex_t` structure is defined in `<regex.h>` and contains at least the following member:

Member Type	Member Name	Description
size_t	re_nsub	Number of parenthesized subexpressions.

55362 The `regmatch_t` structure is defined in `<regex.h>` and contains at least the following members:

Member Type	Member Name	Description
<code>regoff_t</code>	<code>rm_so</code>	Byte offset from start of <i>string</i> to start of substring.
<code>regoff_t</code>	<code>rm_eo</code>	Byte offset from start of <i>string</i> of the first character after the end of substring.

55367 The `regcomp()` function shall compile the regular expression contained in the string pointed to by
55368 the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags*
55369 argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in
55370 the `<regex.h>` header:

55371 REG_EXTENDED Use Extended Regular Expressions.
55372 REG_ICASE Ignore case in match (see XBD [Chapter 9](#), on page 167). |
55373 REG_NOSUB Report only success/fail in `regex()`.
55374 REG_NEWLINE Change the handling of <newline>s, as described in the text.

55375 The default regular expression type for *pattern* is a Basic Regular Expression. The application can
55376 specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

55377 If the REG_NOSUB flag was not set in *cflags*, then `regcomp()` shall set `re_nsub` to the number of
55378 parenthesized subexpressions (delimited by "`\\(\\)`" in basic regular expressions or "`()`" in
55379 extended regular expressions) found in *pattern*.

55380 The `regex()` function compares the null-terminated string specified by *string* with the compiled
55381 regular expression *preg* initialized by a previous call to `regcomp()`. If it finds a match, `regex()`
55382 shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The
55383 *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are
55384 defined in the `<regex.h>` header:

55385 REG_NOTBOL The first character of the string pointed to by *string* is not the beginning of the
 55386 line. Therefore, the circumflex character ('^'), when taken as a special
 55387 character, shall not match the beginning of *string*.

55388 REG_NOTEOL The last character of the string pointed to by *string* is not the end of the line.
 55389 Therefore, the dollar sign ('\$'), when taken as a special character, shall not
 55390 match the end of *string*.

55391 If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexec()* shall
 55392 ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument
 55393 points to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that
 55394 array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions
 55395 of *pattern*: *pmatch[i].rm_so* shall be the byte offset of the beginning and *pmatch[i].rm_eo* shall be
 55396 one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th
 55397 matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* identify the substring that
 55398 corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]*
 55399 shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself
 55400 counts as a subexpression), then *regexec()* shall still do the match, but shall record only the first
 55401 *nmatch* substrings.

55402 When matching a basic or extended regular expression, any given parenthesized subexpression
 55403 of *pattern* might participate in the match of several different substrings of *string*, or it might not
 55404 match any substring even though the pattern as a whole did match. The following rules shall be
 55405 used to determine which substrings to report in *pmatch* when matching regular expressions:

55406 1. If subexpression *i* in a regular expression is not contained within another subexpression,
 55407 and it participated in the match several times, then the byte offsets in *pmatch[i]* shall
 55408 delimit the last such match.

55409 2. If subexpression *i* is not contained within another subexpression, and it did not
 55410 participate in an otherwise successful match, the byte offsets in *pmatch[i]* shall be -1. A
 55411 subexpression does not participate in the match when:

55412 ' * ' or " \{ \} " appears immediately after the subexpression in a basic regular
 55413 expression, or ' * ', ' ? ', or " { } " appears immediately after the subexpression in
 55414 an extended regular expression, and the subexpression did not match (matched 0
 55415 times)

55416 or:

55417 ' | ' is used in an extended regular expression to select this subexpression or
 55418 another, and the other subexpression matched.

55419 3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained
 55420 within any other subexpression that is contained within *j*, and a match of subexpression *j*
 55421 is reported in *pmatch[j]*, then the match or non-match of subexpression *i* reported in
 55422 *pmatch[i]* shall be as described in 1. and 2. above, but within the substring reported in
 55423 *pmatch[j]* rather than the whole string. The offsets in *pmatch[i]* are still relative to the start
 55424 of *string*.

55425 4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1,
 55426 then the pointers in *pmatch[i]* shall also be -1.

55427 5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be
 55428 the byte offset of the character or null terminator immediately following the zero-length
 55429 string.

55430 If, when *regexec()* is called, the locale is different from when the regular expression was
 55431 compiled, the result is undefined.

55432 If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an
 55433 ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary
 55434 character except as follows:

- 55435 1. A <newline> in *string* shall not be matched by a period outside a bracket expression or by
 55436 any form of a non-matching list (see XBD Chapter 9, on page 167).
- 55437 2. A circumflex ('^') in *pattern*, when used to specify expression anchoring (see XBD
 55438 Section 9.3.8, on page 173), shall match the zero-length string immediately after a
 55439 <newline> in *string*, regardless of the setting of REG_NOTBOL.
- 55440 3. A dollar sign ('\$') in *pattern*, when used to specify expression anchoring, shall match the
 55441 zero-length string immediately before a <newline> in *string*, regardless of the setting of
 55442 REG_NOTEOL.

55443 The *regfree()* function frees any memory allocated by *regcomp()* associated with *preg*.

55444 The following constants are defined as error return values:

55445	REG_NOMATCH	<i>regexec()</i> failed to match.
55446	REG_BADPAT	Invalid regular expression.
55447	REG_ECOLLATE	Invalid collating element referenced.
55448	REG_ECTYPE	Invalid character class type referenced.
55449	REG_EESCAPE	Trailing '\\' in pattern.
55450	REG_ESUBREG	Number in "\digit" invalid or in error.
55451	REG_EBRACK	"[]" imbalance.
55452	REG_EPAREN	"\(\)" or "()" imbalance.
55453	REG_EBRACE	"\{\}" imbalance.
55454	REG_BADBR	Content of "\{\}" invalid: not a number, number too large, more than 55455 two numbers, first larger than second.
55456	REG_ERANGE	Invalid endpoint in range expression.
55457	REG_ESPACE	Out of memory.
55458	REG_BADRPT	'?', '*', or '+' not preceded by valid regular expression.

55459 The *regerror()* function provides a mapping from error codes returned by *regcomp()* and
 55460 *regexec()* to unspecified printable strings. It generates a string corresponding to the value of the
 55461 *errcode* argument, which the application shall ensure is the last non-zero value returned by
 55462 *regcomp()* or *regexec()* with the given value of *preg*. If *errcode* is not such a value, the content of
 55463 the generated string is unspecified.

55464 If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec()* or *regcomp()*,
 55465 the *regerror()* still generates an error string corresponding to the value of *errcode*, but it might not
 55466 be as detailed under some implementations.

55467 If the *errbuf_size* argument is not 0, *regerror()* shall place the generated string into the buffer of
 55468 size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit
 55469 in the buffer, *regerror()* shall truncate the string and null-terminate the result.

55470 If *errbuf_size* is 0, *regerror()* shall ignore the *errbuf* argument, and return the size of the buffer
 55471 needed to hold the generated string.

55472 If the *preg* argument to *regexec()* or *regfree()* is not a compiled regular expression returned by
 55473 *regcomp()*, the result is undefined. A *preg* is no longer treated as a compiled regular expression
 55474 after it is given to *regfree()*.

55475 **RETURN VALUE**

55476 Upon successful completion, the *regcomp()* function shall return 0. Otherwise, it shall return an
 55477 integer value indicating an error as described in <regex.h>, and the content of *preg* is undefined.
 55478 If a code is returned, the interpretation shall be as given in <regex.h>.

55479 If *regcomp()* detects an invalid RE, it may return REG_BADPAT, or it may return one of the error
 55480 codes that more precisely describes the error.

55481 Upon successful completion, the *regexec()* function shall return 0. Otherwise, it shall return
 55482 REG_NOMATCH to indicate no match.

55483 Upon successful completion, the *regerror()* function shall return the number of bytes needed to
 55484 hold the entire generated string, including the null termination. If the return value is greater
 55485 than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

55486 The *regfree()* function shall not return a value.

55487 **ERRORS**

55488 No errors are defined.

55489 **EXAMPLES**

```

55490 #include <regex.h>
55491 /*
55492  * Match string against the extended regular expression in
55493  * pattern, treating errors as no match.
55494  *
55495  * Return 1 for match, 0 for no match.
55496  */
55497 int
55498 match(const char *string, char *pattern)
55499 {
55500     int     status;
55501     regex_t re;
55502     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
55503         return(0); /* Report error. */
55504     }
55505     status = regexec(&re, string, (size_t) 0, NULL, 0);
55506     regfree(&re);
55507     if (status != 0) {
55508         return(0); /* Report error. */
55509     }
55510     return(1);
55511 }

```

55512 The following demonstrates how the REG_NOTBOL flag could be used with *regexec()* to find all
 55513 substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very
 55514 little error checking is done.)

```

55515 (void) regcomp (&re, pattern, 0);
55516 /* This call to regexec() finds the first match on the line. */
55517 error = regexec (&re, &buffer[0], 1, &pm, 0);
55518 while (error == 0) { /* While matches found. */
55519     /* Substring found between pm.rm_so and pm.rm_eo. */
55520     /* This call to regexec() finds the next match. */
55521     error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
55522 }

```

APPLICATION USAGE

An application could use:

```
regerror(code, preg, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc()* a buffer to hold the string, and then call *regerror()* again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc()* to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in XCU Section 2.13 (on page 2278), use the *fnmatch()* function.

RATIONALE

The *regexexec()* function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application developer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg->re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexexec()* as a single long string and specify REG_NEWLINE to *regcomp()* to get the desired behavior. The application must ensure that there are no explicit <newline>s in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexexec()*, but it is in the *cflags* parameter to *regcomp()* to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp()* regardless of the setting of REG_NEWLINE and have *regexexec()* handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep -i* option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch[]* are defined using offsets from the start of the string rather than pointers. This allows type-safe access to both constant and non-constant strings.

The type **regoff_t** is used for the elements of *pmatch[]* to ensure that the application can represent large arrays in memory (important for an application conforming to the Shell and Utilities volume of POSIX.1-200x).

The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate future extensions in which the string to be searched is taken from a file. However, these future extensions have not appeared. The requirement rules out popular implementations with 32-bit **regoff_t** and 64-bit **off_t**, so it has been removed.

The standard developers rejected the inclusion of a *regsub()* function that would be used to do substitutions for a matched RE. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both RE parsing and substitution are possible to implement without support other than that required by the ISO C standard, but matching is much more complex than substituting. The only difficult part of substitution, given the information supplied by *regexexec()*, is finding the next character in a string when there can be multi-byte characters. That is a much larger issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid filling the *errno* name space for this feature.

The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate

55571 **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used
 55572 simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple
 55573 threads of lightweight processes. (The *preg* argument to *regexec()* is declared with type **const**, so
 55574 the implementation is not permitted to use the structure to store intermediate results.) It also
 55575 allows an application to request an arbitrary number of substrings from an RE. The number of
 55576 subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexec()*,
 55577 consideration was given to dropping the REG_NOSUB flag since the user can now specify this
 55578 with a zero *nmatch* argument to *regexec()*. However, keeping REG_NOSUB allows an
 55579 implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()* that
 55580 no subexpressions need be reported. The implementation is only required to fill in *pmatch* if
 55581 *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in
 55582 the ISO C standard, is unsigned, so the description of *regexec()* does not need to address
 55583 negative values of *nmatch*.

55584 REG_NOTBOL was added to allow an application to do repeated searches for the same pattern
 55585 in a line. If the pattern contains a circumflex character that should match the beginning of a line,
 55586 then the pattern should only match when matched against the beginning of the line. Without
 55587 the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches,
 55588 but in the general case this would require parsing the expression. The need for REG_NOTEOL is
 55589 not as clear; it was added for symmetry.

55590 The addition of the *regerror()* function addresses the historical need for conforming application
 55591 programs to have access to error information more than “Function failed to compile/match your
 55592 RE for unknown reasons”.

55593 This interface provides for two different methods of dealing with error conditions. The specific
 55594 error codes (REG_EBRACE, for example), defined in **<regex.h>**, allow an application to recover
 55595 from an error if it is so able. Many applications, especially those that use patterns supplied by a
 55596 user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-
 55597 readable error message to present to the user.

55598 The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating
 55599 memory to hold the generated string. The scheme used by *strerror()* in the ISO C standard was
 55600 considered unacceptable since it creates difficulties for multi-threaded applications.

55601 The *preg* argument is provided to *regerror()* to allow an implementation to generate a more
 55602 descriptive message than would be possible with *errcode* alone. An implementation might, for
 55603 example, save the character offset of the offending character of the pattern in a field of *preg*, and
 55604 then include that in the generated message string. The implementation may also ignore *preg*.

55605 A REG_FILENAME flag was considered, but omitted. This flag caused *regexec()* to match |
 55606 patterns as described in XCU [Section 2.13](#) (on page 2278) instead of REs. This service is now |
 55607 provided by the *fnmatch()* function.

55608 Notice that there is a difference in philosophy between the ISO POSIX-2:1993 standard and
 55609 POSIX.1-200x in how to handle a “bad” regular expression. The ISO POSIX-2:1993 standard says
 55610 that many bad constructs “produce undefined results”, or that “the interpretation is undefined”.
 55611 POSIX.1-200x, however, says that the interpretation of such REs is unspecified. The term
 55612 “undefined” means that the action by the application is an error, of similar severity to passing a
 55613 bad pointer to a function.

55614 The *regcomp()* and *regexec()* functions are required to accept any null-terminated string as the
 55615 *pattern* argument. If the meaning of the string is “undefined”, the behavior of the function is
 55616 “unspecified”. POSIX.1-200x does not specify how the functions will interpret the pattern; they
 55617 might return error codes, or they might do pattern matching in some completely unexpected
 55618 way, but they should not do something like abort the process.

55619
55620
55621
55622
55623
55624
55625
55626
55627
55628
55629
55630
55631
55632
55633
55634
55635
55636
55637
55638
55639
55640
55641**FUTURE DIRECTIONS**

None.

SEE ALSO*fnmatch()*, *glob()*XBD Chapter 9 (on page 167), `<regex.h>`, `<sys/types.h>`

XCU Section 2.13 (on page 2278)

CHANGE HISTORY

First released in Issue 4. Derived from the ISO POSIX-2 standard.

Issue 5

Moved from POSIX2 C-language Binding to BASE.

Issue 6In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The normative text is updated to avoid use of the term “must” for application requirements.

The REG_ENOSYS constant is removed.

The **restrict** keyword is added to the *regcomp()*, *regerror()*, and *regexec()* prototypes for alignment with the ISO/IEC 9899:1999 standard.**Issue 7**

SD5-XBD-ERN-60 is applied.

55642 **NAME**
 55643 remainder, remainderf, remainderl — remainder function

55644 **SYNOPSIS**
 55645 #include <math.h>
 55646 double remainder(double x, double y);
 55647 float remainderf(float x, float y);
 55648 long double remainderl(long double x, long double y);

55649 **DESCRIPTION**

55650 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55651 conflict between the requirements described here and the ISO C standard is unintentional. This
 55652 volume of POSIX.1-200x defers to the ISO C standard.

55653 These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value
 55654 n is the integral value nearest the exact value x/y . When $|n-x/y|=\frac{1}{2}$, the value n is chosen to
 55655 be even.

55656 The behavior of *remainder()* shall be independent of the rounding mode.

55657 **RETURN VALUE**

55658 Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$
 55659 when y is non-zero.

55660 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
 55661 implementation-defined whether a domain error occurs or zero is returned.

55662 MX If x or y is NaN, a NaN shall be returned.

55663 If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and either a NaN (if
 55664 supported), or an implementation-defined value shall be returned.

55665 **ERRORS**

55666 These functions shall fail if:

55667 MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-
 55668 NaN.

55669 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 55670 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 55671 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 55672 shall be raised.

55673 These functions may fail if:

55674 **Domain Error** The y argument is zero.

55675 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 55676 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 55677 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 55678 shall be raised.

remainder()

55679

EXAMPLES

55680

None.

55681

APPLICATION USAGE

55682

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

55683

55684

RATIONALE

55685

None.

55686

FUTURE DIRECTIONS

55687

None.

55688

SEE ALSO

55689

abs(), *div()*, *feclearexcept()*, *fetestexcept()*, *ldiv()*

55690

XBD Section 4.19 (on page 104), **<math.h>**

55691

CHANGE HISTORY

55692

First released in Issue 4, Version 2.

55693

Issue 5

55694

Moved from X/OPEN UNIX extension to BASE.

55695

Issue 6

55696

The *remainder()* function is no longer marked as an extension.

55697

The *remainderf()* and *remainderl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

55698

55699

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

55700

55701

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

55702

55703

Issue 7

55704

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #55 (SD5-XSH-ERN-82) is applied.

55705 **NAME**

55706 remove — remove a file

55707 **SYNOPSIS**

55708 #include <stdio.h>

55709 int remove(const char *path);

55710 **DESCRIPTION**

55711 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55712 conflict between the requirements described here and the ISO C standard is unintentional. This
 55713 volume of POSIX.1-200x defers to the ISO C standard.

55714 The *remove()* function shall cause the file named by the pathname pointed to by *path* to be no
 55715 longer accessible by that name. A subsequent attempt to open that file using that name shall fail,
 55716 unless it is created anew.

55717 CX If *path* does not name a directory, *remove(path)* shall be equivalent to *unlink(path)*.

55718 If *path* names a directory, *remove(path)* shall be equivalent to *rmdir(path)*.

55719 **RETURN VALUE**55720 CX Refer to *rmdir* or *unlink*.55721 **ERRORS**55722 CX Refer to *rmdir* or *unlink*.55723 **EXAMPLES**55724 **Removing Access to a File**55725 The following example shows how to remove access to a file named */home/cnd/old_mods*.

```
55726 #include <stdio.h>
55727 int status;
55728 ...
55729 status = remove("/home/cnd/old_mods");
```

55730 **APPLICATION USAGE**

55731 None.

55732 **RATIONALE**

55733 None.

55734 **FUTURE DIRECTIONS**

55735 None.

55736 **SEE ALSO**55737 *rmdir*, *unlink*

55738 XBD <stdio.h>

55739 **CHANGE HISTORY**

55740 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard and the ISO C
 55741 standard.

55742

Issue 6

55743

Extensions beyond the ISO C standard are marked.

55744

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

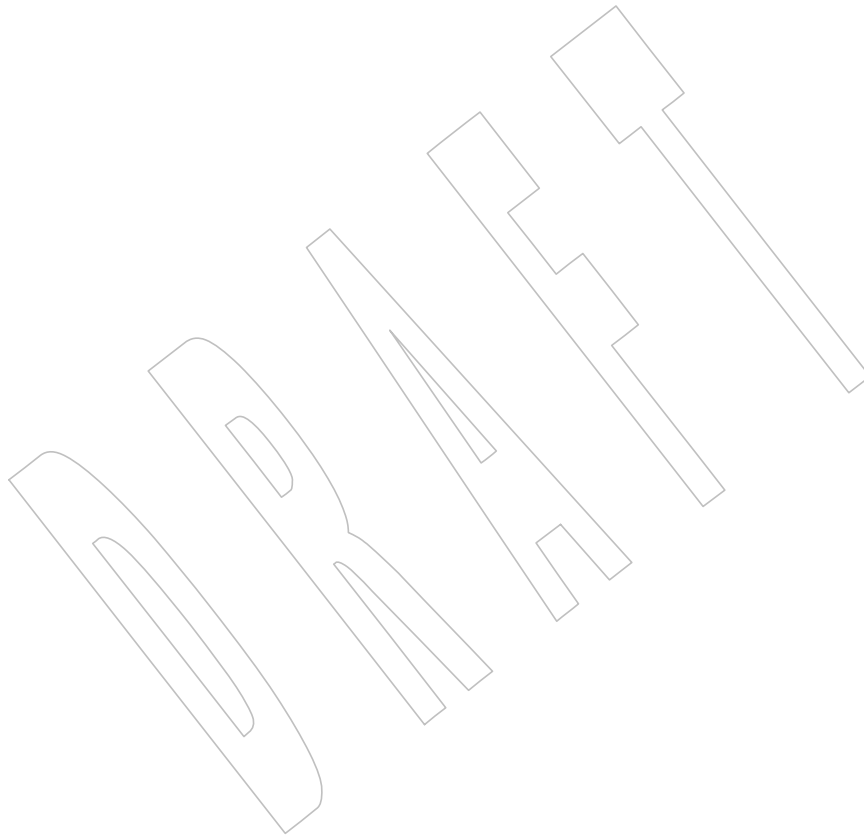
55745

55746

- The DESCRIPTION, RETURN VALUE, and ERRORS sections are updated so that if *path* is not a directory, *remove()* is equivalent to *unlink()*, and if it is a directory, it is equivalent to *rmdir()*.

55747

55748



55749 **NAME**
55750 `remque` — remove an element from a queue

55751 **SYNOPSIS**

55752 XSI `#include <search.h>`
55753 `void remque(void *element);`

55754 **DESCRIPTION**

55755 Refer to *insque()*.

55756 **NAME**
 55757 remquo, remquof, remquol — remainder functions

55758 **SYNOPSIS**
 55759 #include <math.h>
 55760 double remquo(double x, double y, int *quo);
 55761 float remquof(float x, float y, int *quo);
 55762 long double remquol(long double x, long double y, int *quo);

55763 **DESCRIPTION**
 55764 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 55765 conflict between the requirements described here and the ISO C standard is unintentional. This
 55766 volume of POSIX.1-200x defers to the ISO C standard.

55767 The *remquo()*, *remquof()*, and *remquol()* functions shall compute the same remainder as the
 55768 *remainder()*, *remainderf()*, and *remainderl()* functions, respectively. In the object pointed to by *quo*,
 55769 they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n
 55770 to the magnitude of the integral quotient of x/y , where n is an implementation-defined integer
 55771 greater than or equal to 3. If y is zero, the value stored in the object pointed to by *quo* is
 55772 unspecified.

55773 An application wishing to check for error situations should set *errno* to zero and call
 55774 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 55775 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 55776 zero, an error has occurred.

55777 **RETURN VALUE**
 55778 These functions shall return $x \text{ REM } y$.
 55779 On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is
 55780 implementation-defined whether a domain error occurs or zero is returned.

55781 MX If x or y is NaN, a NaN shall be returned.
 55782 If x is $\pm\text{Inf}$ or y is zero and the other argument is non-NaN, a domain error shall occur, and either
 55783 a NaN (if supported), or an implementation-defined value shall be returned.

55784 **ERRORS**
 55785 These functions shall fail if:
 55786 MX **Domain Error** The x argument is $\pm\text{Inf}$, or the y argument is ± 0 and the other argument is non-
 55787 NaN.
 55788 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 55789 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 55790 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 55791 shall be raised.

55792 These functions may fail if:
 55793 **Domain Error** The y argument is zero.
 55794 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 55795 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 55796 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 55797 shall be raised.

55798

EXAMPLES

55799

None.

55800

APPLICATION USAGE

55801

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

55802

55803

RATIONALE

55804

These functions are intended for implementing argument reductions which can exploit a few low-order bits of the quotient. Note that x may be so large in magnitude relative to y that an exact representation of the quotient is not practical.

55805

55806

55807

FUTURE DIRECTIONS

55808

None.

55809

SEE ALSO

55810

feclearexcept(), *fetestexcept()*, *remainder()*

55811

XBD Section 4.19 (on page 104), `<math.h>`

55812

CHANGE HISTORY

55813

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

55814

Issue 7

55815

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #56 (SD5-XSH-ERN-83) is applied.

DRAFT

55816 **NAME**

55817 rename, renameat — rename file relative to directory file descriptor

55818 **SYNOPSIS**

55819 #include <stdio.h>

55820 int rename(const char *old, const char *new);

55821 CX int renameat(int oldfd, const char *old, int newfd,
55822 const char *new);55823 **DESCRIPTION**55824 CX For *rename()*: The functionality described on this reference page is aligned with the ISO C
55825 standard. Any conflict between the requirements described here and the ISO C standard is
55826 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.55827 The *rename()* function shall change the name of a file. The *old* argument points to the pathname
55828 of the file to be renamed. The *new* argument points to the new pathname of the file.55829 CX If either the *old* or *new* argument names a symbolic link, *rename()* shall operate on the symbolic
55830 link itself, and shall not resolve the last component of the argument. If the *old* argument and the
55831 *new* argument resolve to the same existing file, *rename()* shall return successfully and perform no
55832 other action.55833 If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall
55834 not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be
55835 removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other
55836 processes throughout the renaming operation and refer either to the file referred to by *new* or *old*
55837 before the operation began. Write access permission is required for both the directory containing
55838 *old* and the directory containing *new*.55839 If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the
55840 pathname of a file that is not a directory. If the directory named by the *new* argument exists, it
55841 shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout
55842 the renaming operation and shall refer either to the directory referred to by *new* or *old* before the
55843 operation began. If *new* names an existing directory, it shall be required to be an empty directory.55844 If either *pathname* argument refers to a path whose final component is either dot or dot-dot,
55845 *rename()* shall fail.55846 If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed.
55847 If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.55848 The *new* pathname shall not contain a path prefix that names *old*. Write access permission is
55849 required for the directory containing *old* and the directory containing *new*. If the *old* argument
55850 points to the pathname of a directory, write access permission may be required for the directory
55851 named by *old*, and, if it exists, the directory named by *new*.55852 If the link named by the *new* argument exists and the file's link count becomes 0 when it is
55853 removed and no process has the file open, the space occupied by the file shall be freed and the
55854 file shall no longer be accessible. If one or more processes have the file open when the last link is
55855 removed, the link shall be removed before *rename()* returns, but the removal of the file contents
55856 shall be postponed until all references to the file are closed.55857 Upon successful completion, *rename()* shall mark for update the last data modification and last
55858 file status change timestamps of the parent directory of each file.55859 If the *rename()* function fails for any reason other than [EIO], any file named by *new* shall be

55860

unaffected.

55861

55862

55863

55864

55865

55866

55867

The *renameat()* function shall be equivalent to the *rename()* function except in the case where either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *oldfd* instead of the current working directory. If *new* is a relative path, the same happens only relative to the directory associated with *newfd*. It is unspecified whether directory searches are permitted based on whether the file was opened with search permission or on the current permissions of the directory underlying the file descriptor.

55868

55869

If *renameat()* is passed the special value *AT_FDCWD* in the *oldfd* or *newfd* parameter, the current working directory shall be used in the determination of the file for the respective *path* parameter.

55870

RETURN VALUE

55871

CX Upon successful completion, the *rename()* function shall return 0. Otherwise, it shall return *-1*, *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be changed or created.

55874

55875

CX Upon successful completion, the *renameat()* function shall return 0. Otherwise, it shall return *-1* and set *errno* to indicate the error.

55876

ERRORS

55877

CX The *rename()* and *renameat()* functions shall fail if:

55878

55879

55880

55881

CX **[EACCES]** A component of either path prefix denies search permission; or one of the directories containing *old* or *new* denies write permissions; or, write permission is required and is denied for a directory pointed to by the *old* or *new* arguments.

55882

55883

CX **[EBUSY]** The directory named by *old* or *new* is currently in use by the system or another process, and the implementation considers this an error.

55884

55885

CX **[EEXIST] or [ENOTEMPTY]** The link named by *new* is a directory that is not an empty directory.

55886

55887

55888

CX **[EINVAL]** The *new* directory pathname contains a path prefix that names the *old* directory, or either *pathname* argument contains a final component that is dot or dot-dot.

55889

CX **[EIO]** A physical I/O error has occurred.

55890

55891

CX **[EISDIR]** The *new* argument points to a directory and the *old* argument points to a file that is not a directory.

55892

55893

CX **[ELOOP]** A loop exists in symbolic links encountered during resolution of the *path* argument.

55894

55895

CX **[EMLINK]** The file named by *old* is a directory, and the link count of the parent directory of *new* would exceed *{LINK_MAX}*.

55896

55897

55898

CX **[ENAMETOOLONG]** The length of the *old* or *new* argument exceeds *{PATH_MAX}* or a pathname component is longer than *{NAME_MAX}*.

55899

55900

CX **[ENOENT]** The link named by *old* does not name an existing file, or either *old* or *new* points to an empty string.

55901

CX **[ENOSPC]** The directory that would contain *new* cannot be extended.

55902

55903

CX **[ENOTDIR]** A component of either path prefix is not a directory; or the *old* argument names a directory and *new* argument names a non-directory file.

rename()

System Interfaces

55904	XSI	[EPERM] or [EACCES]	
55905			The S_ISVTX flag is set on the directory containing the file referred to by <i>old</i>
55906			and the caller is not the file owner, nor is the caller the directory owner, nor
55907			does the caller have appropriate privileges; or <i>new</i> refers to an existing file, the
55908			S_ISVTX flag is set on the directory containing this file, and the caller is not
55909			the file owner, nor is the caller the directory owner, nor does the caller have
55910			appropriate privileges.
55911	CX	[EROFS]	The requested operation requires writing in a directory on a read-only file
55912			system.
55913	CX	[EXDEV]	The links named by <i>new</i> and <i>old</i> are on different file systems and the
55914			implementation does not support links between file systems.
55915	CX		In addition, the <i>renameat()</i> function shall fail if:
55916		[EBADF]	The <i>old</i> argument does not specify an absolute path and the <i>oldfd</i> argument is
55917			neither AT_FDCWD nor a valid file descriptor open for reading, or the <i>new</i>
55918			argument does not specify an absolute path and the <i>newfd</i> argument is neither
55919			AT_FDCWD nor a valid file descriptor open for reading.
55920	CX		The <i>rename()</i> and <i>renameat()</i> functions may fail if:
55921	OB XSR	[EBUSY]	The file named by the <i>old</i> or <i>new</i> arguments is a named STREAM.
55922	CX	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during
55923			resolution of the <i>path</i> argument.
55924	CX	[ENAMETOOLONG]	
55925			As a result of encountering a symbolic link in resolution of the <i>path</i> argument,
55926			the length of the substituted pathname string exceeded {PATH_MAX}.
55927	CX	[ETXTBSY]	The file to be renamed is a pure procedure (shared text) file that is being
55928			executed.
55929	CX		The <i>renameat()</i> function may fail if:
55930		[ENOTDIR]	The <i>old</i> argument is not an absolute path and <i>oldfd</i> is neither AT_FDCWD nor
55931			a file descriptor associated with a directory, or the <i>new</i> argument is not an
55932			absolute path and <i>newfd</i> is neither AT_FDCWD nor a file descriptor associated
55933			with a directory.

EXAMPLES**Renaming a File**

The following example shows how to rename a file named `/home/cnd/mod1` to `/home/cnd/mod2`.

```
#include <stdio.h>

int status;
...
status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

RATIONALE

This *rename()* function is equivalent for regular files to that defined by the ISO C standard. Its inclusion here expands that definition to include actions on directories and specifies behavior when the *new* parameter names a file that already exists. That specification requires that the action of the function be atomic.

One of the reasons for introducing this function was to have a means of renaming directories while permitting implementations to prohibit the use of *link()* and *unlink()* with directories, thus constraining links to directories to those made by *mkdir()*.

The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
rename("x", "x");
```

does not remove the file.

Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

See also the descriptions of [ENOTEMPTY] and [ENAMETOOLONG] in *rmdir* and [EBUSY] in *unlink*. For a discussion of [EXDEV], see *link*.

The purpose of the *renameat()* function is to rename files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *rename()*, resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the *renameat()* function it can be guaranteed that that renamed file is located correctly and the resulting file is in the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

link, *rmdir*, *symlink()*, *unlink*

XBD <stdio.h>

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The [EBUSY] error is added to the optional part of the ERRORS section.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.
- The [ETXTBSY] optional error condition is added.

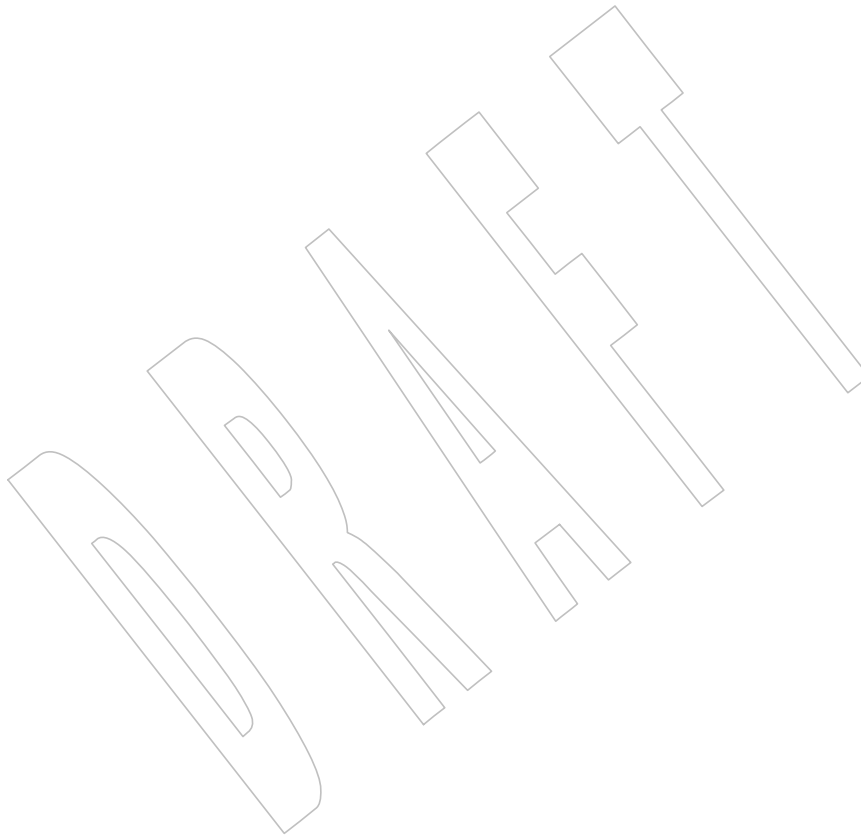
The following changes were made to align with the IEEE P1003.1a draft standard:

- Details are added regarding the treatment of symbolic links.
- The [ELOOP] optional error condition is added.

The normative text is updated to avoid use of the term “must” for application requirements.

rename()

55988	Issue 7	
55989		Austin Group Interpretation 1003.1-2001 #076 is applied, clarifying the behavior if the final component of a path is either dot or dot-dot, and adding the associated [EINVAL] error case.
55990		
55991		The <i>renameat()</i> function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.
55992		
55993		Changes are made related to support for finegrained timestamps.
55994		



55995 **NAME**
 55996 `rewind` — reset the file position indicator in a stream

55997 **SYNOPSIS**
 55998 `#include <stdio.h>`

55999 `void rewind(FILE *stream);`

56000 **DESCRIPTION**

56001 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56002 conflict between the requirements described here and the ISO C standard is unintentional. This
 56003 volume of POSIX.1-200x defers to the ISO C standard.

56004 The call:

56005 `rewind(stream)`

56006 shall be equivalent to:

56007 `(void) fseek(stream, 0L, SEEK_SET)`

56008 except that `rewind()` shall also clear the error indicator.

56009 CX Since `rewind()` does not return a value, an application wishing to detect errors should clear `errno`,
 56010 then call `rewind()`, and if `errno` is non-zero, assume an error has occurred.

56011 **RETURN VALUE**

56012 The `rewind()` function shall not return a value.

56013 **ERRORS**

56014 CX Refer to `fseek()` with the exception of [EINVAL] which does not apply.

56015 **EXAMPLES**

56016 None.

56017 **APPLICATION USAGE**

56018 None.

56019 **RATIONALE**

56020 None.

56021 **FUTURE DIRECTIONS**

56022 None.

56023 **SEE ALSO**

56024 `fseek()`

56025 XBD `<stdio.h>`

56026 **CHANGE HISTORY**

56027 First released in Issue 1. Derived from Issue 1 of the SVID.

56028 **Issue 6**

56029 Extensions beyond the ISO C standard are marked.

56030 **NAME**56031 `rewinddir` — reset the position of a directory stream to the beginning of a directory56032 **SYNOPSIS**56033 `#include <dirent.h>`56034 `void rewinddir(DIR *dirp);`56035 **DESCRIPTION**

56036 The `rewinddir()` function shall reset the position of the directory stream to which `dirp` refers to the
 56037 beginning of the directory. It shall also cause the directory stream to refer to the current state of
 56038 the corresponding directory, as a call to `opendir()` would have done. If `dirp` does not refer to a
 56039 directory stream, the effect is undefined.

56040 After a call to the `fork()` function, either the parent or child (but not both) may continue
 56041 processing the directory stream using `readdir()`, `rewinddir()`, or `seekdir()`. If both the parent and
 56042 child processes use these functions, the result is undefined.

56043 **RETURN VALUE**56044 The `rewinddir()` function shall not return a value.56045 **ERRORS**

56046 No errors are defined.

56047 **EXAMPLES**

56048 None.

56049 **APPLICATION USAGE**

56050 The `rewinddir()` function should be used in conjunction with `opendir()`, `readdir()`, and `closedir()` to
 56051 examine the contents of the directory. This method is recommended for portability.

56052 **RATIONALE**

56053 None.

56054 **FUTURE DIRECTIONS**

56055 None.

56056 **SEE ALSO**56057 `closedir()`, `fdopendir()`, `readdir()`56058 XBD `<dirent.h>`, `<sys/types.h>`56059 **CHANGE HISTORY**

56060 First released in Issue 2.

56061 **Issue 6**56062 In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

56063 The following new requirements on POSIX implementations derive from alignment with the
 56064 Single UNIX Specification:

- 56065 • The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was
 56066 required for conforming implementations of previous POSIX specifications, it was not
 56067 required for UNIX applications.

56068 **NAME**
 56069 rint, rintf, rintl — round-to-nearest integral value

56070 **SYNOPSIS**
 56071 #include <math.h>
 56072 double rint(double x);
 56073 float rintf(float x);
 56074 long double rintl(long double x);

56075 DESCRIPTION

56076 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56077 conflict between the requirements described here and the ISO C standard is unintentional. This
 56078 volume of POSIX.1-200x defers to the ISO C standard.

56079 These functions shall return the integral value (represented as a **double**) nearest x in the
 56080 direction of the current rounding mode. The current rounding mode is implementation-defined.

56081 If the current rounding mode rounds toward negative infinity, then *rint()* shall be equivalent to
 56082 *floor()*. If the current rounding mode rounds toward positive infinity, then *rint()* shall be
 56083 equivalent to *ceil()*.

56084 These functions differ from the *nearbyint()*, *nearbyintf()*, and *nearbyintl()* functions only in that
 56085 they may raise the inexact floating-point exception if the result differs in value from the
 56086 argument.

56087 An application wishing to check for error situations should set *errno* to zero and call
 56088 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 56089 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 56090 zero, an error has occurred.

56091 RETURN VALUE

56092 Upon successful completion, these functions shall return the integer (represented as a double
 56093 precision number) nearest x in the direction of the current rounding mode.

56094 MX If x is NaN, a NaN shall be returned.

56095 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

56096 XSI If the correct value would cause overflow, a range error shall occur and *rint()*, *rintf()*, and *rintl()*
 56097 shall return the value of the macro $\pm \text{HUGE_VAL}$, $\pm \text{HUGE_VALF}$, and $\pm \text{HUGE_VALL}$ (with the
 56098 same sign as x), respectively.

56099 ERRORS

56100 These functions shall fail if:

56101 XSI **Range Error** The result would cause an overflow.

56102 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56103 then *errno* shall be set to [ERANGE]. If the integer expression
 56104 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 56105 floating-point exception shall be raised.

56106
56107
56108
56109
56110
56111
56112
56113
56114
56115
56116
56117
56118
56119
56120
56121
56122
56123
56124
56125
56126
56127
56128
56129

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs(), *ceil()*, *feclearexcept()*, *fetestexcept()*, *floor()*, *isnan()*, *nearbyint()*

XBD Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 4, Version 2.

Issue 5

Moved from X/OPEN UNIX extension to BASE.

Issue 6

The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *rintf()* and *rintl()* functions are added.
- The *rint()* function is no longer marked as an extension.
- The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

56130 **NAME**56131 `rmdir` — remove a directory56132 **SYNOPSIS**56133 `#include <unistd.h>`56134 `int rmdir(const char *path);`56135 **DESCRIPTION**56136 The `rmdir()` function shall remove a directory whose name is given by *path*. The directory shall
56137 be removed only if it is an empty directory.56138 If the directory is the root directory or the current working directory of any process, it is
56139 unspecified whether the function succeeds, or whether it shall fail and set *errno* to [EBUSY].56140 If *path* names a symbolic link, then `rmdir()` shall fail and set *errno* to [ENOTDIR].56141 If the *path* argument refers to a path whose final component is either dot or dot-dot, `rmdir()` shall
56142 fail.56143 If the directory's link count becomes 0 and no process has the directory open, the space occupied
56144 by the directory shall be freed and the directory shall no longer be accessible. If one or more
56145 processes have the directory open when the last link is removed, the dot and dot-dot entries, if
56146 present, shall be removed before `rmdir()` returns and no new entries may be created in the
56147 directory, but the directory shall not be removed until all references to the directory are closed.56148 If the directory is not an empty directory, `rmdir()` shall fail and set *errno* to [EEXIST] or
56149 [ENOTEMPTY].56150 Upon successful completion, `rmdir()` shall mark for update the last data modification and last
56151 file status change timestamps of the parent directory.56152 **RETURN VALUE**56153 Upon successful completion, the function `rmdir()` shall return 0. Otherwise, -1 shall be returned,
56154 and *errno* set to indicate the error. If -1 is returned, the named directory shall not be changed.56155 **ERRORS**56156 The `rmdir()` function shall fail if:56157 [EACCES] Search permission is denied on a component of the path prefix, or write
56158 permission is denied on the parent directory of the directory to be removed.56159 [EBUSY] The directory to be removed is currently in use by the system or some process
56160 and the implementation considers this to be an error.56161 [EEXIST] or [ENOTEMPTY] The *path* argument names a directory that is not an empty directory, or there
56162 are hard links to the directory other than dot or a single entry in dot-dot.
5616356164 [EINVAL] The *path* argument contains a last component that is dot.

56165 [EIO] A physical I/O error has occurred.

56166 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
56167 argument.56168 [ENAMETOOLONG] The length of the *path* argument exceeds {PATH_MAX} or a pathname
56169 component is longer than {NAME_MAX}.
56170

56171	[ENOENT]	A component of <i>path</i> does not name an existing file, or the <i>path</i> argument names a nonexistent directory or points to an empty string.
56172		
56173	[ENOTDIR]	A component of <i>path</i> is not a directory.
56174	XSI [EPERM] or [EACCES]	The S_ISVTX flag is set on the parent directory of the directory to be removed and the caller is not the owner of the directory to be removed, nor is the caller the owner of the parent directory, nor does the caller have the appropriate privileges.
56175		
56176		
56177		
56178		
56179	[EROFS]	The directory entry to be removed resides on a read-only file system.
56180		The <i>rmdir()</i> function may fail if:
56181	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
56182		
56183	[ENAMETOOLONG]	As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.
56184		
56185		

EXAMPLES**Removing a Directory**

The following example shows how to remove a directory named **/home/cnd/mod1**.

```
#include <unistd.h>

int status;
...
status = rmdir("/home/cnd/mod1");
```

APPLICATION USAGE

None.

RATIONALE

The *rmdir()* and *rename()* functions originated in 4.2 BSD, and they used [ENOTEMPTY] for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained [EEXIST] instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both [EEXIST] and [ENOTEMPTY] in **<errno.h>** with distinct values, so that applications can use both values in C-language **case** statements.

The meaning of deleting *pathname/dot* is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir()* when there are multiple hard links to the directory being removed. The requirement to set *errno* to [EEXIST] or [ENOTEMPTY] clarifies the behavior in this case.

If the current working directory of the process is being removed, that should be an allowed error.

Virtually all existing implementations detect [ENOTEMPTY] or the case of dot-dot. The text in [Section 2.3](#) (on page 456) about returning any one of the possible errors permits that behavior to continue. The [ELOOP] error may be returned if more than {SYMLOOP_MAX} symbolic links are encountered during resolution of the *path* argument.

56216
56217
56218
56219
56220
56221
56222
56223
56224
56225
56226
56227
56228
56229
56230
56231
56232
56233

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3 (on page 456), *mkdir*, *remove()*, *rename()*, *unlink*

XBD <*unistd.h*>

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION is updated to indicate the results of naming a symbolic link in *path*.
- The [EIO] mandatory error condition is added.
- The [ELOOP] mandatory error condition is added.
- A second [ENAMETOOLONG] is added as an optional error condition.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The [ELOOP] optional error condition is added.

Issue 7

Changes are made related to support for finegrained timestamps.

DRAFT

56234 **NAME**

56235 round, roundf, roundl — round to the nearest integer value in a floating-point format

56236 **SYNOPSIS**

```
56237 #include <math.h>
56238
56238 double round(double x);
56239 float roundf(float x);
56240 long double roundl(long double x);
```

56241 **DESCRIPTION**

56242 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56243 conflict between the requirements described here and the ISO C standard is unintentional. This
 56244 volume of POSIX.1-200x defers to the ISO C standard.

56245 These functions shall round their argument to the nearest integer value in floating-point format,
 56246 rounding halfway cases away from zero, regardless of the current rounding direction.

56247 An application wishing to check for error situations should set *errno* to zero and call
 56248 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 56249 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 56250 zero, an error has occurred.

56251 **RETURN VALUE**

56252 Upon successful completion, these functions shall return the rounded integer value.

56253 MX If *x* is NaN, a NaN shall be returned.56254 If *x* is ± 0 or $\pm \text{Inf}$, *x* shall be returned.

56255 XSI If the correct value would cause overflow, a range error shall occur and *round()*, *roundf()*, and
 56256 *roundl()* shall return the value of the macro $\pm \text{HUGE_VAL}$, $\pm \text{HUGE_VALF}$, and $\pm \text{HUGE_VALL}$
 56257 (with the same sign as *x*), respectively.

56258 **ERRORS**

56259 These functions may fail if:

56260 XSI **Range Error** The result overflows.

56261 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56262 then *errno* shall be set to [ERANGE]. If the integer expression
 56263 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 56264 floating-point exception shall be raised.

56265 **EXAMPLES**

56266 None.

56267 **APPLICATION USAGE**

56268 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 56269 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

56270 **RATIONALE**

56271 None.

56272 **FUTURE DIRECTIONS**

56273 None.

56274

SEE ALSO

56275

feclearexcept(), *fetestexcept()*

56276

XBD Section 4.19 (on page 104), `<math.h>`

56277

CHANGE HISTORY

56278

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

DRAFT

56279 **NAME**

56280 scalbn, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX

56281 **SYNOPSIS**

```
56282 #include <math.h>
56283
56283 double scalbn(double x, long n);
56284 float scalblnf(float x, long n);
56285 long double scalblnl(long double x, long n);
56286 double scalbn(double x, int n);
56287 float scalbnf(float x, int n);
56288 long double scalbnl(long double x, int n);
```

56289 **DESCRIPTION**

56290 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 56291 conflict between the requirements described here and the ISO C standard is unintentional. This
 56292 volume of POSIX.1-200x defers to the ISO C standard.

56293 These functions shall compute $x * FLT_RADIX^n$ efficiently, not normally by computing
 56294 FLT_RADIX^n explicitly.

56295 An application wishing to check for error situations should set *errno* to zero and call
 56296 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 56297 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 56298 zero, an error has occurred.

56299 **RETURN VALUE**56300 Upon successful completion, these functions shall return $x * FLT_RADIX^n$.

56301 If the result would cause overflow, a range error shall occur and these functions shall return
 56302 $\pm HUGE_VAL$, $\pm HUGE_VALF$, and $\pm HUGE_VALL$ (according to the sign of x) as appropriate for
 56303 the return type of the function.

56304 If the correct value would cause underflow, and is not representable, a range error may occur,
 56305 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

56306 MX If x is NaN, a NaN shall be returned.

56307 If x is ± 0 or $\pm Inf$, x shall be returned.

56308 If n is 0, x shall be returned.

56309 If the correct value would cause underflow, and is representable, a range error may occur and
 56310 the correct value shall be returned.

56311 **ERRORS**

56312 These functions shall fail if:

56313 Range Error The result overflows.

56314 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56315 then *errno* shall be set to [ERANGE]. If the integer expression
 56316 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 56317 floating-point exception shall be raised.

56318 These functions may fail if:

56319 Range Error The result underflows.

56320 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 56321 then *errno* shall be set to [ERANGE]. If the integer expression

(*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are named so as to avoid conflicting with the historical definition of the *scalb()* function from the Single UNIX Specification. The difference is that the *scalb()* function has a second argument of **double** instead of **int**. The *scalb()* function is not part of the ISO C standard. The three functions whose second type is **long** are provided because the factor required to scale from the smallest positive floating-point value to the largest finite one, on many implementations, is too large to represent in the minimum-width **int** format.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*

XBD Section 4.19 (on page 104), [<math.h>](#)

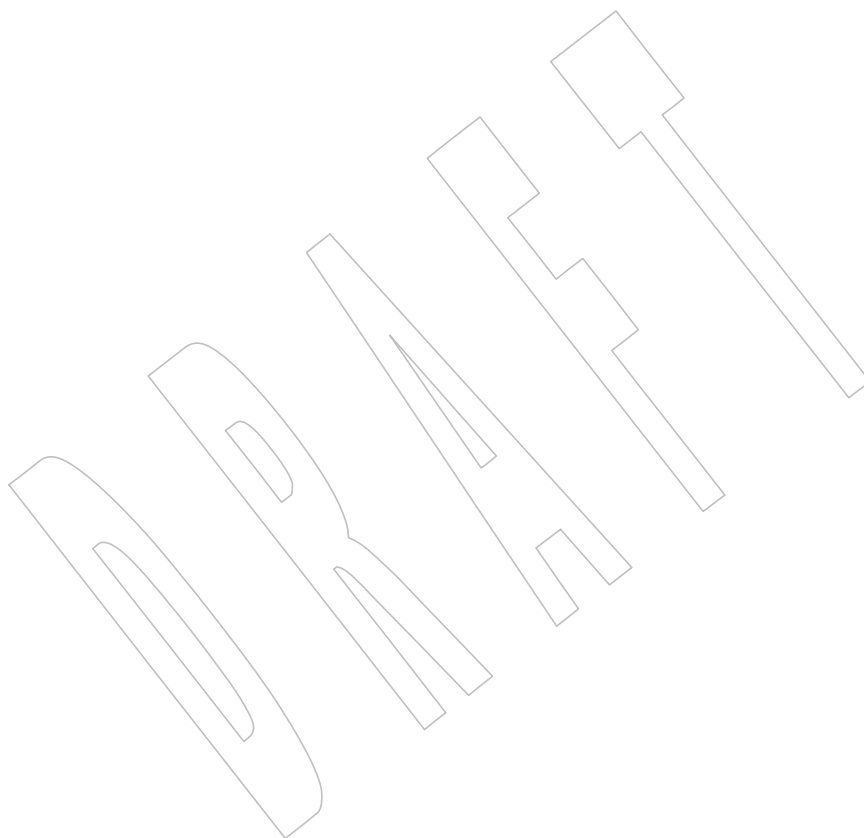
CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

56343 **NAME**
56344 `scandir` — scan a directory

56345 **SYNOPSIS**
56346 `#include <dirent.h>`
56347 `int scandir(const char *dir, struct dirent ***namelist,`
56348 `int (*sel)(const struct dirent *),`
56349 `int (*compar)(const struct dirent **, const struct dirent **));`

56350 **DESCRIPTION**
56351 Refer to *alphasort()*.



56352

NAME

56353

scanf — convert formatted input

56354

SYNOPSIS

56355

#include <stdio.h>

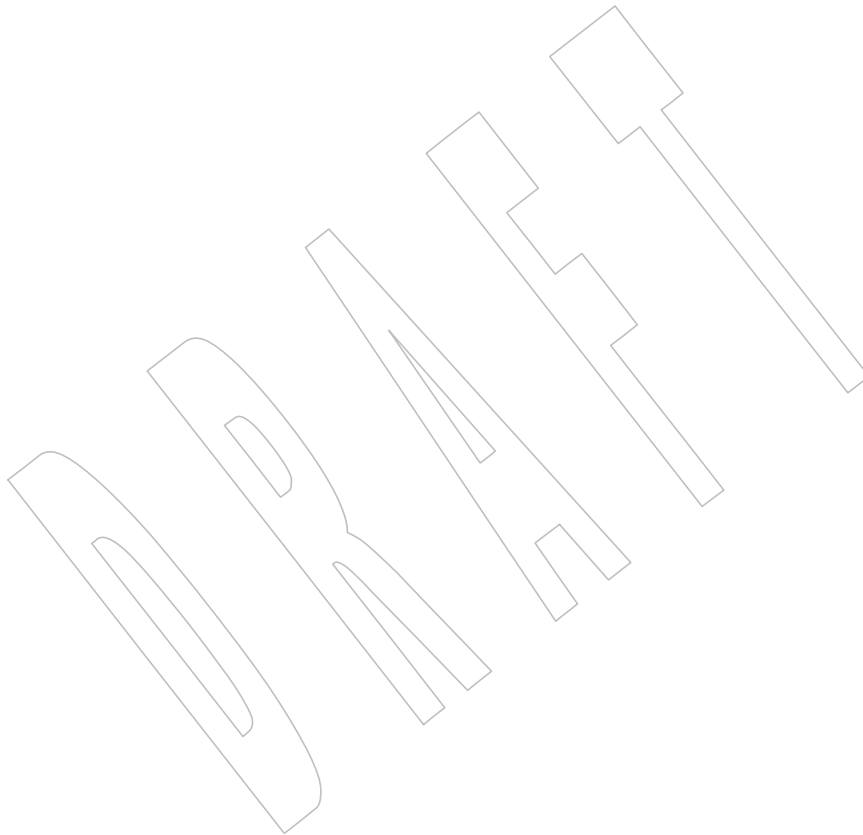
56356

int scanf(const char *restrict *format*, ...);

56357

DESCRIPTION

56358

Refer to *fscanf()*.

56359 **NAME**56360 sched_get_priority_max, sched_get_priority_min — get priority limits (**REALTIME**)56361 **SYNOPSIS**

```
56362 PS|TPS #include <sched.h>
56363 int sched_get_priority_max(int policy);
56364 int sched_get_priority_min(int policy);
```

56365 **DESCRIPTION**56366 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the appropriate
56367 maximum or minimum, respectively, for the scheduling policy specified by *policy*.56368 The value of *policy* shall be one of the scheduling policy values defined in **<sched.h>**.56369 **RETURN VALUE**56370 If successful, the *sched_get_priority_max()* and *sched_get_priority_min()* functions shall return the
56371 appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a
56372 value of -1 and set *errno* to indicate the error.56373 **ERRORS**56374 The *sched_get_priority_max()* and *sched_get_priority_min()* functions shall fail if:56375 [EINVAL] The value of the *policy* parameter does not represent a defined scheduling
56376 policy.56377 **EXAMPLES**

56378 None.

56379 **APPLICATION USAGE**

56380 None.

56381 **RATIONALE**

56382 None.

56383 **FUTURE DIRECTIONS**

56384 None.

56385 **SEE ALSO**56386 *sched_getparam()*, *sched_setparam()*, *sched_getscheduler()*, *sched_rr_get_interval()*,
56387 *sched_setscheduler()*56388 XBD **<sched.h>** +56389 **CHANGE HISTORY**

56390 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56391 **Issue 6**

56392 These functions are marked as part of the Process Scheduling option.

56393 The [ENOSYS] error condition has been removed as stubs need not be provided if an
56394 implementation does not support the Process Scheduling option.56395 The [ESRCH] error condition has been removed since these functions do not take a *pid*
56396 argument.56397 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/52 is applied, changing the PS margin
56398 code in the SYNOPSIS to PS|TPS.

56399 **NAME**56400 sched_getparam — get scheduling parameters (**REALTIME**)56401 **SYNOPSIS**

```
56402 PS #include <sched.h>
56403 int sched_getparam(pid_t pid, struct sched_param *param);
```

56404 **DESCRIPTION**

56405 The *sched_getparam()* function shall return the scheduling parameters of a process specified by
56406 *pid* in the **sched_param** structure pointed to by *param*.

56407 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
56408 parameters for the process whose process ID is equal to *pid* shall be returned.

56409 If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of
56410 the *sched_getparam()* function is unspecified if the value of *pid* is negative.

56411 **RETURN VALUE**

56412 Upon successful completion, the *sched_getparam()* function shall return zero. If the call to
56413 *sched_getparam()* is unsuccessful, the function shall return a value of -1 and set *errno* to indicate
56414 the error.

56415 **ERRORS**

56416 The *sched_getparam()* function shall fail if:

56417 [EPERM] The requesting process does not have permission to obtain the scheduling
56418 parameters of the specified process.

56419 [ESRCH] No process can be found corresponding to that specified by *pid*.

56420 **EXAMPLES**

56421 None.

56422 **APPLICATION USAGE**

56423 None.

56424 **RATIONALE**

56425 None.

56426 **FUTURE DIRECTIONS**

56427 None.

56428 **SEE ALSO**

56429 [sched_getscheduler\(\)](#), [sched_setparam\(\)](#), [sched_setscheduler\(\)](#)

56430 XBD [<sched.h>](#)

56431 **CHANGE HISTORY**

56432 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56433 **Issue 6**

56434 The *sched_getparam()* function is marked as part of the Process Scheduling option.

56435 The [ENOSYS] error condition has been removed as stubs need not be provided if an
56436 implementation does not support the Process Scheduling option.

56437 **NAME**56438 sched_getscheduler — get scheduling policy (**REALTIME**)56439 **SYNOPSIS**

```
56440 PS #include <sched.h>
56441 int sched_getscheduler(pid_t pid);
```

56442 **DESCRIPTION**

56443 The *sched_getscheduler()* function shall return the scheduling policy of the process specified by
 56444 *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler()* function is
 56445 unspecified.

56446 The values that can be returned by *sched_getscheduler()* are defined in the **<sched.h>** header.

56447 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 56448 policy shall be returned for the process whose process ID is equal to *pid*.

56449 If *pid* is zero, the scheduling policy shall be returned for the calling process.

56450 **RETURN VALUE**

56451 Upon successful completion, the *sched_getscheduler()* function shall return the scheduling policy
 56452 of the specified process. If unsuccessful, the function shall return -1 and set *errno* to indicate the
 56453 error.

56454 **ERRORS**

56455 The *sched_getscheduler()* function shall fail if:

56456 [EPERM] The requesting process does not have permission to determine the scheduling
 56457 policy of the specified process.

56458 [ESRCH] No process can be found corresponding to that specified by *pid*.

56459 **EXAMPLES**

56460 None.

56461 **APPLICATION USAGE**

56462 None.

56463 **RATIONALE**

56464 None.

56465 **FUTURE DIRECTIONS**

56466 None.

56467 **SEE ALSO**

56468 *sched_getparam()*, *sched_setparam()*, *sched_setscheduler()*

56469 XBD **<sched.h>**

56470 **CHANGE HISTORY**

56471 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56472 **Issue 6**

56473 The *sched_getscheduler()* function is marked as part of the Process Scheduling option.

56474 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56475 implementation does not support the Process Scheduling option.

56476 **NAME**56477 sched_rr_get_interval — get execution time limits (**REALTIME**)56478 **SYNOPSIS**

56479 PS|TPS #include <sched.h>

56480 int sched_rr_get_interval(pid_t pid, struct timespec *interval);

56481 **DESCRIPTION**

56482 The *sched_rr_get_interval()* function shall update the **timespec** structure referenced by the
 56483 *interval* argument to contain the current execution time limit (that is, time quantum) for the
 56484 process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process
 56485 shall be returned.

56486 **RETURN VALUE**

56487 If successful, the *sched_rr_get_interval()* function shall return zero. Otherwise, it shall return a
 56488 value of -1 and set *errno* to indicate the error.

56489 **ERRORS**56490 The *sched_rr_get_interval()* function shall fail if:56491 [ESRCH] No process can be found corresponding to that specified by *pid*.56492 **EXAMPLES**

56493 None.

56494 **APPLICATION USAGE**

56495 None.

56496 **RATIONALE**

56497 None.

56498 **FUTURE DIRECTIONS**

56499 None.

56500 **SEE ALSO**

56501 *sched_getparam()*, *sched_get_priority_max()*, *sched_getscheduler()*, *sched_setparam()*,
 56502 *sched_setscheduler()*

56503 XBD <[sched.h](#)>56504 **CHANGE HISTORY**

56505 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56506 **Issue 6**56507 The *sched_rr_get_interval()* function is marked as part of the Process Scheduling option.

56508 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56509 implementation does not support the Process Scheduling option.

56510 IEEE Std 1003.1-2001/Cor 1-2002, XSH/TC1/D6/53 is applied, changing the PS margin code in
 56511 the SYNOPSIS to PS|TPS.

56512 **NAME**56513 sched_setparam — set scheduling parameters (**REALTIME**)56514 **SYNOPSIS**

```
56515 PS #include <sched.h>
56516 int sched_setparam(pid_t pid, const struct sched_param *param);
```

56517 **DESCRIPTION**

56518 The *sched_setparam()* function shall set the scheduling parameters of the process specified by *pid*
 56519 to the values specified by the **sched_param** structure pointed to by *param*. The value of the
 56520 *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive
 56521 priority range for the current scheduling policy of the process specified by *pid*. Higher
 56522 numerical values for the priority represent higher priorities. If the value of *pid* is negative, the
 56523 behavior of the *sched_setparam()* function is unspecified.

56524 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 56525 parameters shall be set for the process whose process ID is equal to *pid*.

56526 If *pid* is zero, the scheduling parameters shall be set for the calling process.

56527 The conditions under which one process has permission to change the scheduling parameters of
 56528 another process are implementation-defined.

56529 Implementations may require the requesting process to have the appropriate privilege to set its
 56530 own scheduling parameters or those of another process.

56531 See [Scheduling Policies](#) (on page 479) for a description on how this function affects the
 56532 scheduling of the threads within the target process.

56533 SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or
 56534 SCHED_SPORADIC, the result is implementation-defined; this case includes the
 56535 SCHED_OTHER policy.

56536 SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 56537 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

56538 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 56539 function to succeed; if not, the function shall fail. It is unspecified whether the
 56540 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 56541 rounded to align with the resolution of the clock being used.

56542 This function is not atomic with respect to other threads in the process. Threads may continue to
 56543 execute while this function call is in the process of changing the scheduling policy for the
 56544 underlying kernel-scheduled entities used by the process contention scope threads.

56545 **RETURN VALUE**

56546 If successful, the *sched_setparam()* function shall return zero.

56547 If the call to *sched_setparam()* is unsuccessful, the priority shall remain unchanged, and the
 56548 function shall return a value of -1 and set *errno* to indicate the error.

56549 **ERRORS**

56550 The *sched_setparam()* function shall fail if:

56551 [EINVAL] One or more of the requested scheduling parameters is outside the range
 56552 defined for the scheduling policy of the specified *pid*.

56553 [EPERM] The requesting process does not have permission to set the scheduling
 56554 parameters for the specified process, or does not have the appropriate
 56555 privilege to invoke *sched_setparam()*.

56556 [ESRCH] No process can be found corresponding to that specified by *pid*.

EXAMPLES

56557 None.
 56558

APPLICATION USAGE

56559 None.
 56560

RATIONALE

56561 None.
 56562

FUTURE DIRECTIONS

56563 None.
 56564

SEE ALSO

56565 [Scheduling Policies](#) (on page 479), [sched_getparam\(\)](#), [sched_getscheduler\(\)](#), [sched_setscheduler\(\)](#)
 56566

56567 XBD [<sched.h>](#)

CHANGE HISTORY

56568 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.
 56569

Issue 6

56570 The *sched_setparam()* function is marked as part of the Process Scheduling option.
 56571

56572 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56573 implementation does not support the Process Scheduling option.

56574 The following new requirements on POSIX implementations derive from alignment with the
 56575 Single UNIX Specification:

- 56576 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
 56577 added.
- 56578 • Sections describing two-level scheduling and atomicity of the function are added.

56579 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

56580 IEEE PASC Interpretation 1003.1 #100 is applied.

Issue 7

56581 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.
 56582

56583 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements +
 56584 for the *sched_ss_repl_period* and *sched_ss_init_budget* values.

56585 **NAME**56586 sched_setscheduler — set scheduling policy and parameters (**REALTIME**)56587 **SYNOPSIS**

```
56588 PS #include <sched.h>
56589
56589 int sched_setscheduler(pid_t pid, int policy,
56590 const struct sched_param *param);
```

56591 **DESCRIPTION**

56592 The *sched_setscheduler()* function shall set the scheduling policy and scheduling parameters of
 56593 the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure
 56594 pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param**
 56595 structure shall be any integer within the inclusive priority range for the scheduling policy
 56596 specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler()*
 56597 function is unspecified.

56598 The possible values for the *policy* parameter are defined in the **<sched.h>** header.

56599 If a process specified by *pid* exists, and if the calling process has permission, the scheduling
 56600 policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

56601 If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling
 56602 process.

56603 The conditions under which one process has the appropriate privilege to change the scheduling
 56604 parameters of another process are implementation-defined.

56605 Implementations may require that the requesting process have permission to set its own
 56606 scheduling parameters or those of another process. Additionally, implementation-defined
 56607 restrictions may apply as to the appropriate privileges required to set the scheduling policy of
 56608 the process, or the scheduling policy of another process, to a particular value.

56609 The *sched_setscheduler()* function shall be considered successful if it succeeds in setting the
 56610 scheduling policy and scheduling parameters of the process specified by *pid* to the values
 56611 specified by *policy* and the structure pointed to by *param*, respectively.

56612 See [Scheduling Policies](#) (on page 479) for a description on how this function affects the
 56613 scheduling of the threads within the target process.

56614 SS If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or
 56615 SCHED_SPORADIC, the result is implementation-defined; this case includes the
 56616 SCHED_OTHER policy.

56617 SS The specified *sched_ss_repl_period* shall be greater than or equal to the specified
 56618 *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

56619 The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the
 56620 function to succeed; if not, the function shall fail. It is unspecified whether the
 56621 *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are
 56622 rounded to align with the resolution of the clock being used.

56623 This function is not atomic with respect to other threads in the process. Threads may continue to
 56624 execute while this function call is in the process of changing the scheduling policy and
 56625 associated scheduling parameters for the underlying kernel-scheduled entities used by the
 56626 process contention scope threads.

56627 **RETURN VALUE**

56628 Upon successful completion, the function shall return the former scheduling policy of the
 56629 specified process. If the `sched_setscheduler()` function fails to complete successfully, the policy
 56630 and scheduling parameters shall remain unchanged, and the function shall return a value of `-1`
 56631 and set `errno` to indicate the error.

56632 **ERRORS**

56633 The `sched_setscheduler()` function shall fail if:

- 56634 [EINVAL] The value of the `policy` parameter is invalid, or one or more of the parameters
 56635 contained in `param` is outside the valid range for the specified scheduling
 56636 policy.
- 56637 [EPERM] The requesting process does not have permission to set either or both of the
 56638 scheduling parameters or the scheduling policy of the specified process.
- 56639 [ESRCH] No process can be found corresponding to that specified by `pid`.

56640 **EXAMPLES**

56641 None.

56642 **APPLICATION USAGE**

56643 None.

56644 **RATIONALE**

56645 None.

56646 **FUTURE DIRECTIONS**

56647 None.

56648 **SEE ALSO**

56649 [Scheduling Policies](#) (on page 479), [sched_getparam\(\)](#), [sched_getscheduler\(\)](#), [sched_setparam\(\)](#)

56650 XBD [<sched.h>](#)

56651 **CHANGE HISTORY**

56652 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56653 **Issue 6**

56654 The `sched_setscheduler()` function is marked as part of the Process Scheduling option.

56655 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56656 implementation does not support the Process Scheduling option.

56657 The following new requirements on POSIX implementations derive from alignment with the
 56658 Single UNIX Specification:

- 56659 • In the DESCRIPTION, the effect of this function on a thread's scheduling parameters is
 56660 added.
- 56661 • Sections describing two-level scheduling and atomicity of the function are added.

56662 The SCHED_SPORADIC scheduling policy is added for alignment with IEEE Std 1003.1d-1999.

56663 **Issue 7**

56664 Austin Group Interpretation 1003.1-2001 #061 is applied, updating the DESCRIPTION.

56665 Austin Group Interpretation 1003.1-2001 #119 is applied, clarifying the accuracy requirements +
 56666 for the `sched_ss_repl_period` and `sched_ss_init_budget` values.

56667 **NAME**

56668 sched_yield — yield the processor

56669 **SYNOPSIS**

56670 #include <sched.h>

56671 int sched_yield(void);

56672 **DESCRIPTION**56673 The *sched_yield()* function shall force the running thread to relinquish the processor until it again
56674 becomes the head of its thread list. It takes no arguments.56675 **RETURN VALUE**56676 The *sched_yield()* function shall return 0 if it completes successfully; otherwise, it shall return a
56677 value of -1 and set *errno* to indicate the error.56678 **ERRORS**

56679 No errors are defined.

56680 **EXAMPLES**

56681 None.

56682 **APPLICATION USAGE**56683 The conceptual model for scheduling semantics in POSIX.1-200x defines a set of thread lists. |
56684 This set of thread lists is always present regardless of the scheduling options supported by the
56685 system. On a system where the Process Scheduling option is not supported, portable
56686 applications should not make any assumptions regarding whether threads from other processes
56687 will be on the same thread list.56688 **RATIONALE**

56689 None.

56690 **FUTURE DIRECTIONS**

56691 None.

56692 **SEE ALSO**56693 XBD [<sched.h>](#) |56694 **CHANGE HISTORY**56695 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
56696 POSIX Threads Extension.56697 **Issue 6**56698 The *sched_yield()* function is now marked as part of the Process Scheduling and Threads options.56699 **Issue 7**

56700 SD5-XSH-ERN-120 is applied, adding APPLICATION USAGE.

56701 The *sched_yield()* function is moved to the Base.

56702 **NAME**
56703 seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

56704 **SYNOPSIS**

56705 XSI #include <stdlib.h>
56706 unsigned short *seed48(unsigned short seed16v[3]);

56707 **DESCRIPTION**

56708 Refer to *drand48()*.

56709 **NAME**
 56710 seekdir — set the position of a directory stream

56711 **SYNOPSIS**

```
56712 XSI #include <dirent.h>
56713 void seekdir(DIR *dirp, long loc);
```

56714 **DESCRIPTION**

56715 The *seekdir()* function shall set the position of the next *readdir()* operation on the directory
 56716 stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been
 56717 returned from an earlier call to *telldir()* using the same directory stream. The new position
 56718 reverts to the one associated with the directory stream when *telldir()* was performed.

56719 If the value of *loc* was not obtained from an earlier call to *telldir()*, or if a call to *rewinddir()*
 56720 occurred between the call to *telldir()* and the call to *seekdir()*, the results of subsequent calls to
 56721 *readdir()* are unspecified.

56722 **RETURN VALUE**

56723 The *seekdir()* function shall not return a value.

56724 **ERRORS**

56725 No errors are defined.

56726 **EXAMPLES**

56727 None.

56728 **APPLICATION USAGE**

56729 None.

56730 **RATIONALE**

56731 The original standard developers perceived that there were restrictions on the use of the
 56732 *seekdir()* and *telldir()* functions related to implementation details, and for that reason these
 56733 functions need not be supported on all POSIX-conforming systems. They are required on
 56734 implementations supporting the XSI option.

56735 One of the perceived problems of implementation is that returning to a given point in a directory
 56736 is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-
 56737 trees, hashing functions, or other similar mechanisms to order their directories are considered.
 56738 The definition of *seekdir()* and *telldir()* does not specify whether, when using these interfaces, a
 56739 given directory entry will be seen at all, or more than once.

56740 On systems not supporting these functions, their capability can sometimes be accomplished by
 56741 saving a filename found by *readdir()* and later using *rewinddir()* and a loop on *readdir()* to
 56742 relocate the position from which the filename was saved.

56743 **FUTURE DIRECTIONS**

56744 None.

56745 **SEE ALSO**

56746 *fdopendir()*, *readdir()*, *telldir()*

56747 XBD [<dirent.h>](#), [<stdio.h>](#), [<sys/types.h>](#)

56748

CHANGE HISTORY

56749

First released in Issue 2.

56750

Issue 6

56751

In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

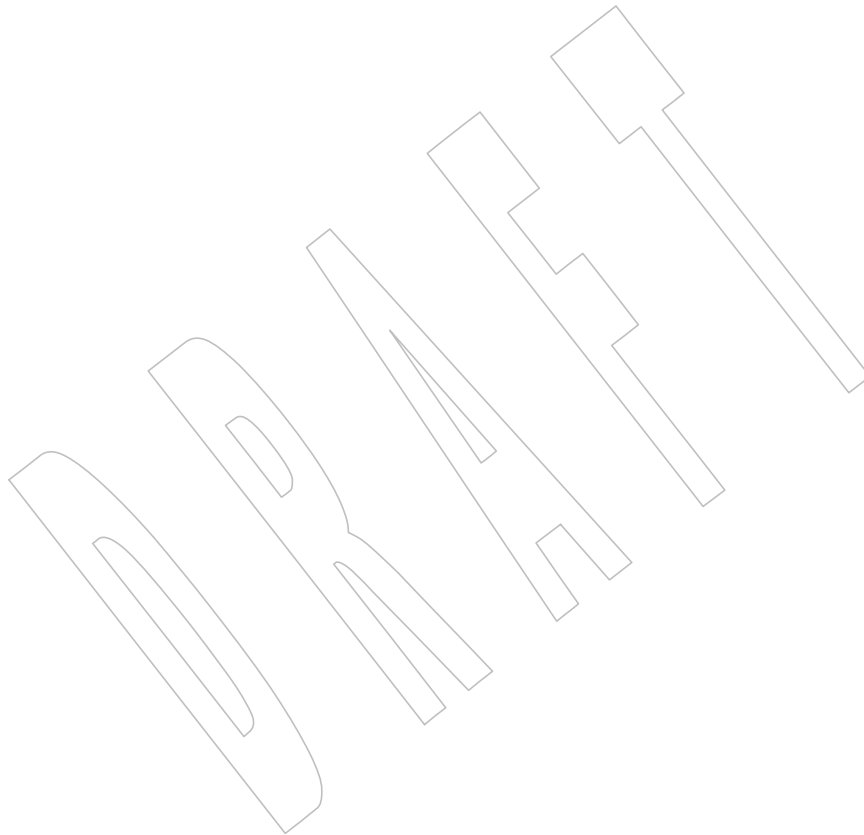
56752

Issue 7

56753

SD5-XSH-ERN-200 is applied, updating the DESCRIPTION to note that the value of *loc* should have been returned from an earlier call to `telldir()` using the same directory stream. +

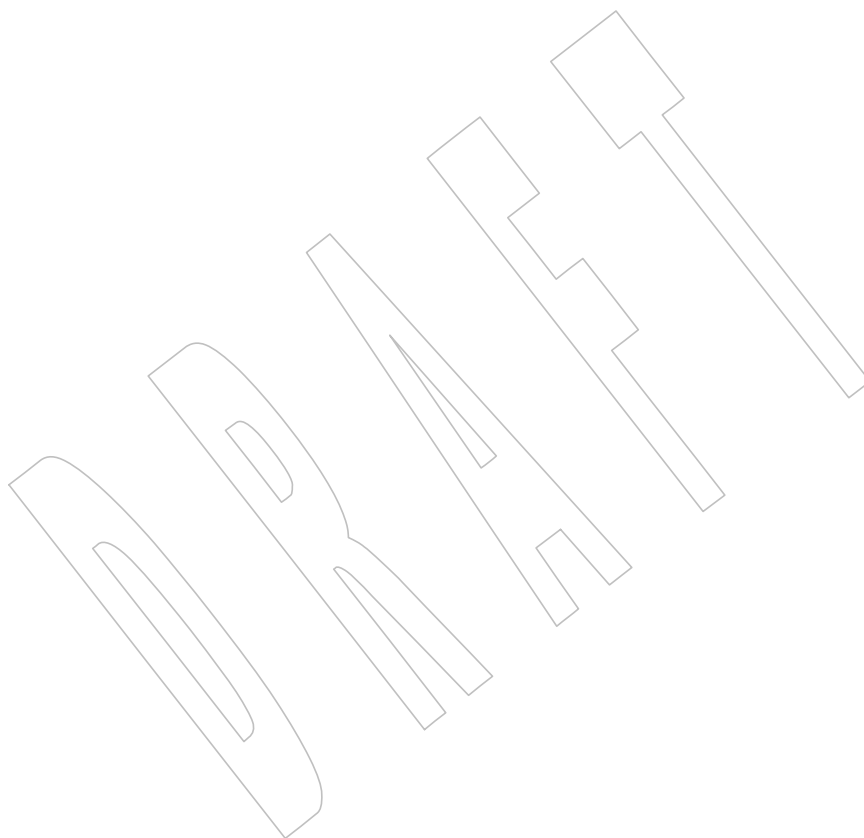
56754



56755 **NAME**
56756 `select` — synchronous I/O multiplexing

56757 **SYNOPSIS**
56758 `#include <sys/select.h>`
56759 `int select(int nfds, fd_set *restrict readfds,`
56760 `fd_set *restrict writefds, fd_set *restrict errorfds,`
56761 `struct timeval *restrict timeout);`

56762 **DESCRIPTION**
56763 Refer to [pselect\(\)](#).



56764 **NAME**
 56765 sem_close — close a named semaphore

56766 **SYNOPSIS**
 56767 #include <semaphore.h>
 56768 int sem_close(sem_t *sem);

56769 **DESCRIPTION**
 56770 The *sem_close()* function shall indicate that the calling process is finished using the named
 56771 semaphore indicated by *sem*. The effects of calling *sem_close()* for an unnamed semaphore (one
 56772 created by *sem_init()*) are undefined. The *sem_close()* function shall deallocate (that is, make
 56773 available for reuse by a subsequent *sem_open()* by this process) any system resources allocated
 56774 by the system for use by this process for this semaphore. The effect of subsequent use of the
 56775 semaphore indicated by *sem* by this process is undefined. If the semaphore has not been
 56776 removed with a successful call to *sem_unlink()*, then *sem_close()* has no effect on the state of the
 56777 semaphore. If the *sem_unlink()* function has been successfully invoked for *name* after the most
 56778 recent call to *sem_open()* with O_CREAT for this semaphore, then when all processes that have
 56779 opened the semaphore close it, the semaphore is no longer accessible.

56780 **RETURN VALUE**
 56781 Upon successful completion, a value of zero shall be returned. Otherwise, a value of -1 shall be
 56782 returned and *errno* set to indicate the error.

56783 **ERRORS**
 56784 The *sem_close()* function may fail if:
 56785 [EINVAL] The *sem* argument is not a valid semaphore descriptor.

56786 **EXAMPLES**
 56787 None.

56788 **APPLICATION USAGE**
 56789 The *sem_close()* function is part of the Semaphores option and need not be available on all
 56790 implementations.

56791 **RATIONALE**
 56792 None.

56793 **FUTURE DIRECTIONS**
 56794 None.

56795 **SEE ALSO**
 56796 *semctl()*, *semget()*, *semop()*, *sem_init()*, *sem_open()*, *sem_unlink()*
 56797 XBD <semaphore.h>

56798 **CHANGE HISTORY**
 56799 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

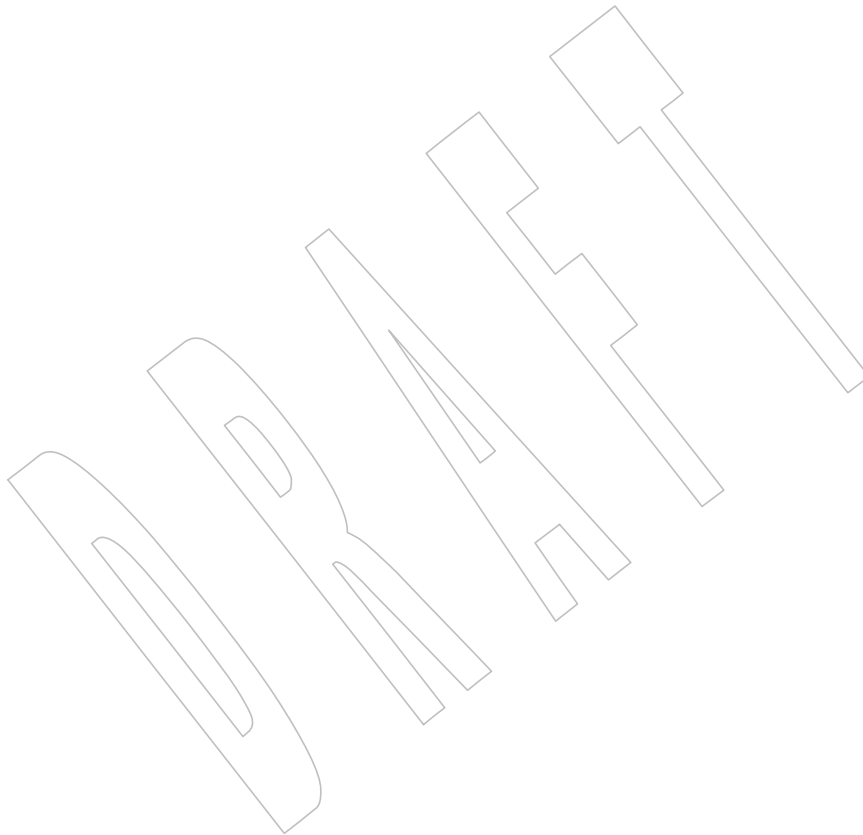
56800 **Issue 6**
 56801 The *sem_close()* function is marked as part of the Semaphores option.
 56802 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56803 implementation does not support the Semaphores option.
 56804 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/113 is applied, updating the ERRORS
 56805 section so that the [EINVAL] error becomes optional.

56806

Issue 7

56807

The *sem_close()* function is moved from the Semaphores option to the Base.



56808 **NAME**
 56809 `sem_destroy` — destroy an unnamed semaphore

56810 **SYNOPSIS**
 56811 `#include <semaphore.h>`
 56812 `int sem_destroy(sem_t *sem);`

56813 **DESCRIPTION**
 56814 The `sem_destroy()` function shall destroy the unnamed semaphore indicated by `sem`. Only a
 56815 semaphore that was created using `sem_init()` may be destroyed using `sem_destroy()`; the effect of
 56816 calling `sem_destroy()` with a named semaphore is undefined. The effect of subsequent use of the
 56817 semaphore `sem` is undefined until `sem` is reinitialized by another call to `sem_init()`.

56818 It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The
 56819 effect of destroying a semaphore upon which other threads are currently blocked is undefined.

56820 **RETURN VALUE**
 56821 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
 56822 returned and `errno` set to indicate the error.

56823 **ERRORS**
 56824 The `sem_destroy()` function may fail if:
 56825 [EINVAL] The `sem` argument is not a valid semaphore.
 56826 [EBUSY] There are currently processes blocked on the semaphore.

56827 **EXAMPLES**
 56828 None.

56829 **APPLICATION USAGE**
 56830 The `sem_destroy()` function is part of the Semaphores option and need not be available on all
 56831 implementations.

56832 **RATIONALE**
 56833 None.

56834 **FUTURE DIRECTIONS**
 56835 None.

56836 **SEE ALSO**
 56837 [*semctl\(\)*](#), [*semget\(\)*](#), [*semop\(\)*](#), [*sem_init\(\)*](#), [*sem_open\(\)*](#)
 56838 XBD [**<semaphore.h>**](#)

56839 **CHANGE HISTORY**
 56840 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

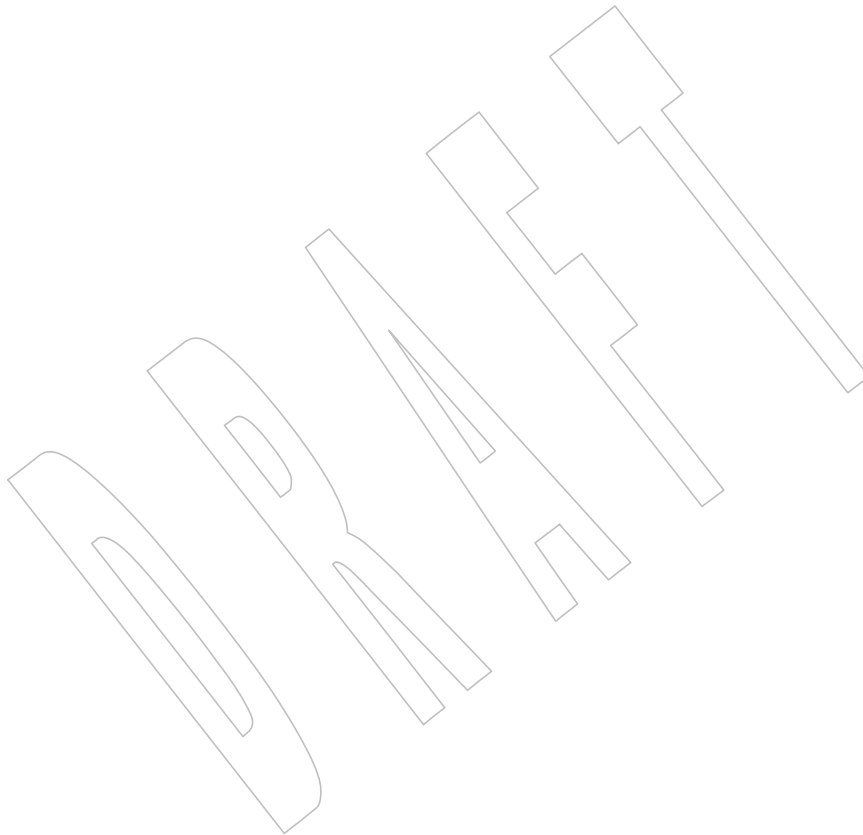
56841 **Issue 6**
 56842 The `sem_destroy()` function is marked as part of the Semaphores option.
 56843 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56844 implementation does not support the Semaphores option.
 56845 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/114 is applied, updating the ERRORS
 56846 section so that the [EINVAL] error becomes optional.

56847

Issue 7

56848

The *sem_destroy()* function is moved from the Semaphores option to the Base.



56849 **NAME**56850 `sem_getvalue` — get the value of a semaphore56851 **SYNOPSIS**56852 `#include <semaphore.h>`56853 `int sem_getvalue(sem_t *restrict sem, int *restrict sval);`56854 **DESCRIPTION**

56855 The `sem_getvalue()` function shall update the location referenced by the `sval` argument to have
 56856 the value of the semaphore referenced by `sem` without affecting the state of the semaphore. The
 56857 updated value represents an actual semaphore value that occurred at some unspecified time
 56858 during the call, but it need not be the actual value of the semaphore when it is returned to the
 56859 calling process.

56860 If `sem` is locked, then the object to which `sval` points shall either be set to zero or to a negative
 56861 number whose absolute value represents the number of processes waiting for the semaphore at
 56862 some unspecified time during the call.

56863 **RETURN VALUE**

56864 Upon successful completion, the `sem_getvalue()` function shall return a value of zero. Otherwise,
 56865 it shall return a value of `-1` and set `errno` to indicate the error.

56866 **ERRORS**56867 The `sem_getvalue()` function may fail if:56868 [EINVAL] The `sem` argument does not refer to a valid semaphore.56869 **EXAMPLES**

56870 None.

56871 **APPLICATION USAGE**

56872 The `sem_getvalue()` function is part of the Semaphores option and need not be available on all
 56873 implementations.

56874 **RATIONALE**

56875 None.

56876 **FUTURE DIRECTIONS**

56877 None.

56878 **SEE ALSO**56879 [`semctl\(\)`](#), [`semget\(\)`](#), [`semop\(\)`](#), [`sem_post\(\)`](#), [`sem_timedwait\(\)`](#), [`sem_trywait\(\)`](#)56880 XBD [`<semaphore.h>`](#)56881 **CHANGE HISTORY**

56882 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

56883 **Issue 6**56884 The `sem_getvalue()` function is marked as part of the Semaphores option.

56885 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 56886 implementation does not support the Semaphores option.

56887 The `sem_timedwait()` function is added to the SEE ALSO section for alignment with IEEE Std
 56888 1003.1d-1999.

56889 The **restrict** keyword is added to the `sem_getvalue()` prototype for alignment with the
 56890 ISO/IEC 9899:1999 standard.

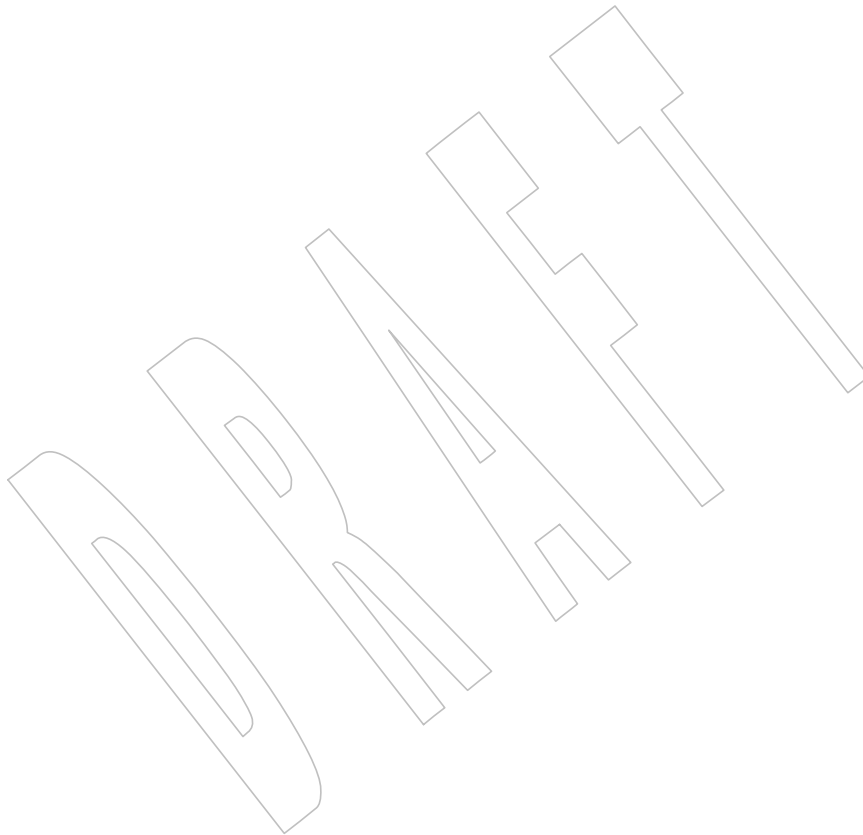
sem_getvalue()

56891 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/54 is applied.

56892 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/115 is applied, updating the ERRORS
56893 section so that the [EINVAL] error becomes optional.

Issue 7

56894 The *sem_getvalue()* function is moved from the Semaphores option to the Base.
56895



56896 **NAME**

56897 sem_init — initialize an unnamed semaphore

56898 **SYNOPSIS**

56899 #include <semaphore.h>

56900 int sem_init(sem_t *sem, int pshared, unsigned value);

56901 **DESCRIPTION**

56902 The *sem_init()* function shall initialize the unnamed semaphore referred to by *sem*. The value of
 56903 the initialized semaphore shall be *value*. Following a successful call to *sem_init()*, the semaphore
 56904 may be used in subsequent calls to *sem_wait()*, *sem_timedwait()*, *sem_trywait()*, *sem_post()*, and
 56905 *sem_destroy()*. This semaphore shall remain usable until the semaphore is destroyed.

56906 If the *pshared* argument has a non-zero value, then the semaphore is shared between processes;
 56907 in this case, any process that can access the semaphore *sem* can use *sem* for performing
 56908 *sem_wait()*, *sem_timedwait()*, *sem_trywait()*, *sem_post()*, and *sem_destroy()* operations.

56909 Only *sem* itself may be used for performing synchronization. The result of referring to copies of
 56910 *sem* in calls to *sem_wait()*, *sem_timedwait()*, *sem_trywait()*, *sem_post()*, and *sem_destroy()* is
 56911 undefined.

56912 If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any
 56913 thread in this process can use *sem* for performing *sem_wait()*, *sem_timedwait()*, *sem_trywait()*,
 56914 *sem_post()*, and *sem_destroy()* operations. The use of the semaphore by threads other than those
 56915 created in the same process is undefined.

56916 Attempting to initialize an already initialized semaphore results in undefined behavior.

56917 **RETURN VALUE**

56918 Upon successful completion, the *sem_init()* function shall initialize the semaphore in *sem* and
 56919 return 0. Otherwise, it shall return -1 and set *errno* to indicate the error.

56920 **ERRORS**

56921 The *sem_init()* function shall fail if:

56922 [EINVAL] The *value* argument exceeds {SEM_VALUE_MAX}.

56923 [ENOSPC] A resource required to initialize the semaphore has been exhausted, or the
 56924 limit on semaphores ({SEM_NSEMS_MAX}) has been reached.

56925 [EPERM] The process lacks the appropriate privileges to initialize the semaphore.

56926 **EXAMPLES**

56927 None.

56928 **APPLICATION USAGE**

56929 The *sem_init()* function is part of the Semaphores option and need not be available on all
 56930 implementations.

56931 **RATIONALE**

56932 None.

56933 **FUTURE DIRECTIONS**

56934 None.

56935
56936
56937
56938
56939
56940
56941
56942
56943
56944
56945
56946
56947
56948
56949
56950**SEE ALSO***sem_destroy()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*XBD <[semaphore.h](#)>**CHANGE HISTORY**

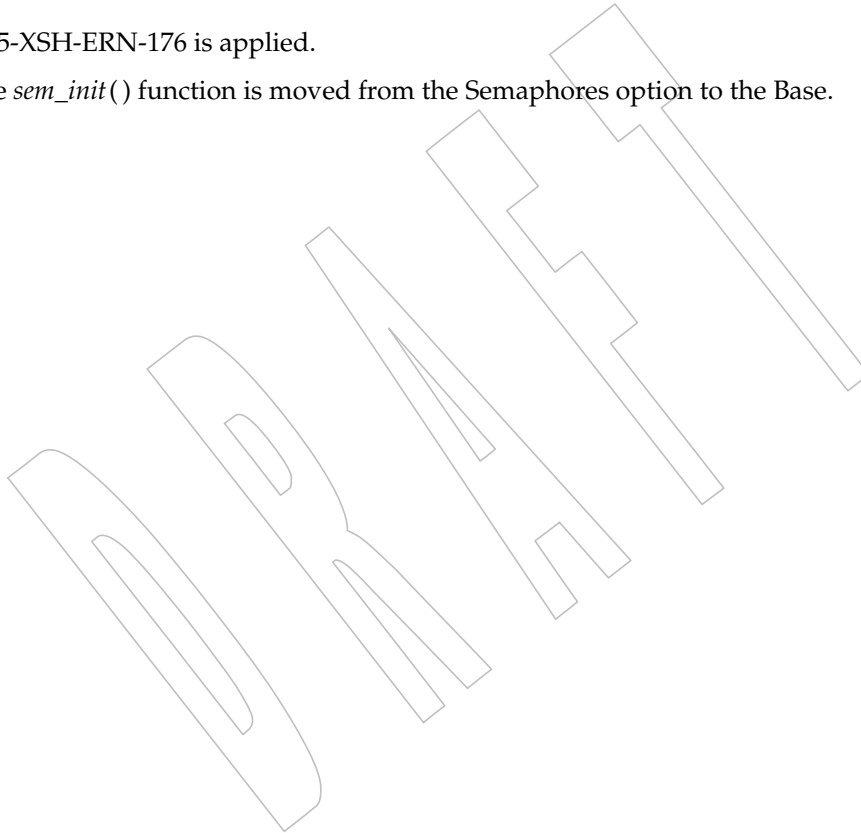
First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6The *sem_init()* function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/116 is applied, updating the DESCRIPTION to add the *sem_timedwait()* function for alignment with IEEE Std 1003.1d-1999.**Issue 7**

SD5-XSH-ERN-176 is applied.

The *sem_init()* function is moved from the Semaphores option to the Base.

56951 NAME

56952 sem_open — initialize and open a named semaphore

56953 SYNOPSIS

56954 #include <semaphore.h>

56955 sem_t *sem_open(const char *name, int oflag, ...);

56956 DESCRIPTION

56957 The *sem_open()* function shall establish a connection between a named semaphore and a process.
 56958 Following a call to *sem_open()* with semaphore name *name*, the process may reference the
 56959 semaphore associated with *name* using the address returned from the call. This semaphore may
 56960 be used in subsequent calls to *sem_wait()*, *sem_timedwait()*, *sem_trywait()*, *sem_post()*, and
 56961 *sem_close()*. The semaphore remains usable by this process until the semaphore is closed by a
 56962 successful call to *sem_close()*, *_exit()*, or one of the *exec* functions.

56963 The *oflag* argument controls whether the semaphore is created or merely accessed by the call to
 56964 *sem_open()*. The following flag bits may be set in *oflag*:

56965 **O_CREAT** This flag is used to create a semaphore if it does not already exist. If **O_CREAT** is
 56966 set and the semaphore already exists, then **O_CREAT** has no effect, except as noted
 56967 under **O_EXCL**. Otherwise, *sem_open()* creates a named semaphore. The **O_CREAT**
 56968 flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and
 56969 *value*, which is of type **unsigned**. The semaphore is created with an initial value of
 56970 *value*. Valid initial values for semaphores are less than or equal to
 56971 {SEM_VALUE_MAX}.

56972 The user ID of the semaphore shall be set to the effective user ID of the process. |
 56973 The group ID of the semaphore shall be set to the effective group ID of the process; |
 56974 however, if the *name* argument is visible in the file system, the group ID may be set |
 56975 to the group ID of the containing directory. The permission bits of the semaphore |
 56976 are set to the value of the *mode* argument except those set in the file mode creation |
 56977 mask of the process. When bits in *mode* other than the file permission bits are |
 56978 specified, the effect is unspecified.

56979 After the semaphore named *name* has been created by *sem_open()* with the
 56980 **O_CREAT** flag, other processes can connect to the semaphore by calling
 56981 *sem_open()* with the same value of *name*.

56982 **O_EXCL** If **O_EXCL** and **O_CREAT** are set, *sem_open()* fails if the semaphore *name* exists.
 56983 The check for the existence of the semaphore and the creation of the semaphore if it
 56984 does not exist are atomic with respect to other processes executing *sem_open()* with
 56985 **O_EXCL** and **O_CREAT** set. If **O_EXCL** is set and **O_CREAT** is not set, the effect is
 56986 undefined.

56987 If flags other than **O_CREAT** and **O_EXCL** are specified in the *oflag* parameter, the
 56988 effect is unspecified.

56989 The *name* argument points to a string naming a semaphore object. It is unspecified whether the
 56990 name appears in the file system and is visible to functions that take pathnames as arguments.
 56991 The *name* argument conforms to the construction rules for a pathname, except that the
 56992 interpretation of slash characters other than the leading slash character in *name* is
 56993 implementation-defined, and that the length limits for the *name* argument are implementation-
 56994 defined and need not be the same as the path name limits {PATH_MAX} and {NAME_MAX}. If
 56995 *name* begins with the slash character, then processes calling *sem_open()* with the same value of
 56996 *name* shall refer to the same semaphore object, as long as that name has not been removed. If
 56997 *name* does not begin with the slash character, the effect is implementation-defined.

sem_open()

56998 If a process makes multiple successful calls to *sem_open()* with the same value for *name*, the same
 56999 semaphore address shall be returned for each such successful call, provided that there have been
 57000 no calls to *sem_unlink()* for this semaphore, and at least one previous successful *sem_open()* call
 57001 for this semaphore has not been matched with a *sem_close()* call.

57002 References to copies of the semaphore produce undefined results.

RETURN VALUE

57003 Upon successful completion, the *sem_open()* function shall return the address of the semaphore.
 57004 Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol
 57005 SEM_FAILED is defined in the **<semaphore.h>** header. No successful return from *sem_open()*
 57006 shall return the value SEM_FAILED.
 57007

ERRORS

57008 If any of the following conditions occur, the *sem_open()* function shall return SEM_FAILED and
 57009 set *errno* to the corresponding value:
 57010

57011 [EACCES] The named semaphore exists and the permissions specified by *oflag* are
 57012 denied, or the named semaphore does not exist and permission to create the
 57013 named semaphore is denied.

57014 [EEXIST] O_CREAT and O_EXCL are set and the named semaphore already exists.

57015 [EINTR] The *sem_open()* operation was interrupted by a signal.

57016 [EINVAL] The *sem_open()* operation is not supported for the given name, or O_CREAT
 57017 was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

57018 [EMFILE] Too many semaphore descriptors or file descriptors are currently in use by this
 57019 process.

57020 [ENFILE] Too many semaphores are currently open in the system.

57021 [ENOENT] O_CREAT is not set and the named semaphore does not exist.

57022 [ENOMEM] There is insufficient memory for the creation of the new named semaphore.

57023 [ENOSPC] There is insufficient space on a storage device for the creation of the new
 57024 named semaphore.

57025 If any of the following conditions occur, the *sem_open()* function may return SEM_FAILED and
 57026 set *errno* to the corresponding value:

57027 [ENAMETOOLONG]
 57028 The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems
 57029 XSI that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI
 57030 systems, or has a pathname component that is longer than
 57031 XSI {_POSIX_NAME_MAX} on systems that do not support the XSI option or
 57032 longer than {_XOPEN_NAME_MAX} on XSI systems.

EXAMPLES

57033 None.
 57034

APPLICATION USAGE

57035 The *sem_open()* function is part of the Semaphores option and need not be available on all
 57036 implementations.
 57037

RATIONALE

57038 Early drafts required an error return value of -1 with the type **sem_t *** for the *sem_open()*
 57039 function, which is not guaranteed to be portable across implementations. The revised text
 57040 provides the symbolic error code SEM_FAILED to eliminate the type conflict.
 57041

57042 **FUTURE DIRECTIONS**

57043 None.

57044 **SEE ALSO**57045 *semctl()*, *semget()*, *semop()*, *sem_close()*, *sem_post()*, *sem_timedwait()*, *sem_trywait()*, *sem_unlink()*57046 XBD <**semaphore.h**> +57047 **CHANGE HISTORY**

57048 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57049 **Issue 6**57050 The *sem_open()* function is marked as part of the Semaphores option.57051 The [ENOSYS] error condition has been removed as stubs need not be provided if an
57052 implementation does not support the Semaphores option.57053 The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std
57054 1003.1d-1999.57055 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/117 is applied, updating the
57056 DESCRIPTION to add the *sem_timedwait()* function for alignment with IEEE Std 1003.1d-1999.57057 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/118 is applied, updating the
57058 DESCRIPTION to describe the conditions to return the same semaphore address on a call to
57059 *sem_open()*. The words “and at least one previous successful *sem_open()* call for this semaphore
57060 has not been matched with a *sem_close()* call” are added.57061 **Issue 7**57062 Austin Group Interpretation 1003.1-2001 #066 is applied, updating the [ENOSPC] error case and
57063 adding the [ENOMEM] error case.57064 Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and
57065 adding [ENAMETOOLONG] as a “may fail” error.57066 SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the +
57067 user ID and group ID of the semaphore. +57068 The *sem_open()* function is moved from the Semaphores option to the Base.

57069 **NAME**57070 `sem_post` — unlock a semaphore57071 **SYNOPSIS**57072 `#include <semaphore.h>`57073 `int sem_post(sem_t *sem);`57074 **DESCRIPTION**57075 The `sem_post()` function shall unlock the semaphore referenced by `sem` by performing a
57076 semaphore unlock operation on that semaphore.57077 If the semaphore value resulting from this operation is positive, then no threads were blocked
57078 waiting for the semaphore to become unlocked; the semaphore value is simply incremented.57079 If the value of the semaphore resulting from this operation is zero, then one of the threads
57080 blocked waiting for the semaphore shall be allowed to return successfully from its call to
57081 PS `sem_wait()`. If the Process Scheduling option is supported, the thread to be unblocked shall be
57082 chosen in a manner appropriate to the scheduling policies and parameters in effect for the
57083 blocked threads. In the case of the schedulers `SCHED_FIFO` and `SCHED_RR`, the highest
57084 priority waiting thread shall be unblocked, and if there is more than one highest priority thread
57085 blocked waiting for the semaphore, then the highest priority thread that has been waiting the
57086 longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread
57087 to unblock is unspecified.57088 SS If the Process Sporadic Server option is supported, and the scheduling policy is
57089 `SCHED_SPORADIC`, the semantics are as per `SCHED_FIFO` above.57090 The `sem_post()` function shall be reentrant with respect to signals and may be invoked from a
57091 signal-catching function.57092 **RETURN VALUE**57093 If successful, the `sem_post()` function shall return zero; otherwise, the function shall return `-1`
57094 and set `errno` to indicate the error.57095 **ERRORS**57096 The `sem_post()` function may fail if:57097 [EINVAL] The `sem` argument does not refer to a valid semaphore.57098 **EXAMPLES**

57099 None.

57100 **APPLICATION USAGE**57101 The `sem_post()` function is part of the Semaphores option and need not be available on all
57102 implementations.57103 **RATIONALE**

57104 None.

57105 **FUTURE DIRECTIONS**

57106 None.

57107 **SEE ALSO**57108 [semctl\(\)](#), [semget\(\)](#), [semop\(\)](#), [sem_timedwait\(\)](#), [sem_trywait\(\)](#)57109 XBD Section 4.11 (on page 98), [<semaphore.h>](#)

57110
57111
57112
57113
57114
57115
57116
57117
57118
57119
57120
57121
57122
57123

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *sem_post()* function is marked as part of the Semaphores option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

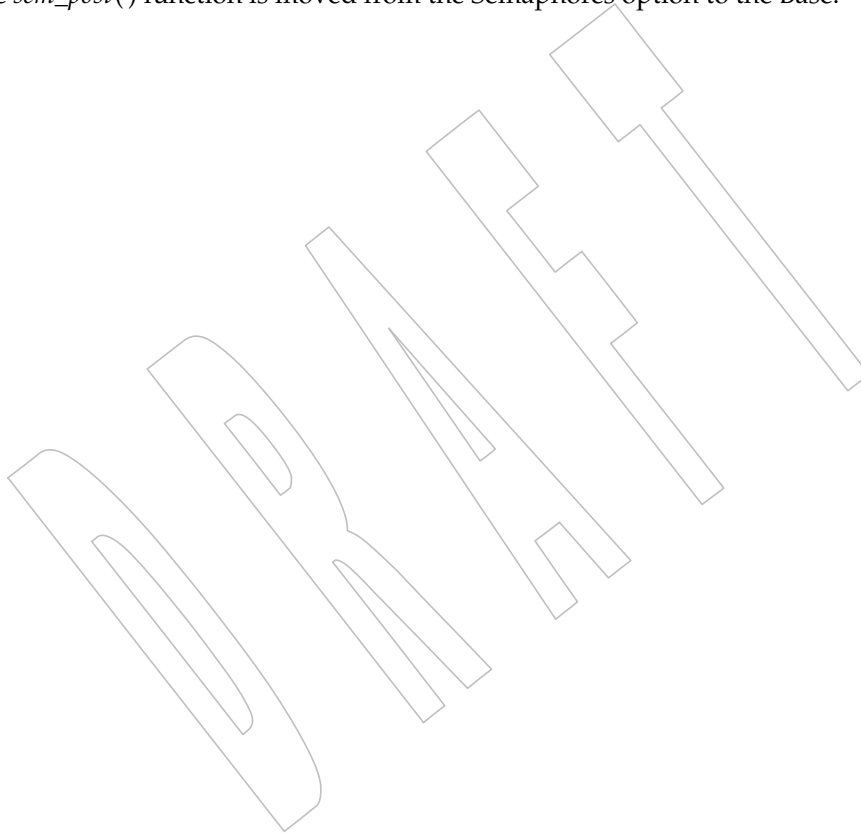
The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.

SCHED_SPORADIC is added to the list of scheduling policies for which the thread that is to be unblocked is specified for alignment with IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/119 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_post()* function is moved from the Semaphores option to the Base.



57124 **NAME**
 57125 `sem_timedwait` — lock a semaphore

57126 **SYNOPSIS**
 57127 `#include <semaphore.h>`
 57128 `#include <time.h>`
 57129 `int sem_timedwait(sem_t *restrict sem,`
 57130 `const struct timespec *restrict abstime);`

57131 **DESCRIPTION**
 57132 The `sem_timedwait()` function shall lock the semaphore referenced by `sem` as in the `sem_wait()`
 57133 function. However, if the semaphore cannot be locked without waiting for another process or
 57134 thread to unlock the semaphore by performing a `sem_post()` function, this wait shall be
 57135 terminated when the specified timeout expires.

57136 The timeout shall expire when the absolute time specified by `abstime` passes, as measured by the
 57137 clock on which timeouts are based (that is, when the value of that clock equals or exceeds
 57138 `abstime`), or if the absolute time specified by `abstime` has already been passed at the time of the
 57139 call.

57140 The timeout shall be based on the `CLOCK_REALTIME` clock. The resolution of the timeout shall
 57141 be the resolution of the clock on which it is based. The `timespec` data type is defined as a
 57142 structure in the `<time.h>` header.

57143 Under no circumstance shall the function fail with a timeout if the semaphore can be locked
 57144 immediately. The validity of the `abstime` need not be checked if the semaphore can be locked
 57145 immediately.

57146 **RETURN VALUE**
 57147 The `sem_timedwait()` function shall return zero if the calling process successfully performed the
 57148 semaphore lock operation on the semaphore designated by `sem`. If the call was unsuccessful, the
 57149 state of the semaphore shall be unchanged, and the function shall return a value of `-1` and set
 57150 `errno` to indicate the error.

57151 **ERRORS**
 57152 The `sem_timedwait()` function shall fail if:
 57153 [EINVAL] The process or thread would have blocked, and the `abstime` parameter
 57154 specified a nanoseconds field value less than zero or greater than or equal to
 57155 1 000 million.

57156 [ETIMEDOUT] The semaphore could not be locked before the specified timeout expired.

57157 The `sem_timedwait()` function may fail if:

57158 [EDEADLK] A deadlock condition was detected.

57159 [EINTR] A signal interrupted this function.

57160 [EINVAL] The `sem` argument does not refer to a valid semaphore.

57161
57162
57163
57164
57165
57166
57167
57168
57169
57170
57171
57172
57173
57174
57175
57176
57177
57178
57179**EXAMPLES**

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in XBD Section 3.284 (on page 72).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_post(), *sem_trywait()*, *semctl()*, *semget()*, *semop()*, *time*

XBD Section 3.284 (on page 72), **<semaphore.h>**, **<time.h>**

CHANGE HISTORY

First released in Issue 6. Derived from IEEE Std 1003.1d-1999.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/120 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

Issue 7

The *sem_timedwait()* function is moved from the Semaphores option to the Base.

Functionality relating to the Timers option is moved to the Base.

57180 **NAME**
 57181 `sem_trywait`, `sem_wait` — lock a semaphore

57182 **SYNOPSIS**
 57183 `#include <semaphore.h>`
 57184 `int sem_trywait(sem_t *sem);`
 57185 `int sem_wait(sem_t *sem);`

57186 **DESCRIPTION**
 57187 The `sem_trywait()` function shall lock the semaphore referenced by `sem` only if the semaphore is
 57188 currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not
 57189 lock the semaphore.

57190 The `sem_wait()` function shall lock the semaphore referenced by `sem` by performing a semaphore
 57191 lock operation on that semaphore. If the semaphore value is currently zero, then the calling
 57192 thread shall not return from the call to `sem_wait()` until it either locks the semaphore or the call is
 57193 interrupted by a signal.

57194 Upon successful return, the state of the semaphore shall be locked and shall remain locked until
 57195 the `sem_post()` function is executed and returns successfully.

57196 The `sem_wait()` function is interruptible by the delivery of a signal.

57197 **RETURN VALUE**
 57198 The `sem_trywait()` and `sem_wait()` functions shall return zero if the calling process successfully
 57199 performed the semaphore lock operation on the semaphore designated by `sem`. If the call was
 57200 unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a
 57201 value of `-1` and set `errno` to indicate the error.

57202 **ERRORS**
 57203 The `sem_trywait()` function shall fail if:
 57204 [EAGAIN] The semaphore was already locked, so it cannot be immediately locked by the
 57205 `sem_trywait()` operation.

57206 The `sem_trywait()` and `sem_wait()` functions may fail if:
 57207 [EDEADLK] A deadlock condition was detected.
 57208 [EINTR] A signal interrupted this function.
 57209 [EINVAL] The `sem` argument does not refer to a valid semaphore.

57210 **EXAMPLES**
 57211 None.

57212 **APPLICATION USAGE**
 57213 Applications using these functions may be subject to priority inversion, as discussed in XBD |
 57214 [Section 3.284](#) (on page 72).

57215 The `sem_trywait()` and `sem_wait()` functions are part of the Semaphores option and need not be
 57216 provided on all implementations.

57217 **RATIONALE**
 57218 None.

57219
57220**FUTURE DIRECTIONS**

None.

57221
57222**SEE ALSO***semctl()*, *semget()*, *semop()*, *sem_post()*, *sem_timedwait()*

57223

XBD Section 3.284 (on page 72), Section 4.11 (on page 98), <semaphore.h>

57224
57225**CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57226
57227**Issue 6**The *sem_trywait()* and *sem_wait()* functions are marked as part of the Semaphores option.57228
57229

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

57230
57231The *sem_timedwait()* function is added to the SEE ALSO section for alignment with IEEE Std 1003.1d-1999.57232
57233

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/121 is applied, updating the ERRORS section so that the [EINVAL] error becomes optional.

57234
57235**Issue 7**SD5-XSH-ERN-54 is applied, removing the *sem_wait()* function from the “shall fail” error cases.

57236

The *sem_trywait()* and *sem_wait()* functions are moved from the Semaphores option to the Base.

DRAFT

57237 **NAME**57238 `sem_unlink` — remove a named semaphore57239 **SYNOPSIS**57240 `#include <semaphore.h>`57241 `int sem_unlink(const char *name);`57242 **DESCRIPTION**

57243 The `sem_unlink()` function shall remove the semaphore named by the string *name*. If the
 57244 semaphore named by *name* is currently referenced by other processes, then `sem_unlink()` shall
 57245 have no effect on the state of the semaphore. If one or more processes have the semaphore open
 57246 when `sem_unlink()` is called, destruction of the semaphore is postponed until all references to the
 57247 semaphore have been destroyed by calls to `sem_close()`, `_exit()`, or `exec`. Calls to `sem_open()`
 57248 to recreate or reconnect to the semaphore refer to a new semaphore after `sem_unlink()` is called. The
 57249 `sem_unlink()` call shall not block until all references have been destroyed; it shall return
 57250 immediately.

57251 **RETURN VALUE**

57252 Upon successful completion, the `sem_unlink()` function shall return a value of 0. Otherwise, the
 57253 semaphore shall not be changed and the function shall return a value of -1 and set *errno* to
 57254 indicate the error.

57255 **ERRORS**57256 The `sem_unlink()` function shall fail if:

57257 [EACCES] Permission is denied to unlink the named semaphore.

57258 [ENOENT] The named semaphore does not exist.

57259 The `sem_unlink()` function may fail if:

57260 [ENAMETOOLONG]

57261 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
 57262 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 57263 systems, or has a pathname component that is longer than
 57264 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or
 57265 longer than `{_XOPEN_NAME_MAX}` on XSI systems. A call to `sem_unlink()`
 57266 with a *name* argument that contains the same semaphore name as was
 57267 previously used in a successful `sem_open()` call shall not give an
 57268 [ENAMETOOLONG] error.

57269 **EXAMPLES**

57270 None.

57271 **APPLICATION USAGE**

57272 The `sem_unlink()` function is part of the Semaphores option and need not be available on all
 57273 implementations.

57274 **RATIONALE**

57275 None.

57276 **FUTURE DIRECTIONS**

57277 None.

57278

SEE ALSO

57279

semctl(), *semget()*, *semop()*, *sem_close()*, *sem_open()*

57280

XBD <[semaphore.h](#)>

57281

CHANGE HISTORY

57282

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

57283

Issue 6

57284

The *sem_unlink()* function is marked as part of the Semaphores option.

57285

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Semaphores option.

57286

57287

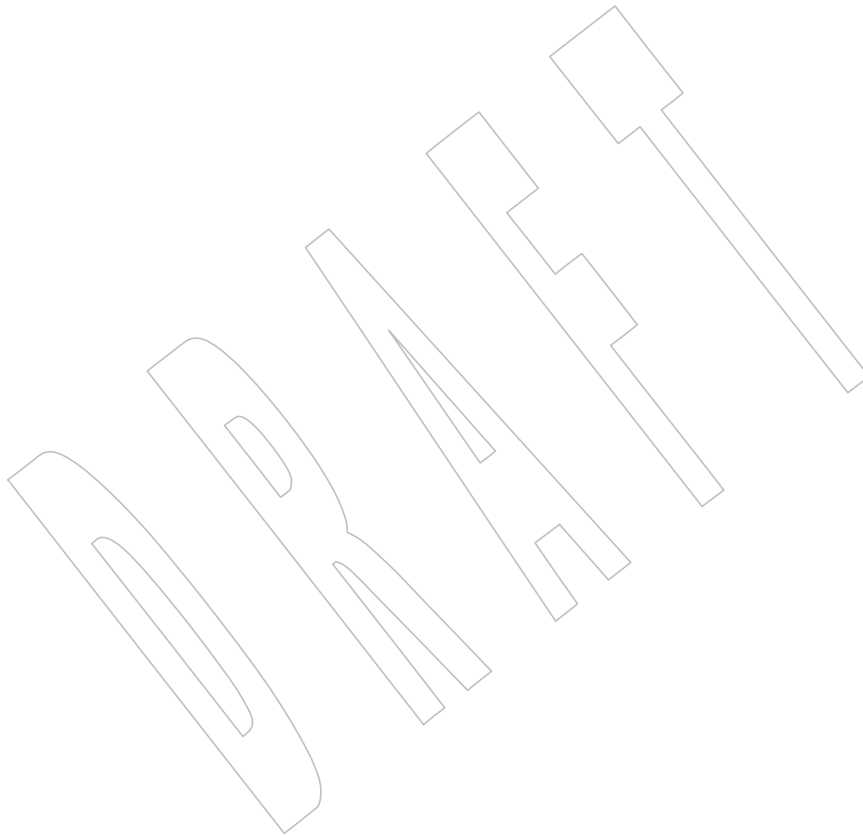
Issue 7

57288

Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

57289

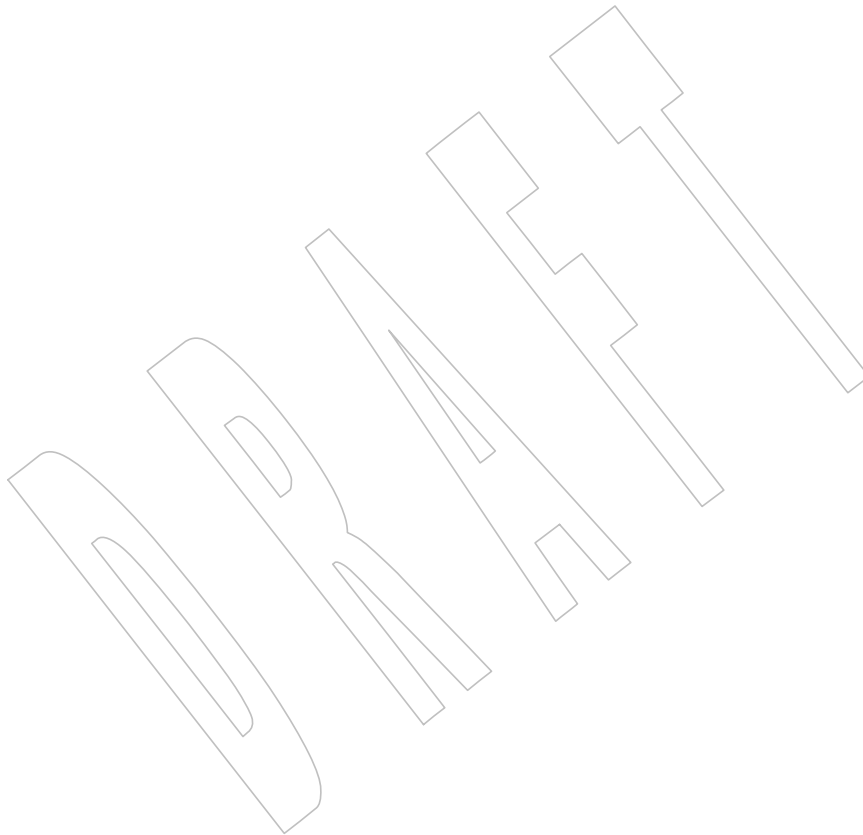
57290

The *sem_unlink()* function is moved from the Semaphores option to the Base.

57291 **NAME**
57292 `sem_wait` — lock a semaphore

57293 **SYNOPSIS**
57294 `#include <semaphore.h>`
57295 `int sem_wait(sem_t *sem);`

57296 **DESCRIPTION**
57297 Refer to *sem_trywait()*.



57298 **NAME**
 57299 semctl — XSI semaphore control operations

57300 **SYNOPSIS**

```
57301 XSI #include <sys/sem.h>
57302 int semctl(int semid, int semnum, int cmd, ...);
```

57303 **DESCRIPTION**

57304 The *semctl()* function operates on XSI semaphores (see XBD [Section 4.16](#), on page 101). It is
 57305 unspecified whether this function interoperates with the realtime interprocess communication
 57306 facilities defined in [Section 2.8](#) (on page 476).

57307 The *semctl()* function provides a variety of semaphore control operations as specified by *cmd*.
 57308 The fourth argument is optional and depends upon the operation requested. If required, it is of
 57309 type **union semun**, which the application shall explicitly declare:

```
57310 union semun {
57311     int val;
57312     struct semid_ds *buf;
57313     unsigned short *array;
57314 } arg;
```

57315 The following semaphore control operations as specified by *cmd* are executed with respect to the
 57316 semaphore specified by *semid* and *semnum*. The level of permission required for each operation
 57317 is shown with each command; see [Section 2.7](#) (on page 474). The symbolic names for the values
 57318 of *cmd* are defined in the `<sys/sem.h>` header:

57319	GETVAL	Return the value of <i>semval</i> ; see <code><sys/sem.h></code> . Requires read permission.
57320	SETVAL	Set the value of <i>semval</i> to <i>arg.val</i> , where <i>arg</i> is the value of the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> value corresponding to the specified semaphore in all processes is cleared. Requires alter permission; see Section 2.7 (on page 474).
57321		
57322		
57323		
57324	GETPID	Return the value of <i>sempid</i> . Requires read permission.
57325	GETNCNT	Return the value of <i>semmcnt</i> . Requires read permission.
57326	GETZCNT	Return the value of <i>semzcnt</i> . Requires read permission.

57327 The following values of *cmd* operate on each *semval* in the set of semaphores:

57328	GETALL	Return the value of <i>semval</i> for each semaphore in the semaphore set and place into the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . Requires read permission.
57329		
57330		
57331	SETALL	Set the value of <i>semval</i> for each semaphore in the semaphore set according to the array pointed to by <i>arg.array</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . When this command is successfully executed, the <i>semadj</i> values corresponding to each specified semaphore in all processes are cleared. Requires alter permission.
57332		
57333		
57334		
57335		

57336 The following values of *cmd* are also available:

57337	IPC_STAT	Place the current value of each member of the semid_ds data structure associated with <i>semid</i> into the structure pointed to by <i>arg.buf</i> , where <i>arg</i> is the fourth argument to <i>semctl()</i> . The contents of this structure are defined in <code><sys/sem.h></code> . Requires read permission.
57338		
57339		
57340		

semctl()

57341 57342 57343 57344 57345 57346 57347 57348 57349 57350 57351 57352 57353 57354 57355 57356 57357 57358 57359 57360 57361 57362 57363 57364 57365 57366 57367 57368 57369 57370 57371 57372 57373 57374 57375 57376 57377 57378 57379 57380	<p>IPC_SET</p> <p>Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i>.</p> <p>Remove the semaphore identifier specified by <i>semid</i> from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i>.</p> <p>RETURN VALUE</p> <p>If successful, the value returned by <i>semctl()</i> depends on <i>cmd</i> as follows:</p> <p>GETVAL The value of <i>semval</i>.</p> <p>GETPID The value of <i>sempid</i>.</p> <p>GETNCNT The value of <i>semincnt</i>.</p> <p>GETZCNT The value of <i>semzcnt</i>.</p> <p>All others 0.</p> <p>Otherwise, <i>semctl()</i> shall return -1 and set <i>errno</i> to indicate the error.</p> <p>ERRORS</p> <p>The <i>semctl()</i> function shall fail if:</p> <p>[EACCES] Operation permission is denied to the calling process; see Section 2.7 (on page 474).</p> <p>[EINVAL] The value of <i>semid</i> is not a valid semaphore identifier, or the value of <i>semnum</i> is less than 0 or greater than or equal to <i>sem_nsems</i>, or the value of <i>cmd</i> is not a valid command.</p> <p>[EPERM] The argument <i>cmd</i> is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the data structure associated with <i>semid</i>.</p> <p>[ERANGE] The argument <i>cmd</i> is equal to SETVAL or SETALL and the value to which <i>semval</i> is to be set is greater than the system-imposed maximum.</p>	<p>Set the value of the following members of the semid_ds data structure associated with <i>semid</i> to the corresponding value found in the structure pointed to by <i>arg.buf</i>, where <i>arg</i> is the fourth argument to <i>semctl()</i>:</p> <p><i>sem_perm.uid</i> <i>sem_perm.gid</i> <i>sem_perm.mode</i></p> <p>The mode bits specified in Section 2.7.1 (on page 475) are copied into the corresponding bits of the <i>sem_perm.mode</i> associated with <i>semid</i>. The stored values of any other bits are unspecified.</p> <p>This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of <i>sem_perm.cuid</i> or <i>sem_perm.uid</i> in the semid_ds data structure associated with <i>semid</i>.</p>
--	--	---

57381

EXAMPLES

57382

None.

57383

APPLICATION USAGE

57384

The fourth parameter in the SYNOPSIS section is now specified as ". . ." in order to avoid a clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for backwards-compatibility.

57385

57386

57387

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

57388

57389

57390

57391

RATIONALE

57392

None.

57393

FUTURE DIRECTIONS

57394

None.

57395

SEE ALSO

57396

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [semget\(\)](#), [semop\(\)](#), [sem_close\(\)](#), [sem_destroy\(\)](#), [sem_getvalue\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), [sem_post\(\)](#), [sem_trywait\(\)](#), [sem_unlink\(\)](#)

57397

57398

XBD [Section 4.16](#) (on page 101), [<sys/sem.h>](#)

+

57399

CHANGE HISTORY

57400

First released in Issue 2. Derived from Issue 2 of the SVID.

57401

Issue 5

57402

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to the APPLICATION USAGE section.

57403

57404 **NAME**

57405 semget — get set of XSI semaphores

57406 **SYNOPSIS**

```
57407 XSI #include <sys/sem.h>
57408 int semget(key_t key, int nsems, int semflg);
```

57409 **DESCRIPTION**

57410 The *semget()* function operates on XSI semaphores (see XBD [Section 4.16](#), on page 101). It is
 57411 unspecified whether this function interoperates with the realtime interprocess communication
 57412 facilities defined in [Section 2.8](#) (on page 476).

57413 The *semget()* function shall return the semaphore identifier associated with *key*.

57414 A semaphore identifier with its associated **semid_ds** data structure and its associated set of
 57415 *nsems* semaphores (see [<sys/sem.h>](#)) is created for *key* if one of the following is true:

- 57416 • The argument *key* is equal to `IPC_PRIVATE`.
- 57417 • The argument *key* does not already have a semaphore identifier associated with it and
 57418 (*semflg* & `IPC_CREAT`) is non-zero.

57419 Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is
 57420 initialized as follows:

- 57421 • In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and
 57422 *sem_perm.gid* shall be set equal to the effective user ID and effective group ID, respectively,
 57423 of the calling process.
- 57424 • The low-order 9 bits of *sem_perm.mode* shall be set equal to the low-order 9 bits of *semflg*.
- 57425 • The variable *sem_nsems* shall be set equal to the value of *nsems*.
- 57426 • The variable *sem_otime* shall be set equal to 0 and *sem_ctime* shall be set equal to the current
 57427 time.
- 57428 • The data structure associated with each semaphore in the set need not be initialized. The
 57429 *semctl()* function with the command `SETVAL` or `SETALL` can be used to initialize each
 57430 semaphore.

57431 **RETURN VALUE**

57432 Upon successful completion, *semget()* shall return a non-negative integer, namely a semaphore
 57433 identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

57434 **ERRORS**

57435 The *semget()* function shall fail if:

- | | | |
|-------|----------|---|
| 57436 | [EACCES] | A semaphore identifier exists for <i>key</i> , but operation permission as specified by the low-order 9 bits of <i>semflg</i> would not be granted; see Section 2.7 (on page 474). |
| 57437 | | |
| 57438 | | |
| 57439 | [EEXIST] | A semaphore identifier exists for the argument <i>key</i> but $((\textit{semflg} \ \&\textit{IPC_CREAT}) \ \&\&(\textit{semflg} \ \&\textit{IPC_EXCL}))$ is non-zero. |
| 57440 | | |
| 57441 | [EINVAL] | The value of <i>nsems</i> is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0. |
| 57442 | | |
| 57443 | | |
| 57444 | | |

- 57445 [ENOENT] A semaphore identifier does not exist for the argument *key* and (*semflg*
57446 &IPC_CREAT) is equal to 0.
- 57447 [ENOSPC] A semaphore identifier is to be created but the system-imposed limit on the
57448 maximum number of allowed semaphores system-wide would be exceeded.

57449 EXAMPLES

57450 Creating a Semaphore Identifier

57451 The following example gets a unique semaphore key using the *ftok()* function, then gets a
57452 semaphore ID associated with that key using the *semget()* function (the first call also tests to
57453 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
57454 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
57455 program attempts to create one semaphore with read/write permission for all. It also uses the
57456 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

57457 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
57458 the *sbuf* array. The number of processes that can execute concurrently without queuing is
57459 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
57460 the program.

```
57461 #include <sys/types.h>
57462 #include <stdio.h>
57463 #include <sys/ipc.h>
57464 #include <sys/sem.h>
57465 #include <sys/stat.h>
57466 #include <errno.h>
57467 #include <unistd.h>
57468 #include <stdlib.h>
57469 #include <pwd.h>
57470 #include <fcntl.h>
57471 #include <limits.h>
57472 ...
57473 key_t semkey;
57474 int semid, pfd, fv;
57475 struct sembuf sbuf;
57476 char *lgn;
57477 char filename[PATH_MAX+1];
57478 struct stat outstat;
57479 struct passwd *pw;
57480 ...
57481 /* Get unique key for semaphore. */
57482 if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
57483     perror("IPC error: ftok"); exit(1);
57484 }
57485
57486 /* Get semaphore ID associated with this key. */
57487 if ((semid = semget(semkey, 0, 0)) == -1) {
57488     /* Semaphore does not exist - Create. */
57489     if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
57490         S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
57491     {
57492         /* Initialize the semaphore. */
57493         sbuf.sem_num = 0;
57494         sbuf.sem_op = 2; /* This is the number of runs
                           without queuing. */

```

```

57495         sbuf.sem_flg = 0;
57496         if (semop(semid, &sbuf, 1) == -1) {
57497             perror("IPC error: semop"); exit(1);
57498         }
57499     }
57500     else if (errno == EEXIST) {
57501         if ((semid = semget(semkey, 0, 0)) == -1) {
57502             perror("IPC error 1: semget"); exit(1);
57503         }
57504     }
57505     else {
57506         perror("IPC error 2: semget"); exit(1);
57507     }
57508 }
57509 ...

```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [semctl\(\)](#), [semop\(\)](#), [sem_close\(\)](#), [sem_destroy\(\)](#), [sem_getvalue\(\)](#), [sem_init\(\)](#), [sem_open\(\)](#), [sem_post\(\)](#), [sem_trywait\(\)](#), [sem_unlink\(\)](#)

XBD [Section 4.16](#) (on page 101), [<sys/sem.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 6

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/122 is applied, updating the DESCRIPTION from “each semaphore in the set shall not be initialized” to “each semaphore in the set need not be initialized”.

57532 **NAME**
 57533 semop — XSI semaphore operations

57534 **SYNOPSIS**

```
57535 XSI #include <sys/sem.h>
57536 int semop(int semid, struct sembuf *sops, size_t nsops);
```

57537 **DESCRIPTION**

57538 The *semop()* function operates on XSI semaphores (see XBD Section 4.16, on page 101). It is
 57539 unspecified whether this function interoperates with the realtime interprocess communication
 57540 facilities defined in Section 2.8 (on page 476).

57541 The *semop()* function shall perform atomically a user-defined array of semaphore operations in
 57542 array order on the set of semaphores associated with the semaphore identifier specified by the
 57543 argument *semid*.

57544 The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The
 57545 implementation shall not modify elements of this array unless the application uses
 57546 implementation-defined extensions.

57547 The argument *nsops* is the number of such structures in the array.

57548 Each structure, **sembuf**, includes the following members:

Member Type	Member Name	Description
short	<i>sem_num</i>	Semaphore number.
short	<i>sem_op</i>	Semaphore operation.
short	<i>sem_flg</i>	Operation flags.

57549 Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore
 57550 specified by *semid* and *sem_num*.

57551 The variable *sem_op* specifies one of three semaphore operations:

- 57552 If *sem_op* is a negative integer and the calling process has alter permission, one of the
 57553 following shall occur:
 57554
 - 57555 • If *semval*(see <sys/sem.h>) is greater than or equal to the absolute value of *sem_op*,
 57556 the absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg*
 57557 &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the
 57558 *semadj* value of the calling process for the specified semaphore.
 - 57559 • If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is
 57560 non-zero, *semop()* shall return immediately.
 - 57561 • If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is 0,
 57562 *semop()* shall increment the *semncnt* associated with the specified semaphore and
 57563 suspend execution of the calling thread until one of the following conditions occurs:
 57564 — The value of *semval* becomes greater than or equal to the absolute value of
 57565 *sem_op*. When this occurs, the value of *semncnt* associated with the specified
 57566 semaphore shall be decremented, the absolute value of *sem_op* shall be
 57567 subtracted from *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the
 57568 absolute value of *sem_op* shall be added to the *semadj* value of the calling
 57569 process for the specified semaphore.
 57570
 57571
 57572

- 57573 — The *semid* for which the calling thread is awaiting action is removed from the
 57574 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be
 57575 returned.
- 57576 — The calling thread receives a signal that is to be caught. When this occurs, the
 57577 value of *semncnt* associated with the specified semaphore shall be
 57578 decremented, and the calling thread shall resume execution in the manner
 57579 prescribed in *sigaction()*.
- 57580 2. If *sem_op* is a positive integer and the calling process has alter permission, the value of
 57581 *sem_op* shall be added to *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the value of
 57582 *sem_op* shall be subtracted from the *semadj* value of the calling process for the specified
 57583 semaphore.
- 57584 3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:
- 57585 • If *semval* is 0, *semop()* shall return immediately.
 - 57586 • If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is non-zero, *semop()* shall return
 57587 immediately.
 - 57588 • If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is 0, *semop()* shall increment the
 57589 *semzcnt* associated with the specified semaphore and suspend execution of the
 57590 calling thread until one of the following occurs:
 - 57591 — The value of *semval* becomes 0, at which time the value of *semzcnt* associated
 57592 with the specified semaphore shall be decremented.
 - 57593 — The *semid* for which the calling thread is awaiting action is removed from the
 57594 system. When this occurs, *errno* shall be set equal to [EIDRM] and -1 shall be
 57595 returned.
 - 57596 — The calling thread receives a signal that is to be caught. When this occurs, the
 57597 value of *semzcnt* associated with the specified semaphore shall be
 57598 decremented, and the calling thread shall resume execution in the manner
 57599 prescribed in *sigaction()*.

57600 Upon successful completion, the value of *sempid* for each semaphore specified in the array
 57601 pointed to by *sops* shall be set equal to the process ID of the calling process.

57602 RETURN VALUE

57603 Upon successful completion, *semop()* shall return 0; otherwise, it shall return -1 and set *errno* to
 57604 indicate the error.

57605 ERRORS

57606 The *semop()* function shall fail if:

- | | | |
|-------|----------|--|
| 57607 | [E2BIG] | The value of <i>nsops</i> is greater than the system-imposed maximum. |
| 57608 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page
57609 474). |
| 57610 | [EAGAIN] | The operation would result in suspension of the calling process but (<i>sem_flg</i>
57611 &IPC_NOWAIT) is non-zero. |
| 57612 | [EFBIG] | The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of
57613 semaphores in the set associated with <i>semid</i> . |
| 57614 | [EIDRM] | The semaphore identifier <i>semid</i> is removed from the system. |
| 57615 | [EINTR] | The <i>semop()</i> function was interrupted by a signal. |

57616	[EINVAL]	The value of <i>semid</i> is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.
57617		
57618		
57619	[ENOSPC]	The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.
57620		
57621	[ERANGE]	An operation would cause a <i>semval</i> to overflow the system-imposed limit, or an operation would cause a <i>semadj</i> value to overflow the system-imposed limit.
57622		
57623		

57624 EXAMPLES

57625 Setting Values in Semaphores

57626 The following example sets the values of the two semaphores associated with the *semid* identifier
57627 to the values contained in the *sb* array.

```
57628 #include <sys/sem.h>
57629 ...
57630 int semid;
57631 struct sembuf sb[2];
57632 int nsops = 2;
57633 int result;
57634
57635 /* Adjust value of semaphore in the semaphore array semid. */
57636 sb[0].sem_num = 0;
57637 sb[0].sem_op = -1;
57638 sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
57639 sb[1].sem_num = 1;
57640 sb[1].sem_op = 1;
57641 sb[1].sem_flg = 0;
57642
57643 result = semop(semid, sb, nsops);
```

57642 Creating a Semaphore Identifier

57643 The following example gets a unique semaphore key using the *ftok()* function, then gets a
57644 semaphore ID associated with that key using the *semget()* function (the first call also tests to
57645 make sure the semaphore exists). If the semaphore does not exist, the program creates it, as
57646 shown by the second call to *semget()*. In creating the semaphore for the queuing process, the
57647 program attempts to create one semaphore with read/write permission for all. It also uses the
57648 IPC_EXCL flag, which forces *semget()* to fail if the semaphore already exists.

57649 After creating the semaphore, the program uses a call to *semop()* to initialize it to the values in
57650 the *sbuf* array. The number of processes that can execute concurrently without queuing is
57651 initially set to 2. The final call to *semget()* creates a semaphore identifier that can be used later in
57652 the program.

57653 The final call to *semop()* acquires the semaphore and waits until it is free; the SEM_UNDO
57654 option releases the semaphore when the process exits, waiting until there are less than two
57655 processes running concurrently.

```
57656 #include <sys/types.h>
57657 #include <stdio.h>
57658 #include <sys/ipc.h>
57659 #include <sys/sem.h>
57660 #include <sys/stat.h>
57661 #include <errno.h>
```

```

57662     #include <unistd.h>
57663     #include <stdlib.h>
57664     #include <pwd.h>
57665     #include <fcntl.h>
57666     #include <limits.h>
57667     ...
57668     key_t semkey;
57669     int semid, pfd, fv;
57670     struct sembuf sbuf;
57671     char *lgn;
57672     char filename[PATH_MAX+1];
57673     struct stat outstat;
57674     struct passwd *pw;
57675     ...
57676     /* Get unique key for semaphore. */
57677     if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
57678         perror("IPC error: ftok"); exit(1);
57679     }
57680     /* Get semaphore ID associated with this key. */
57681     if ((semid = semget(semkey, 0, 0)) == -1) {
57682         /* Semaphore does not exist - Create. */
57683         if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
57684             S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
57685             {
57686                 /* Initialize the semaphore. */
57687                 sbuf.sem_num = 0;
57688                 sbuf.sem_op = 2; /* This is the number of runs without queuing. */
57689                 sbuf.sem_flg = 0;
57690                 if (semop(semid, &sbuf, 1) == -1) {
57691                     perror("IPC error: semop"); exit(1);
57692                 }
57693             }
57694         else if (errno == EEXIST) {
57695             if ((semid = semget(semkey, 0, 0)) == -1) {
57696                 perror("IPC error 1: semget"); exit(1);
57697             }
57698         }
57699         else {
57700             perror("IPC error 2: semget"); exit(1);
57701         }
57702     }
57703     ...
57704     sbuf.sem_num = 0;
57705     sbuf.sem_op = -1;
57706     sbuf.sem_flg = SEM_UNDO;
57707     if (semop(semid, &sbuf, 1) == -1) {
57708         perror("IPC Error: semop"); exit(1);
57709     }

```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

57715
57716
57717
57718
57719
57720
57721
57722
57723
57724
57725
57726
57727
57728
57729
57730
57731

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7 (on page 474), Section 2.8 (on page 476), *exec*, *exit()*, *fork()*, *semctl()*, *semget()*, *sem_close()*, *sem_destroy()*, *sem_getvalue()*, *sem_init()*, *sem_open()*, *sem_post()*, *sem_trywait()*, *sem_unlink()*

XBD Section 4.16 (on page 101), `<sys/ipc.h>`, `<sys/sem.h>`, `<sys/types.h>`

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

Issue 7

SD5-XSH-ERN-171 is applied, updating the DESCRIPTION to clarify the order in which the operations in *sops* will be performed when there are multiple operations.

DRAFT

57732 **NAME**

57733 send — send a message on a socket

57734 **SYNOPSIS**

57735 #include <sys/socket.h>

57736 ssize_t send(int socket, const void *buffer, size_t length, int flags);

57737 **DESCRIPTION**

57738 The *send()* function shall initiate transmission of a message from the specified socket to its peer.
 57739 The *send()* function shall send a message only when the socket is connected. If the socket is a
 57740 connectionless-mode socket, the message shall be sent to the pre-specified peer address.

57741 The *send()* function takes the following arguments:

57742	<i>socket</i>	Specifies the socket file descriptor.
57743	<i>buffer</i>	Points to the buffer containing the message to send.
57744	<i>length</i>	Specifies the length of the message in bytes.
57745	<i>flags</i>	Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:
57747	MSG_EOR	Terminates a record (if supported by the protocol).
57748	MSG_OOB	Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.
57751	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.

57755 The length of the message to be sent is specified by the *length* argument. If the message is too
 57756 long to pass through the underlying protocol, *send()* shall fail and no data shall be transmitted.

57757 Successful completion of a call to *send()* does not guarantee delivery of the message. A return
 57758 value of -1 indicates only locally-detected errors.

57759 If space is not available at the sending socket to hold the message to be transmitted, and the
 57760 socket file descriptor does not have O_NONBLOCK set, *send()* shall block until space is
 57761 available. If space is not available at the sending socket to hold the message to be transmitted,
 57762 and the socket file descriptor does have O_NONBLOCK set, *send()* shall fail. The *select()* and
 57763 *poll()* functions can be used to determine when it is possible to send more data.

57764 The socket in use may require the process to have appropriate privileges to use the *send()*
 57765 function.

57766 **RETURN VALUE**

57767 Upon successful completion, *send()* shall return the number of bytes sent. Otherwise, -1 shall be
 57768 returned and *errno* set to indicate the error.

57769 **ERRORS**57770 The *send()* function shall fail if:

57771 [EAGAIN] or [EWOULDBLOCK]

57772 The socket's file descriptor is marked O_NONBLOCK and the requested
 57773 operation would block.

- 57774 [EBADF] The *socket* argument is not a valid file descriptor.
- 57775 [ECONNRESET] A connection was forcibly closed by a peer.
- 57776 [EDESTADDRREQ]
57777 The socket is not connection-mode and no peer address is set.
- 57778 [EINTR] A signal interrupted *send()* before any data was transmitted.
- 57779 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.
- 57780 [ENOTCONN] The socket is not connected.
- 57781 [ENOTSOCK] The *socket* argument does not refer to a socket.
- 57782 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
57783 more of the values set in *flags*.
- 57784 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
57785 no longer connected. In the latter case, and if the socket is of type
57786 SOCK_STREAM, the SIGPIPE signal is generated to the calling thread.
- 57787 The *send()* function may fail if:
- 57788 [EACCES] The calling process does not have the appropriate privileges.
- 57789 [EIO] An I/O error occurred while reading from or writing to the file system.
- 57790 [ENETDOWN] The local network interface used to reach the destination is down.
- 57791 [ENETUNREACH]
57792 No route to the network is present.
- 57793 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

EXAMPLES

None.

APPLICATION USAGE

The *send()* function is equivalent to *sendto()* with a null pointer *dest_len* argument, and to *write()* if no flags are used.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[*connect\(\)*](#), [*getsockopt\(\)*](#), [*poll\(\)*](#), [*pselect\(\)*](#), [*recv\(\)*](#), [*recvfrom\(\)*](#), [*recvmsg\(\)*](#), [*sendmsg\(\)*](#), [*sendto\(\)*](#), [*setsockopt\(\)*](#), [*shutdown\(\)*](#), [*socket\(\)*](#)

XBD [<sys/socket.h>](#)

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

Issue 7

Austin Group Interpretation 1003.1-2001 #035 is applied, updating the DESCRIPTION to clarify the behavior when the socket is a connectionless-mode socket.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

57814 **NAME**

57815 sendmsg — send a message on a socket using a message structure

57816 **SYNOPSIS**

57817 #include <sys/socket.h>

57818 ssize_t sendmsg(int socket, const struct msghdr *message, int flags);

57819 **DESCRIPTION**

57820 The *sendmsg()* function shall send a message through a connection-mode or connectionless-
 57821 mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the
 57822 address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has
 57823 been pre-specified, either the message shall be sent to the address specified in **msghdr**
 57824 (overriding the pre-specified peer address), or the function shall return -1 and set *errno* to
 57825 [EISCONN]. If the socket is connection-mode, the destination address in **msghdr** shall be
 57826 ignored.

57827 The *sendmsg()* function takes the following arguments:

57828	<i>socket</i>	Specifies the socket file descriptor.
57829	<i>message</i>	Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The <i>msg_flags</i> member is ignored.
57832	<i>flags</i>	Specifies the type of message transmission. The application may specify 0 or the following flag:
57834	MSG_EOR	Terminates a record (if supported by the protocol).
57835	MSG_OOB	Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
57838	MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The [EPIPE] error shall still be returned.

57842 The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to
 57843 be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of
 57844 this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field
 57845 gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated
 57846 by *msg_iov* is sent in turn.

57847 Successful completion of a call to *sendmsg()* does not guarantee delivery of the message. A
 57848 return value of -1 indicates only locally-detected errors.

57849 If space is not available at the sending socket to hold the message to be transmitted and the
 57850 socket file descriptor does not have O_NONBLOCK set, the *sendmsg()* function shall block until
 57851 space is available. If space is not available at the sending socket to hold the message to be
 57852 transmitted and the socket file descriptor does have O_NONBLOCK set, the *sendmsg()* function
 57853 shall fail.

57854 If the socket protocol supports broadcast and the specified address is a broadcast address for the
 57855 socket protocol, *sendmsg()* shall fail if the SO_BROADCAST option is not set for the socket.

57856 The socket in use may require the process to have appropriate privileges to use the *sendmsg()*
 57857 function.

57858 **RETURN VALUE**

57859 Upon successful completion, *sendmsg()* shall return the number of bytes sent. Otherwise, *-1*
 57860 shall be returned and *errno* set to indicate the error.

57861 **ERRORS**

57862 The *sendmsg()* function shall fail if:

57863 [EAGAIN] or [EWOULDBLOCK]

57864 The socket's file descriptor is marked *O_NONBLOCK* and the requested
 57865 operation would block.

57866 [EAFNOSUPPORT]

57867 Addresses in the specified address family cannot be used with this socket.

57868 [EBADF] The *socket* argument is not a valid file descriptor.

57869 [ECONNRESET] A connection was forcibly closed by a peer.

57870 [EINTR] A signal interrupted *sendmsg()* before any data was transmitted.

57871 [EINVAL] The sum of the *iov_len* values overflows an *ssize_t*.

57872 [EMSGSIZE] The message is too large to be sent all at once (as the socket requires), or the
 57873 *msg_iovlen* member of the *msghdr* structure pointed to by *message* is less than
 57874 or equal to 0 or is greater than *{IOV_MAX}*.

57875 [ENOTCONN] The socket is connection-mode but is not connected.

57876 [ENOTSOCK] The *socket* argument does not refer to a socket.

57877 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
 57878 more of the values set in *flags*.

57879 [EPIPE] The socket is shut down for writing, or the socket is connection-mode and is
 57880 no longer connected. In the latter case, and if the socket is of type
 57881 *SOCK_STREAM*, the *SIGPIPE* signal is generated to the calling thread.

57882 If the address family of the socket is *AF_UNIX*, then *sendmsg()* shall fail if:

57883 [EIO] An I/O error occurred while reading from or writing to the file system.

57884 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
 57885 in the socket address.

57886 [ENAMETOOLONG]

57887 A component of a pathname exceeded *{NAME_MAX}* characters, or an entire
 57888 pathname exceeded *{PATH_MAX}* characters.

57889 [ENOENT] A component of the pathname does not name an existing file or the path name
 57890 is an empty string.

57891 [ENOTDIR] A component of the path prefix of the pathname in the socket address is not a
 57892 directory.

57893 The *sendmsg()* function may fail if:

57894 [EACCES] Search permission is denied for a component of the path prefix; or write access
 57895 to the named socket is denied.

57896 [EDESTADDRREQ]

57897 The socket is not connection-mode and does not have its peer address set, and
 57898 no destination address was specified.

57899	[EHOSTUNREACH]	
57900		The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).
57901		
57902	[EIO]	An I/O error occurred while reading from or writing to the file system.
57903	[EISCONN]	A destination address was specified and the socket is already connected.
57904	[ENETDOWN]	The local network interface used to reach the destination is down.
57905	[ENETUNREACH]	
57906		No route to the network is present.
57907	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
57908	[ENOMEM]	Insufficient memory was available to fulfill the request.
57909		If the address family of the socket is AF_UNIX, then <i>sendmsg()</i> may fail if:
57910	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.
57911		
57912	[ENAMETOOLONG]	
57913		Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
57914		

EXAMPLES

Done.

APPLICATION USAGE

The *select()* and *poll()* functions can be used to determine when it is possible to send more data.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt(), *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*, *shutdown()*, *socket()*

XBD <[sys/socket.h](#)>

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretation 1003.1-2001 #073 is applied, updating the DESCRIPTION.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

57935 **NAME**57936 `sendto` — send a message on a socket57937 **SYNOPSIS**

```
57938 #include <sys/socket.h>
57939
57939 ssize_t sendto(int socket, const void *message, size_t length,
57940               int flags, const struct sockaddr *dest_addr,
57941               socklen_t dest_len);
```

57942 **DESCRIPTION**

57943 The `sendto()` function shall send a message through a connection-mode or connectionless-mode
 57944 socket. If the socket is a connectionless-mode socket, the message shall be sent to the address
 57945 specified by `dest_addr` if no pre-specified peer address has been set. If a peer address has been
 57946 pre-specified, either the message shall be sent to the address specified by `dest_addr` (overriding
 57947 the pre-specified peer address), or the function shall return `-1` and set `errno` to `[EISCONN]`. If
 57948 the socket is connection-mode, `dest_addr` shall be ignored.

57949 The `sendto()` function takes the following arguments:

57950	<code>socket</code>	Specifies the socket file descriptor.
57951	<code>message</code>	Points to a buffer containing the message to be sent.
57952	<code>length</code>	Specifies the size of the message in bytes.
57953	<code>flags</code>	Specifies the type of message transmission. Values of this argument are 57954 formed by logically OR'ing zero or more of the following flags:
57955	<code>MSG_EOR</code>	Terminates a record (if supported by the protocol).
57956	<code>MSG_OOB</code>	Sends out-of-band data on sockets that support out-of- 57957 band data. The significance and semantics of out-of- 57958 band data are protocol-specific.
57959	<code>MSG_NOSIGNAL</code>	Requests not to send the SIGPIPE signal if an attempt to 57960 send is made on a stream-oriented socket that is no 57961 longer connected. The <code>[EPIPE]</code> error shall still be 57962 returned.
57963	<code>dest_addr</code>	Points to a <code>sockaddr</code> structure containing the destination address. The length 57964 and format of the address depend on the address family of the socket.
57965	<code>dest_len</code>	Specifies the length of the <code>sockaddr</code> structure pointed to by the <code>dest_addr</code> 57966 argument.

57967 If the socket protocol supports broadcast and the specified address is a broadcast address for the
 57968 socket protocol, `sendto()` shall fail if the `SO_BROADCAST` option is not set for the socket.

57969 The `dest_addr` argument specifies the address of the target. The `length` argument specifies the
 57970 length of the message.

57971 Successful completion of a call to `sendto()` does not guarantee delivery of the message. A return
 57972 value of `-1` indicates only locally-detected errors.

57973 If space is not available at the sending socket to hold the message to be transmitted and the
 57974 socket file descriptor does not have `O_NONBLOCK` set, `sendto()` shall block until space is
 57975 available. If space is not available at the sending socket to hold the message to be transmitted
 57976 and the socket file descriptor does have `O_NONBLOCK` set, `sendto()` shall fail.

57977 The socket in use may require the process to have appropriate privileges to use the `sendto()`

sendto()

57978 function.

57979 **RETURN VALUE**

57980 Upon successful completion, *sendto()* shall return the number of bytes sent. Otherwise, `-1` shall
57981 be returned and *errno* set to indicate the error.

57982 **ERRORS**

57983 The *sendto()* function shall fail if:

57984 [EAFNOSUPPORT]

57985 Addresses in the specified address family cannot be used with this socket.

57986 [EAGAIN] or [EWOULDBLOCK]

57987 The socket's file descriptor is marked `O_NONBLOCK` and the requested
57988 operation would block.

57989 [EBADF] The *socket* argument is not a valid file descriptor.

57990 [ECONNRESET] A connection was forcibly closed by a peer.

57991 [EINTR] A signal interrupted *sendto()* before any data was transmitted.

57992 [EMSGSIZE] The message is too large to be sent all at once, as the socket requires.

57993 [ENOTCONN] The socket is connection-mode but is not connected.

57994 [ENOTSOCK] The *socket* argument does not refer to a socket.

57995 [EOPNOTSUPP] The *socket* argument is associated with a socket that does not support one or
57996 more of the values set in *flags*.

57997 [EPIPE]

57998 The socket is shut down for writing, or the socket is connection-mode and is
57999 no longer connected. In the latter case, and if the socket is of type
`SOCK_STREAM`, the `SIGPIPE` signal is generated to the calling thread.

58000 If the address family of the socket is `AF_UNIX`, then *sendto()* shall fail if:

58001 [EIO] An I/O error occurred while reading from or writing to the file system.

58002 [ELOOP] A loop exists in symbolic links encountered during resolution of the pathname
58003 in the socket address.

58004 [ENAMETOOLONG]

58005 A component of a pathname exceeded `{NAME_MAX}` characters, or an entire
58006 pathname exceeded `{PATH_MAX}` characters.

58007 [ENOENT]

58008 A component of the pathname does not name an existing file or the pathname
is an empty string.

58009 [ENOTDIR]

58010 A component of the path prefix of the pathname in the socket address is not a
directory.

58011 The *sendto()* function may fail if:

58012 [EACCES]

58013 Search permission is denied for a component of the path prefix; or write access
to the named socket is denied.

58014 [EDESTADDRREQ]

58015 The socket is not connection-mode and does not have its peer address set, and
58016 no destination address was specified.

58017 [EHOSTUNREACH]

58018 The destination host cannot be reached (probably because the host is down or
58019 a remote router cannot reach it).

58020	[EINVAL]	The <i>dest_len</i> argument is not a valid length for the address family.
58021	[EIO]	An I/O error occurred while reading from or writing to the file system.
58022	[EISCONN]	A destination address was specified and the socket is already connected.
58023	[ENETDOWN]	The local network interface used to reach the destination is down.
58024	[ENETUNREACH]	
58025		No route to the network is present.
58026	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
58027	[ENOMEM]	Insufficient memory was available to fulfill the request.
58028		If the address family of the socket is AF_UNIX, then <i>sendto()</i> may fail if:
58029	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.
58030		
58031	[ENAMETOOLONG]	
58032		Pathname resolution of a symbolic link produced an intermediate result whose length exceeds {PATH_MAX}.
58033		

EXAMPLES

None.

APPLICATION USAGEThe *select()* and *poll()* functions can be used to determine when it is possible to send more data.**RATIONALE**

None.

FUTURE DIRECTIONS

None.

SEE ALSO*getsockopt()*, *poll()*, *pselect()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socket()*XBD <[sys/socket.h](#)>**CHANGE HISTORY**

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

The wording of the mandatory [ELOOP] error condition is updated, and a second optional [ELOOP] error condition is added.

Issue 7

Austin Group Interpretations 1003.1-2001 #035 and #073 are applied, updating the [EISCONN] error and the DESCRIPTION.

The MSG_NOSIGNAL flag is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

58055 **NAME**58056 `setbuf` — assign buffering to a stream58057 **SYNOPSIS**58058 `#include <stdio.h>`58059 `void setbuf(FILE *restrict stream, char *restrict buf);`58060 **DESCRIPTION**58061 CX The functionality described on this reference page is aligned with the ISO C standard. Any
58062 conflict between the requirements described here and the ISO C standard is unintentional. This
58063 volume of POSIX.1-200x defers to the ISO C standard.

58064 Except that it returns no value, the function call:

58065 `setbuf(stream, buf)`

58066 shall be equivalent to:

58067 `setvbuf(stream, buf, _IOFBF, BUFSIZ)`58068 if *buf* is not a null pointer, or to:58069 `setvbuf(stream, buf, _IONBF, BUFSIZ)`58070 if *buf* is a null pointer.58071 **RETURN VALUE**58072 The `setbuf()` function shall not return a value.58073 **ERRORS**

58074 No errors are defined.

58075 **EXAMPLES**

58076 None.

58077 **APPLICATION USAGE**58078 A common source of error is allocating buffer space as an “automatic” variable in a code block,
58079 and then failing to close the stream in the same block.58080 With `setbuf()`, allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ
58081 bytes are used for the buffer area.58082 **RATIONALE**

58083 None.

58084 **FUTURE DIRECTIONS**

58085 None.

58086 **SEE ALSO**58087 [*fopen\(\)*](#), [*setvbuf\(\)*](#)58088 XBD [`<stdio.h>`](#)58089 **CHANGE HISTORY**

58090 First released in Issue 1. Derived from Issue 1 of the SVID.

58091 **Issue 6**58092 The prototype for `setbuf()` is updated for alignment with the ISO/IEC 9899:1999 standard.

58093 **NAME**

58094 setegid — set the effective group ID

58095 **SYNOPSIS**

58096 #include <unistd.h>

58097 int setegid(gid_t gid);

58098 **DESCRIPTION**58099 If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate
58100 privileges, *setegid()* shall set the effective group ID of the calling process to *gid*; the real group
58101 ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.58102 The *setegid()* function shall not affect the supplementary group list in any way.58103 **RETURN VALUE**58104 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
58105 indicate the error.58106 **ERRORS**58107 The *setegid()* function shall fail if:58108 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
58109 implementation.58110 [EPERM] The process does not have appropriate privileges and *gid* does not match the
58111 real group ID or the saved set-group-ID.58112 **EXAMPLES**

58113 None.

58114 **APPLICATION USAGE**

58115 None.

58116 **RATIONALE**58117 Refer to the RATIONALE section in *setuid()*.58118 **FUTURE DIRECTIONS**

58119 None.

58120 **SEE ALSO**58121 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

58122 XBD <sys/types.h>, <unistd.h>

58123 **CHANGE HISTORY**

58124 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

58125 **NAME**

58126 setenv — add or change environment variable

58127 **SYNOPSIS**

```
58128 CX #include <stdlib.h>
58129 int setenv(const char *envname, const char *envval, int overwrite);
```

58130 **DESCRIPTION**

58131 The *setenv()* function shall update or add a variable in the environment of the calling process.
 58132 The *envname* argument points to a string containing the name of an environment variable to be
 58133 added or altered. The environment variable shall be set to the value to which *envval* points. The
 58134 function shall fail if *envname* points to a string which contains an '=' character. If the
 58135 environment variable named by *envname* already exists and the value of *overwrite* is non-zero,
 58136 the function shall return success and the environment shall be updated. If the environment
 58137 variable named by *envname* already exists and the value of *overwrite* is zero, the function shall
 58138 return success and the environment shall remain unchanged.

58139 If the application modifies *environ* or the pointers to which it points, the behavior of *setenv()* is
 58140 undefined. The *setenv()* function shall update the list of pointers to which *environ* points.

58141 The strings described by *envname* and *envval* are copied by this function.

58142 The *setenv()* function need not be thread-safe. A function that is not required to be thread-safe is
 58143 not required to be reentrant.

58144 **RETURN VALUE**

58145 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 58146 indicate the error, and the environment shall be unchanged.

58147 **ERRORS**

58148 The *setenv()* function shall fail if:

58149 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a
 58150 string containing an '=' character.

58151 [ENOMEM] Insufficient memory was available to add a variable or its value to the
 58152 environment.

58153 **EXAMPLES**

58154 None.

58155 **APPLICATION USAGE**

58156 See *exec*, for restrictions on changing the environment in multi-threaded applications.

58157 **RATIONALE**

58158 Unanticipated results may occur if *setenv()* changes the external variable *environ*. In particular, if
 58159 the optional *envp* argument to *main()* is present, it is not changed, and thus may point to an
 58160 obsolete copy of the environment (as may any other copy of *environ*). However, other than the
 58161 aforementioned restriction, the standard developers intended that the traditional method of
 58162 walking through the environment by way of the *environ* pointer must be supported.

58163 It was decided that *setenv()* should be required by this version because it addresses a piece of
 58164 missing functionality, and does not impose a significant burden on the implementor.

58165 There was considerable debate as to whether the System V *putenv()* function or the BSD *setenv()*
 58166 function should be required as a mandatory function. The *setenv()* function was chosen because
 58167 it permitted the implementation of the *unsetenv()* function to delete environmental variables,

58168 without specifying an additional interface. The *putenv()* function is available as part of the XSI
58169 option.

58170 The standard developers considered requiring that *setenv()* indicate an error when a call to it
58171 would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might
58172 be temporary, with the application eventually reducing the environment size. The ultimate
58173 success or failure depends on the size at the time of a call to *exec*, which returns an indication of
58174 this error condition.

58175 FUTURE DIRECTIONS

58176 None.

58177 SEE ALSO

58178 *exec*, *getenv()*, *unsetenv()*

58179 XBD <stdlib.h>, <sys/types.h>, <unistd.h>

58180 CHANGE HISTORY

58181 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

58182 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/55 is applied, adding references to *exec* in
58183 the APPLICATION USAGE and SEE ALSO sections.

DRAFT

58184 **NAME**

58185 seteuid — set effective user ID

58186 **SYNOPSIS**

```
58187 #include <unistd.h>
58188 int seteuid(uid_t uid);
```

58189 **DESCRIPTION**

58190 If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate
 58191 privileges, *seteuid()* shall set the effective user ID of the calling process to *uid*; the real user ID
 58192 and saved set-user-ID shall remain unchanged.

58193 The *seteuid()* function shall not affect the supplementary group list in any way.

58194 **RETURN VALUE**

58195 Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and *errno* set to
 58196 indicate the error.

58197 **ERRORS**

58198 The *seteuid()* function shall fail if:

- | | | |
|-------|----------|---|
| 58199 | [EINVAL] | The value of the <i>uid</i> argument is invalid and is not supported by the implementation. |
| 58200 | | |
| 58201 | [EPERM] | The process does not have appropriate privileges and <i>uid</i> does not match the real user ID or the saved set-user-ID. |
| 58202 | | |

58203 **EXAMPLES**

58204 None.

58205 **APPLICATION USAGE**

58206 None.

58207 **RATIONALE**

58208 Refer to the RATIONALE section in *setuid()*.

58209 **FUTURE DIRECTIONS**

58210 None.

58211 **SEE ALSO**

58212 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *setgid()*, *setregid()*, *setreuid()*, *setuid()*

58213 XBD [<sys/types.h>](#), [<unistd.h>](#)

58214 **CHANGE HISTORY**

58215 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

58216 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/123 is applied, making an editorial
 58217 correction to the [EPERM] error in the ERRORS section.

58218 **NAME**

58219 setgid — set-group-ID

58220 **SYNOPSIS**

58221 #include <unistd.h>

58222 int setgid(gid_t gid);

58223 **DESCRIPTION**58224 If the process has appropriate privileges, *setgid()* shall set the real group ID, effective group ID,
58225 and the saved set-group-ID of the calling process to *gid*.58226 If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the
58227 saved set-group-ID, *setgid()* shall set the effective group ID to *gid*; the real group ID and saved
58228 set-group-ID shall remain unchanged.58229 The *setgid()* function shall not affect the supplementary group list in any way.

58230 Any supplementary group IDs of the calling process shall remain unchanged.

58231 **RETURN VALUE**58232 Upon successful completion, 0 is returned. Otherwise, -1 shall be returned and *errno* set to
58233 indicate the error.58234 **ERRORS**58235 The *setgid()* function shall fail if:58236 [EINVAL] The value of the *gid* argument is invalid and is not supported by the
58237 implementation.58238 [EPERM] The process does not have appropriate privileges and *gid* does not match the
58239 real group ID or the saved set-group-ID.58240 **EXAMPLES**

58241 None.

58242 **APPLICATION USAGE**

58243 None.

58244 **RATIONALE**58245 Refer to the RATIONALE section in *setuid()*.58246 **FUTURE DIRECTIONS**

58247 None.

58248 **SEE ALSO**58249 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setregid()*, *setreuid()*, *setuid()*

58250 XBD <sys/types.h>, <unistd.h>

58251 **CHANGE HISTORY**

58252 First released in Issue 1. Derived from Issue 1 of the SVID.

58253 **Issue 6**

58254 In the SYNOPSIS, the optional include of the <sys/types.h> header is removed.

58255 The following new requirements on POSIX implementations derive from alignment with the
58256 Single UNIX Specification:

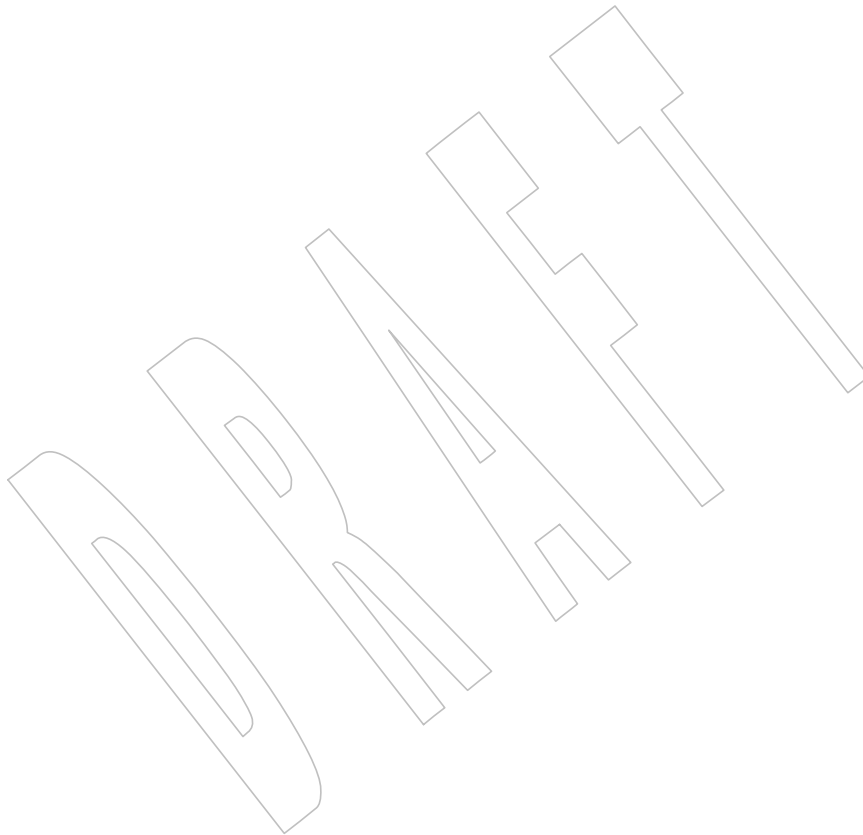
- 58257
- The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
58258 required for conforming implementations of previous POSIX specifications, it was not
58259 required for UNIX applications.

setgid()58260
58261
58262
58263
58264

- Functionality associated with `_POSIX_SAVED_IDS` is now mandated. This is a FIPS requirement.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effects of `setgid()` in processes without appropriate privileges are changed.
- A requirement that the supplementary group list is not affected is added.



58265 **NAME**
58266 setgrent — reset the group database to the first entry

58267 **SYNOPSIS**

58268 XSI #include <grp.h>
58269 void setgrent(void);

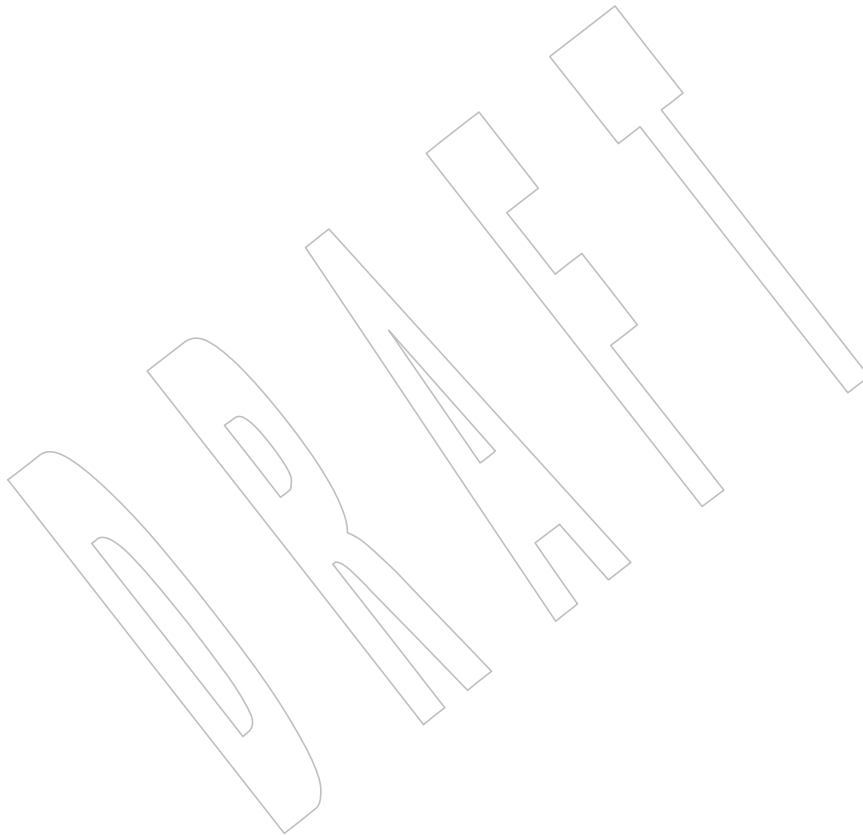
58270 **DESCRIPTION**

58271 Refer to *endgrent()*.

58272 **NAME**
58273 sethostent — network host database functions

58274 **SYNOPSIS**
58275 #include <netdb.h>
58276 void sethostent(int stayopen);

58277 **DESCRIPTION**
58278 Refer to *endhostent()*.



58279 **NAME**

58280 setitimer — set the value of an interval timer

58281 **SYNOPSIS**

58282 OB XSI #include <sys/time.h>

58283 int setitimer(int *which*, const struct itimerval *restrict *value*,
58284 struct itimerval *restrict *ovalue*);58285 **DESCRIPTION**58286 Refer to [getitimer\(\)](#).

58287 **NAME**

58288 setjmp — set jump point for a non-local goto

58289 **SYNOPSIS**

58290 #include <setjmp.h>

58291 int setjmp(jmp_buf env);

58292 **DESCRIPTION**

58293 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 58294 conflict between the requirements described here and the ISO C standard is unintentional. This
 58295 volume of POSIX.1-200x defers to the ISO C standard.

58296 A call to *setjmp()* shall save the calling environment in its *env* argument for later use by
 58297 *longjmp()*.

58298 It is unspecified whether *setjmp()* is a macro or a function. If a macro definition is suppressed in
 58299 order to access an actual function, or a program defines an external identifier with the name
 58300 *setjmp*, the behavior is undefined.

58301 An application shall ensure that an invocation of *setjmp()* appears in one of the following
 58302 contexts only:

- 58303
- The entire controlling expression of a selection or iteration statement
 - 58304 • One operand of a relational or equality operator with the other operand an integral
 58305 constant expression, with the resulting expression being the entire controlling expression
 58306 of a selection or iteration statement
 - 58307 • The operand of a unary '!' operator with the resulting expression being the entire
 58308 controlling expression of a selection or iteration
 - 58309 • The entire expression of an expression statement (possibly cast to **void**)

58310 If the invocation appears in any other context, the behavior is undefined.

58311 **RETURN VALUE**

58312 If the return is from a direct invocation, *setjmp()* shall return 0. If the return is from a call to
 58313 *longjmp()*, *setjmp()* shall return a non-zero value.

58314 **ERRORS**

58315 No errors are defined.

58316 **EXAMPLES**

58317 None.

58318 **APPLICATION USAGE**

58319 In general, *sigsetjmp()* is more useful in dealing with errors and interrupts encountered in a low-
 58320 level subroutine of a program.

58321 **RATIONALE**

58322 None.

58323 **FUTURE DIRECTIONS**

58324 None.

58325 **SEE ALSO**58326 *longjmp()*, *sigsetjmp()*

58327 XBD <setjmp.h>

58328

CHANGE HISTORY

58329

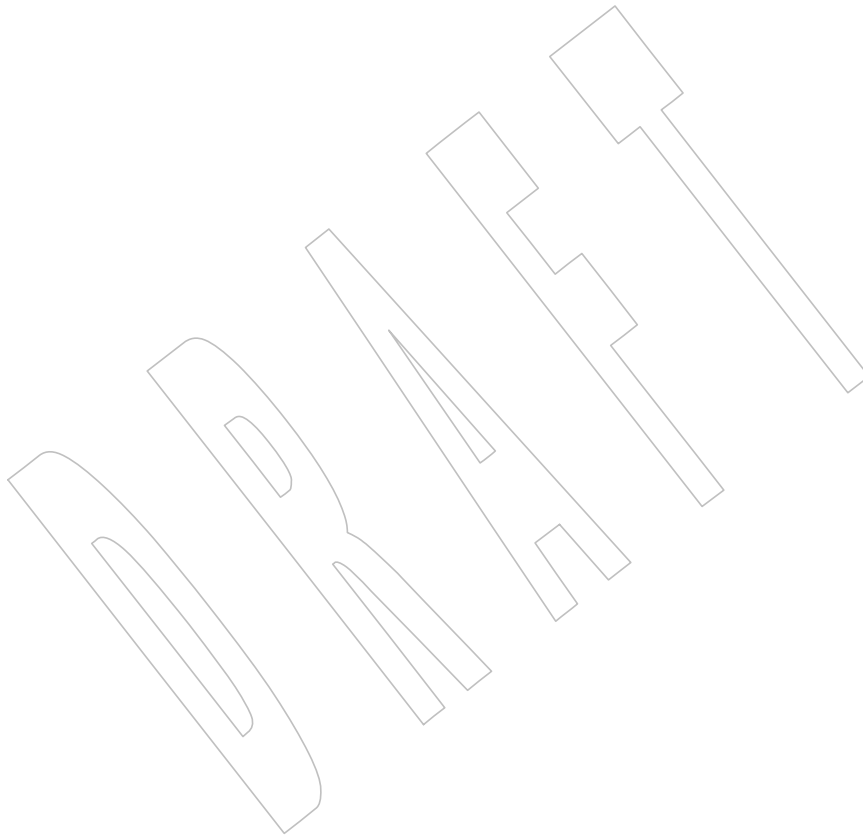
First released in Issue 1. Derived from Issue 1 of the SVID.

58330

Issue 6

58331

The normative text is updated to avoid use of the term “must” for application requirements.



58332 **NAME**58333 setkey — set encoding key (**CRYPT**)58334 **SYNOPSIS**

```
58335 XSI #include <stdlib.h>
58336 void setkey(const char *key);
```

58337 **DESCRIPTION**

58338 The *setkey()* function provides access to an implementation-defined encoding algorithm. The
 58339 argument of *setkey()* is an array of length 64 bytes containing only the bytes with numerical
 58340 value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is
 58341 ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used
 58342 with the algorithm to encode a string *block* passed to *encrypt()*.

58343 The *setkey()* function shall not change the setting of *errno* if successful. An application wishing to
 58344 check for error situations should set *errno* to 0 before calling *setkey()*. If *errno* is non-zero on
 58345 return, an error has occurred.

58346 The *setkey()* function need not be thread-safe. A function that is not required to be thread-safe is
 58347 not required to be reentrant.

58348 **RETURN VALUE**

58349 No values are returned.

58350 **ERRORS**58351 The *setkey()* function shall fail if:

58352 [ENOSYS] The functionality is not supported on this implementation.

58353 **EXAMPLES**

58354 None.

58355 **APPLICATION USAGE**

58356 Decoding need not be implemented in all environments. This is related to government
 58357 restrictions in some countries on encryption and decryption routines. Historical practice has
 58358 been to ship a different version of the encryption library without the decryption feature in the
 58359 routines supplied. Thus the exported version of *encrypt()* does encoding but not decoding.

58360 **RATIONALE**

58361 None.

58362 **FUTURE DIRECTIONS**

58363 None.

58364 **SEE ALSO**58365 *crypt()*, *encrypt()*58366 XBD [<stdlib.h>](#)58367 **CHANGE HISTORY**

58368 First released in Issue 1. Derived from Issue 1 of the SVID.

58369 **Issue 5**58370 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

58371 **NAME**

58372 setlocale — set program locale

58373 **SYNOPSIS**

58374 #include <locale.h>

58375 char *setlocale(int category, const char *locale);

58376 **DESCRIPTION**

58377 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 58378 conflict between the requirements described here and the ISO C standard is unintentional. This
 58379 volume of POSIX.1-200x defers to the ISO C standard.

58380 The *setlocale()* function selects the appropriate piece of the locale of the process, as specified by
 58381 the *category* and *locale* arguments, and may be used to change or query the entire locale of the
 58382 process or portions thereof. The value *LC_ALL* for *category* names the entire locale of the process;
 58383 other values for *category* name only a part of the locale of the process:

58384 *LC_COLLATE* Affects the behavior of regular expressions and the collation functions.

58385 *LC_CTYPE* Affects the behavior of regular expressions, character classification, character
 58386 conversion functions, and wide-character functions.

58387 CX *LC_MESSAGES* Affects what strings are expected by commands and utilities as affirmative or
 58388 negative responses.

58389 XSI It also affects what strings are given by commands and utilities as affirmative
 58390 or negative responses, and the content of messages.

58391 *LC_MONETARY* Affects the behavior of functions that handle monetary values.

58392 *LC_NUMERIC* Affects the behavior of functions that handle numeric values.

58393 *LC_TIME* Affects the behavior of the time conversion functions.

58394 The *locale* argument is a pointer to a character string containing the required setting of *category*.
 58395 The contents of this string are implementation-defined. In addition, the following preset values
 58396 of *locale* are defined for all settings of *category*:

58397 CX "POSIX" Specifies the minimal environment for C-language translation called the
 58398 POSIX locale. If *setlocale()* is not invoked, the POSIX locale is the default at
 58399 entry to *main()*.

58400 "C" Equivalent to "POSIX".

58401 CX "" Specifies an implementation-defined native environment. The determination
 58402 of the name of the new locale for the specified category depends on the value
 58403 of the associated environment variables, *LC_** and *LANG*; see XBD Chapter 7
 58404 (on page 121) and Chapter 8 (on page 159).

58405 A null pointer Used to direct *setlocale()* to query the current internationalized environment
 58406 and return the name of the locale.

58407 CX Setting all of the categories of the locale of the process is similar to successively setting each
 58408 individual category of the locale of the process, except that all error checking is done before any
 58409 actions are performed. To set all the categories of the locale of the process, *setlocale()* is invoked
 58410 as:

```
58411 setlocale(LC_ALL, "");
```

58412 In this case, *setlocale()* shall first verify that the values of all the environment variables it needs

58413 according to the precedence rules (described in XBD [Chapter 8](#), on page 159) indicate supported
 58414 locales. If the value of any of these environment variable searches yields a locale that is not
 58415 supported (and non-null), *setlocale()* shall return a null pointer and the locale of the process shall
 58416 not be changed. If all environment variables name supported locales, *setlocale()* shall proceed as
 58417 if it had been called for each category, using the appropriate value from the associated
 58418 environment variable or from the implementation-defined default if there is no such value.

58419 The locale state is common to all threads within a process.

RETURN VALUE

58420 Upon successful completion, *setlocale()* shall return the string associated with the specified
 58421 category for the new locale. Otherwise, *setlocale()* shall return a null pointer and the locale of the
 58422 process is not changed.
 58423

58424 A null pointer for *locale* causes *setlocale()* to return a pointer to the string associated with the
 58425 *category* for the current locale of the process. The locale of the process shall not be changed.

58426 The string returned by *setlocale()* is such that a subsequent call with that string and its associated
 58427 *category* shall restore that part of the locale of the process. The application shall not modify the
 58428 string returned which may be overwritten by a subsequent call to *setlocale()*.

ERRORS

58429 No errors are defined.
 58430

EXAMPLES

58431 None.
 58432

APPLICATION USAGE

58433 The following code illustrates how a program can initialize the international environment for
 58434 one language, while selectively modifying the locale of the process such that regular expressions
 58435 and string operations can be applied to text recorded in a different language:
 58436

```
58437 setlocale(LC_ALL, "De");
58438 setlocale(LC_COLLATE, "Fr@dict");
```

58439 Internationalized programs must call *setlocale()* to initiate a specific language operation. This can
 58440 be done by calling *setlocale()* as follows:

```
58441 setlocale(LC_ALL, "");
```

58442 Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened
 58443 by calls to *catopen()*.

RATIONALE

58444 The ISO C standard defines a collection of functions to support internationalization. One of the
 58445 most significant aspects of these functions is a facility to set and query the *international*
 58446 *environment*. The international environment is a repository of information that affects the
 58447 behavior of certain functionality, namely:
 58448

- 58449 1. Character handling
- 58450 2. Collating
- 58451 3. Date/time formatting
- 58452 4. Numeric editing
- 58453 5. Monetary formatting
- 58454 6. Messaging

58455 The *setlocale()* function provides the application developer with the ability to set all or portions,
 58456 called *categories*, of the international environment. These categories correspond to the areas of
 58457 functionality mentioned above. The syntax for *setlocale()* is as follows:

58458 `char *setlocale(int category, const char *locale);`

58459 where *category* is the name of one of following categories, namely:

58460 `LC_COLLATE`
 58461 `LC_CTYPE`
 58462 `LC_MESSAGES`
 58463 `LC_MONETARY`
 58464 `LC_NUMERIC`
 58465 `LC_TIME`

58466 In addition, a special value called `LC_ALL` directs `setlocale()` to set all categories.

58467 There are two primary uses of `setlocale()`:

- 58468 1. Querying the international environment to find out what it is set to
- 58469 2. Setting the international environment, or *locale*, to a specific value

58470 The behavior of `setlocale()` in these two areas is described below. Since it is difficult to describe
 58471 the behavior in words, examples are used to illustrate the behavior of specific uses.

58472 To query the international environment, `setlocale()` is invoked with a specific category and the
 58473 NULL pointer as the locale. The NULL pointer is a special directive to `setlocale()` that tells it to
 58474 query rather than set the international environment. The following syntax is used to query the
 58475 name of the international environment:

58476 `setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \`
 58477 `LC_NUMERIC, LC_TIME}, (char *) NULL);`

58478 The `setlocale()` function shall return the string corresponding to the current international
 58479 environment. This value may be used by a subsequent call to `setlocale()` to reset the international
 58480 environment to this value. However, it should be noted that the return value from `setlocale()`
 58481 may be a pointer to a static area within the function and is not guaranteed to remain unchanged
 58482 (that is, it may be modified by a subsequent call to `setlocale()`). Therefore, if the purpose of
 58483 calling `setlocale()` is to save the value of the current international environment so it can be
 58484 changed and reset later, the return value should be copied to an array of `char` in the calling
 58485 program.

58486 There are three ways to set the international environment with `setlocale()`:

58487 `setlocale(category, string)`

58488 This usage sets a specific *category* in the international environment to a specific value
 58489 corresponding to the value of the *string*. A specific example is provided below:

58490 `setlocale(LC_ALL, "fr_FR.ISO-8859-1");`

58491 In this example, all categories of the international environment are set to the locale
 58492 corresponding to the string "fr_FR.ISO-8859-1", or to the French language as spoken in
 58493 France using the ISO/IEC 8859-1:1998 standard codeset.

58494 If the string does not correspond to a valid locale, `setlocale()` shall return a NULL pointer
 58495 and the international environment is not changed. Otherwise, `setlocale()` shall return the
 58496 name of the locale just set.

58497 `setlocale(category, "C")`

58498 The ISO C standard states that one locale must exist on all conforming implementations.
 58499 The name of the locale is C and corresponds to a minimal international environment needed
 58500 to support the C programming language.

58501 `setlocale(category, "")`

58502 This sets a specific category to an implementation-defined default. This corresponds to the
58503 value of the environment variables.

58504 **FUTURE DIRECTIONS**

58505 None.

58506 **SEE ALSO**

58507 *exec*, *fprintf()*, *fscanf()*, *isalnum()*, *isalpha()*, *isblank()*, *iscntrl()*, *isdigit()*, *isgraph()*, *islower()*,
58508 *isprint()*, *ispunct()*, *isspace()*, *isupper()*, *iswalnum()*, *iswalpha()*, *iswblank()*, *iswcntrl()*, *iswctype()*,
58509 *iswdigit()*, *iswgraph()*, *iswlower()*, *iswprint()*, *iswpunct()*, *iswspace()*, *iswupper()*, *iswxdigit()*,
58510 *isxdigit()*, *localeconv()*, *mblen()*, *mbstowcs()*, *mbtowc()*, *nl_langinfo()*, *setlocale()*, *strcoll()*,
58511 *strerror()*, *strfmon()*, *strsignal()*, *strtod()*, *strxfrm()*, *tolower()*, *toupper()*, *towlower()*, *towupper()*,
58512 *uselocale()*, *wscoll()*, *wctod()*, *wcstombs()*, *wcsxfrm()*, *wctomb()*

58513 XBD [Chapter 7](#) (on page 121), [Chapter 8](#) (on page 159), [<langinfo.h>](#), [<locale.h>](#)

58514 **CHANGE HISTORY**

58515 First released in Issue 3.

58516 **Issue 5**

58517 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

58518 **Issue 6**

58519 Extensions beyond the ISO C standard are marked.

58520 The normative text is updated to avoid use of the term “must” for application requirements.

58521 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/124 is applied, updating the
58522 DESCRIPTION to clarify the behavior of:

58523 `setlocale(LC_ALL, "");`

58524 **Issue 7**

58525 Functionality relating to the Threads option is moved to the Base.

58526 **NAME**
58527 setlogmask — set the log priority mask

58528 **SYNOPSIS**

58529 XSI #include <syslog.h>
58530 int setlogmask(int *maskpri*);

58531 **DESCRIPTION**

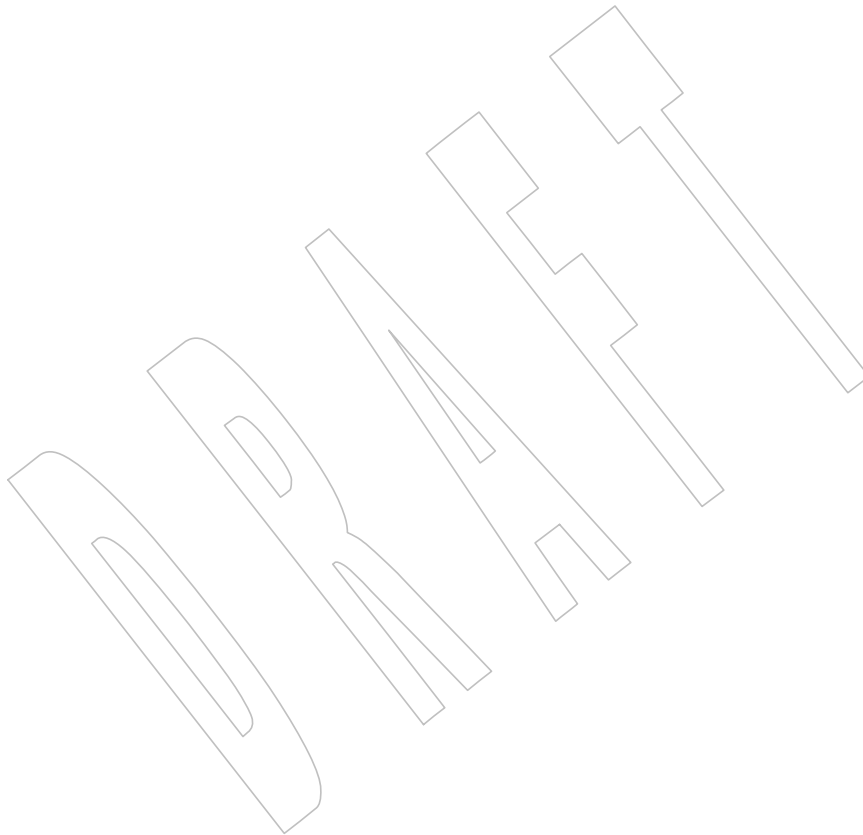
58532 Refer to *closelog()*.

setnetent()

58533 **NAME**
58534 setnetent — network database function

58535 **SYNOPSIS**
58536 #include <netdb.h>
58537 void setnetent(int *stayopen*);

58538 **DESCRIPTION**
58539 Refer to *endnetent()*.



58540 **NAME**

58541 setpgid — set process group ID for job control

58542 **SYNOPSIS**

58543 #include <unistd.h>

58544 int setpgid(pid_t pid, pid_t pgid);

58545 **DESCRIPTION**

58546 The *setpgid()* function shall either join an existing process group or create a new process group
 58547 within the session of the calling process. The process group ID of a session leader shall not
 58548 change. Upon successful completion, the process group ID of the process with a process ID that
 58549 matches *pid* shall be set to *pgid*. As a special case, if *pid* is 0, the process ID of the calling process
 58550 shall be used. Also, if *pgid* is 0, the process ID of the indicated process shall be used.

58551 **RETURN VALUE**

58552 Upon successful completion, *setpgid()* shall return 0; otherwise, -1 shall be returned and *errno*
 58553 shall be set to indicate the error.

58554 **ERRORS**58555 The *setpgid()* function shall fail if:

58556 [EACCES] The value of the *pid* argument matches the process ID of a child process of the
 58557 calling process and the child process has successfully executed one of the *exec*
 58558 functions.

58559 [EINVAL] The value of the *pgid* argument is less than 0, or is not a value supported by
 58560 the implementation.

58561 [EPERM] The process indicated by the *pid* argument is a session leader.

58562 [EPERM] The value of the *pid* argument matches the process ID of a child process of the
 58563 calling process and the child process is not in the same session as the calling
 58564 process.

58565 [EPERM] The value of the *pgid* argument is valid but does not match the process ID of
 58566 the process indicated by the *pid* argument and there is no process with a
 58567 process group ID that matches the value of the *pgid* argument in the same
 58568 session as the calling process.

58569 [ESRCH] The value of the *pid* argument does not match the process ID of the calling
 58570 process or of a child process of the calling process.

58571 **EXAMPLES**

58572 None.

58573 **APPLICATION USAGE**

58574 None.

58575 **RATIONALE**

58576 The *setpgid()* function shall group processes together for the purpose of signaling, placement in
 58577 foreground or background, and other job control actions.

58578 The *setpgid()* function is similar to the *setpgrp()* function of 4.2 BSD, except that 4.2 BSD allowed
 58579 the specified new process group to assume any value. This presents certain security problems
 58580 and is more flexible than necessary to support job control.

58581 To provide tighter security, *setpgid()* only allows the calling process to join a process group
 58582 already in use inside its session or create a new process group whose process group ID was
 58583 equal to its process ID.

58584 When a job control shell spawns a new job, the processes in the job must be placed into a new
58585 process group via *setpgid()*. There are two timing constraints involved in this action:

- 58586 1. The new process must be placed in the new process group before the appropriate
58587 program is launched via one of the *exec* functions.
- 58588 2. The new process must be placed in the new process group before the shell can correctly
58589 send signals to the new process group.

58590 To address these constraints, the following actions are performed. The new processes call
58591 *setpgid()* to alter their own process groups after *fork()* but before *exec*. This satisfies the first
58592 constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of
58593 *vfork()*; that is, the shell is suspended until the child has completed the *exec*, thus ensuring that
58594 the child has completed the *setpgid()*. A new version of *fork()* with this same synchronization
58595 property was considered, but it was decided instead to merely allow the parent shell process to
58596 adjust the process group of its child processes via *setpgid()*. Both timing constraints are now
58597 satisfied by having both the parent shell and the child attempt to adjust the process group of the
58598 child process; it does not matter which succeeds first.

58599 Since it would be confusing to an application to have its process group change after it began
58600 executing (that is, after *exec*), and because the child process would already have adjusted its
58601 process group before this, the [EACCES] error was added to disallow this.

58602 One non-obvious use of *setpgid()* is to allow a job control shell to return itself to its original
58603 process group (the one in effect when the job control shell was executed). A job control shell
58604 does this before returning control back to its parent when it is terminating or suspending itself as
58605 a way of restoring its job control “state” back to what its parent would expect. (Note that the
58606 original process group of the job control shell typically matches the process group of its parent,
58607 but this is not necessarily always the case.)

58608 FUTURE DIRECTIONS

58609 None.

58610 SEE ALSO

58611 *exec*, *getpgrp()*, *setsid()*, *tcsetpgrp()*

58612 XBD [<sys/types.h>](#), [<unistd.h>](#)

58613 CHANGE HISTORY

58614 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

58615 Issue 6

58616 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

58617 The following new requirements on POSIX implementations derive from alignment with the
58618 Single UNIX Specification:

- 58619 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
58620 required for conforming implementations of previous POSIX specifications, it was not
58621 required for UNIX applications.
- 58622 • The *setpgid()* function is mandatory since `_POSIX_JOB_CONTROL` is required to be
58623 defined in this version. This is a FIPS requirement.

58624 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/56 is applied, changing the wording in
58625 the DESCRIPTION from “the process group ID of the indicated process shall be used” to “the
58626 process ID of the indicated process shall be used”. This change reverts the wording to as in the
58627 ISO POSIX-1: 1996 standard; it appeared to be an unintentional change.

58628 **NAME**
 58629 setpgrp — set the process group ID

58630 **SYNOPSIS**

58631 OB XSI #include <unistd.h>
 58632 pid_t setpgrp(void);

58633 **DESCRIPTION**

58634 If the calling process is not already a session leader, *setpgrp()* sets the process group ID of the
 58635 calling process to the process ID of the calling process. If *setpgrp()* creates a new session, then the
 58636 new session has no controlling terminal.

58637 The *setpgrp()* function has no effect when the calling process is a session leader.

58638 **RETURN VALUE**

58639 Upon completion, *setpgrp()* shall return the process group ID.

58640 **ERRORS**

58641 No errors are defined.

58642 **EXAMPLES**

58643 None.

58644 **APPLICATION USAGE**

58645 It is unspecified whether this function behaves as *setpgid(0,0)* or *setsid()* unless the process is
 58646 already a session leader. Therefore, applications are encouraged to use *setpgid()* or *setsid()* as
 58647 appropriate.

58648 **RATIONALE**

58649 None.

58650 **FUTURE DIRECTIONS**

58651 The *setpgrp()* function may be removed in a future version.

58652 **SEE ALSO**

58653 *exec, fork(), getpid(), getsid(), kill, setpgid(), setsid()*

58654 XBD <unistd.h>

58655 **CHANGE HISTORY**

58656 First released in Issue 4, Version 2.

58657 **Issue 5**

58658 Moved from X/OPEN UNIX extension to BASE.

58659 **Issue 7**

58660 The *setpgrp()* function is marked obsolescent.

setpriority()

58661 **NAME**
58662 setpriority — set the nice value

58663 **SYNOPSIS**

58664 XSI #include <sys/resource.h>
58665 int setpriority(int *which*, id_t *who*, int *nice*);

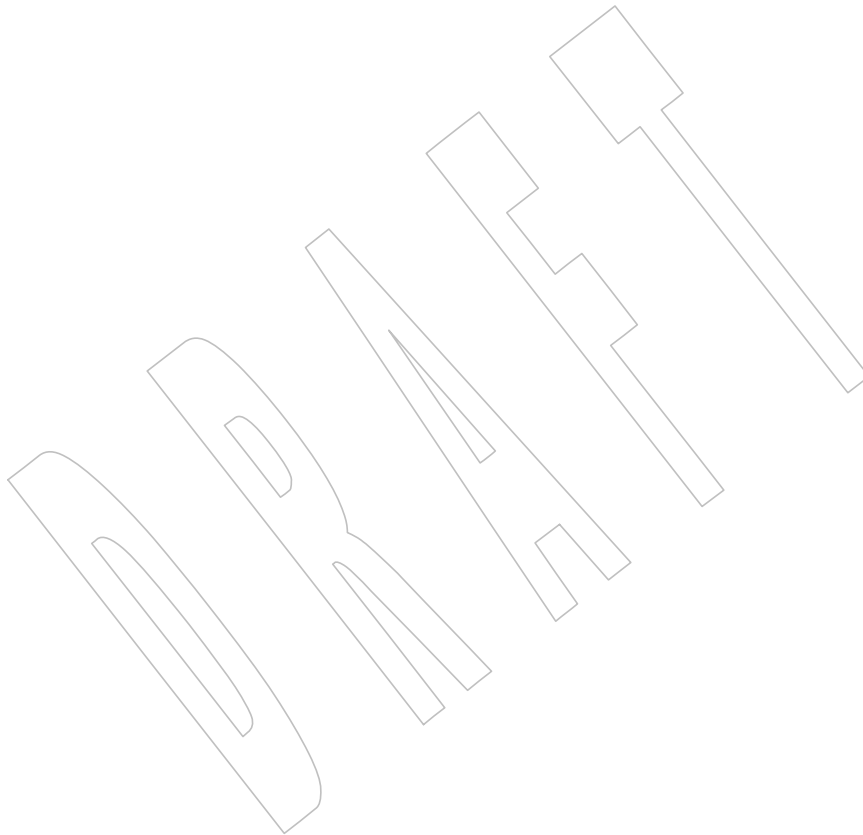
58666 **DESCRIPTION**

58667 Refer to [getpriority\(\)](#).

58668 **NAME**
58669 setprotoent — network protocol database functions

58670 **SYNOPSIS**
58671 #include <netdb.h>
58672 void setprotoent(int stayopen);

58673 **DESCRIPTION**
58674 Refer to *endprotoent()*.



setpwent()

58675 **NAME**
58676 setpwent — user database function

SYNOPSIS

58677 XSI #include <pwd.h>
58678
58679 void setpwent(void);

DESCRIPTION

58680 Refer to *endpwent()*.
58681

58682 **NAME**

58683 setregid — set real and effective group IDs

58684 **SYNOPSIS**

```
58685 XSI #include <unistd.h>
58686 int setregid(gid_t rgid, gid_t egid);
```

58687 **DESCRIPTION**58688 The *setregid()* function shall set the real and effective group IDs of the calling process.58689 If *rgid* is -1 , the real group ID shall not be changed; if *egid* is -1 , the effective group ID shall not
58690 be changed.

58691 The real and effective group IDs may be set to different values in the same call.

58692 Only a process with appropriate privileges can set the real group ID and the effective group ID
58693 to any valid value.58694 A non-privileged process can set either the real group ID to the saved set-group-ID from one of
58695 the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group
58696 ID.58697 If the real group ID is being set (*rgid* is not -1), or the effective group ID is being set to a value +
58698 not equal to the real group ID, then the saved set-group-ID of the current process shall be set +
58699 equal to the new effective group ID. +

58700 Any supplementary group IDs of the calling process remain unchanged.

58701 **RETURN VALUE**58702 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
58703 indicate the error, and neither of the group IDs are changed.58704 **ERRORS**58705 The *setregid()* function shall fail if:58706 [EINVAL] The value of the *rgid* or *egid* argument is invalid or out-of-range.58707 [EPERM] The process does not have appropriate privileges and a change other than
58708 changing the real group ID to the saved set-group-ID, or changing the
58709 effective group ID to the real group ID or the saved set-group-ID, was
58710 requested.58711 **EXAMPLES**

58712 None.

58713 **APPLICATION USAGE**58714 If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can |
58715 only set its effective group ID back to the previous value if *rgid* was -1 in the *setregid()* call, since |
58716 the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the |
58717 *setregid()* call, then the saved set-group-ID will also have been changed to the real user ID. |58718 **RATIONALE**58719 Earlier versions of this standard did not specify whether the saved set-group-ID was affected by |
58720 *setregid()* calls. This version specifies common existing practice that constitutes an important |
58721 security feature. The ability to set both the effective group ID and saved set-group-ID to be the |
58722 same as the real group ID means that any security weakness in code that is executed after that |
58723 point cannot result in malicious code being executed with the previous effective group ID. |
58724 Privileged applications could already do this using just *setgid()*, but for non-privileged |

setregid()

58725 applications the only standard method available is to use this feature of *setregid()*.

58726 **FUTURE DIRECTIONS**

58727 None.

58728 **SEE ALSO**

58729 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setreuid()*, *setuid()*

58730 XBD <[unistd.h](#)>

58731 **CHANGE HISTORY**

58732 First released in Issue 4, Version 2.

58733 **Issue 5**

58734 Moved from X/OPEN UNIX extension to BASE.

58735 The DESCRIPTION is updated to indicate that the saved set-group-ID can be set by any of the
58736 *exec* family of functions, not just *execve()*.

58737 **Issue 7**

58738 SD5-XSH-ERN-177 is applied, adding the ability to set both the effective group ID and saved set-
58739 group-ID to be the same as the real group ID.

58740 **NAME**

58741 setreuid — set real and effective user IDs

58742 **SYNOPSIS**

```
58743 XSI #include <unistd.h>
58744 int setreuid(uid_t ruid, uid_t euid);
```

58745 **DESCRIPTION**

58746 The *setreuid()* function shall set the real and effective user IDs of the current process to the
 58747 values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is -1 , the corresponding effective
 58748 or real user ID of the current process shall be left unchanged.

58749 A process with appropriate privileges can set either ID to any value. An unprivileged process
 58750 can only set the effective user ID if the *euid* argument is equal to either the real, effective, or
 58751 saved user ID of the process.

58752 If the real user ID is being set (*ruid* is not -1), or the effective user ID is being set to a value not
 58753 equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to
 58754 the new effective user ID. +

58755 It is unspecified whether a process without appropriate privileges is permitted to change the real
 58756 user ID to match the current effective user ID or saved set-user-ID of the process. |

58757 **RETURN VALUE**

58758 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 58759 indicate the error.

58760 **ERRORS**

58761 The *setreuid()* function shall fail if:

- | | | |
|-------|----------|---|
| 58762 | [EINVAL] | The value of the <i>ruid</i> or <i>euid</i> argument is invalid or out-of-range. |
| 58763 | [EPERM] | The current process does not have appropriate privileges, and either an
58764 attempt was made to change the effective user ID to a value other than the real
58765 user ID or the saved set-user-ID or an attempt was made to change the real
58766 user ID to a value not permitted by the implementation. |

58767 **EXAMPLES**58768 **Setting the Effective User ID to the Real User ID**

58769 The following example sets the effective user ID of the calling process to the real user ID, so that
 58770 files created later will be owned by the current user. It also sets the saved set-user-ID to the real
 58771 user ID, so any future attempt to set the effective user ID back to its previous value will fail. |

```
58772 #include <unistd.h>
58773 #include <sys/types.h>
58774 ...
58775 setreuid(getuid(), getuid());
58776 ...
```

58777 **APPLICATION USAGE**

58778 None.

58779
58780
58781
58782
58783
58784
58785
58786**RATIONALE**

Earlier versions of this standard did not specify whether the saved set-user-ID was affected by *setreuid()* calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective user ID and saved set-user-ID to be the same as the real user ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective user ID. Privileged applications could already do this using just *setuid()*, but for non-privileged applications the only standard method available is to use this feature of *setreuid()*.

58787
58788**FUTURE DIRECTIONS**

None.

58789
58790**SEE ALSO**

getegid(), *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setuid()*

58791

XBD <[unistd.h](#)>

58792
58793**CHANGE HISTORY**

First released in Issue 4, Version 2.

58794
58795**Issue 5**

Moved from X/OPEN UNIX extension to BASE.

58796
58797**Issue 7**

SD5-XSH-ERN-177 is applied, adding the ability to set both the effective user ID and the saved set-user-ID to be the same as the real user ID.

58798

58799 **NAME**
58800 setrlimit — control maximum resource consumption

58801 **SYNOPSIS**

58802 XSI `#include <sys/resource.h>`
58803 `int setrlimit(int resource, const struct rlimit *rlp);`

58804 **DESCRIPTION**

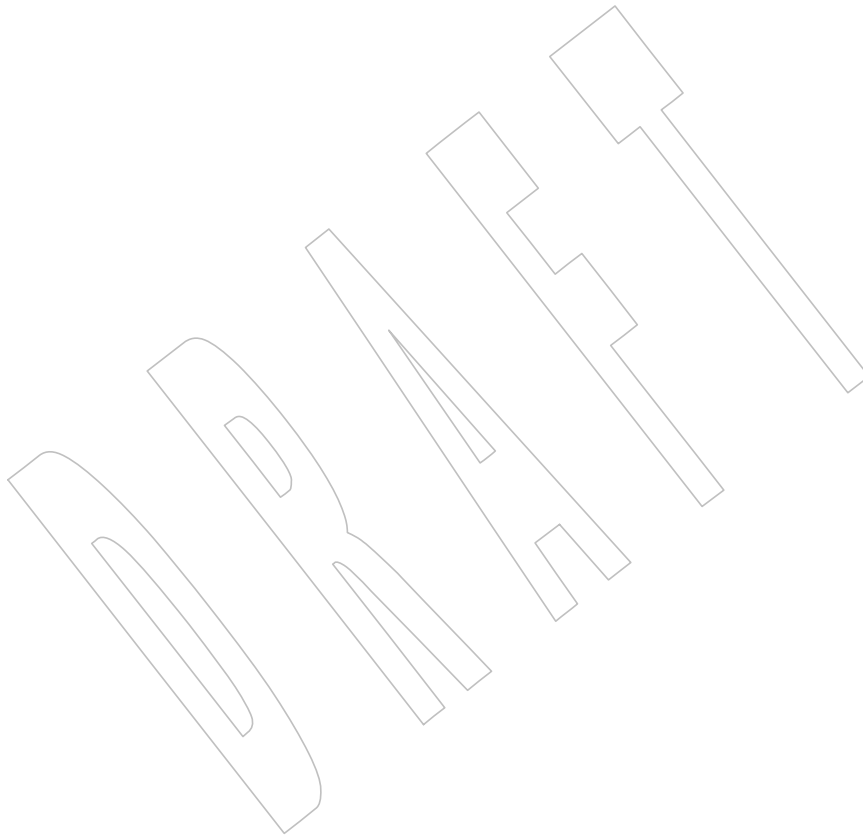
58805 Refer to [getrlimit\(\)](#).

setservent()

58806 **NAME**
58807 setservent — network services database functions

58808 **SYNOPSIS**
58809 #include <netdb.h>
58810 void setservent(int *stayopen*);

58811 **DESCRIPTION**
58812 Refer to *endservent()*.



58813 **NAME**

58814 setsid — create session and set process group ID

58815 **SYNOPSIS**

58816 #include <unistd.h>

58817 pid_t setsid(void);

58818 **DESCRIPTION**

58819 The *setsid()* function shall create a new session, if the calling process is not a process group
 58820 leader. Upon return the calling process shall be the session leader of this new session, shall be
 58821 the process group leader of a new process group, and shall have no controlling terminal. The
 58822 process group ID of the calling process shall be set equal to the process ID of the calling process.
 58823 The calling process shall be the only process in the new process group and the only process in
 58824 the new session.

58825 **RETURN VALUE**

58826 Upon successful completion, *setsid()* shall return the value of the new process group ID of the
 58827 calling process. Otherwise, it shall return (**pid_t**)-1 and set *errno* to indicate the error.

58828 **ERRORS**58829 The *setsid()* function shall fail if:

58830 [EPERM] The calling process is already a process group leader, or the process group ID
 58831 of a process other than the calling process matches the process ID of the
 58832 calling process.

58833 **EXAMPLES**

58834 None.

58835 **APPLICATION USAGE**

58836 None.

58837 **RATIONALE**

58838 The *setsid()* function is similar to the *setpgid()* function of System V. System V, without job
 58839 control, groups processes into process groups and creates new process groups via *setpgid()*; only
 58840 one process group may be part of a login session.

58841 Job control allows multiple process groups within a login session. In order to limit job control
 58842 actions so that they can only affect processes in the same login session, this volume of
 58843 POSIX.1-200x adds the concept of a session that is created via *setsid()*. The *setsid()* function also
 58844 creates the initial process group contained in the session. Additional process groups can be
 58845 created via the *setpgid()* function. A System V process group would correspond to a POSIX
 58846 System Interfaces session containing a single POSIX process group. Note that this function
 58847 requires that the calling process not be a process group leader. The usual way to ensure this is
 58848 true is to create a new process with *fork()* and have it call *setsid()*. The *fork()* function
 58849 guarantees that the process ID of the new process does not match any existing process group ID.

58850 **FUTURE DIRECTIONS**

58851 None.

58852 **SEE ALSO**58853 *getsid()*, *setpgid()*, *setpgid()*

58854 XBD <sys/types.h>, <unistd.h>

58855
58856
58857
58858
58859
58860
58861
58862
58863**CHANGE HISTORY**

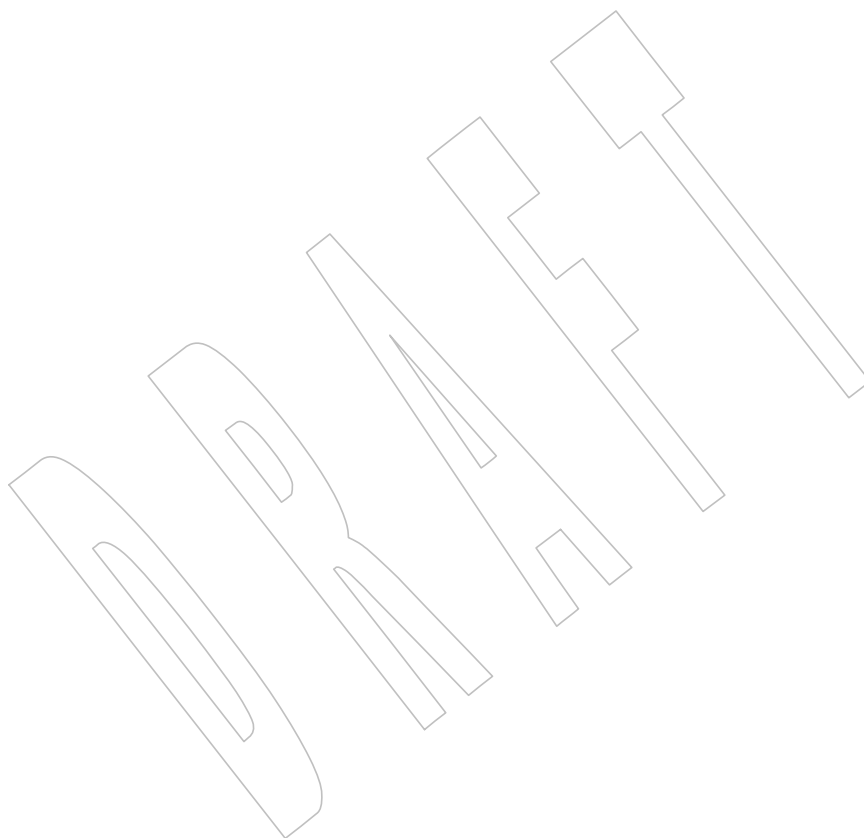
First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.



58864 **NAME**
 58865 setsockopt — set the socket options

58866 **SYNOPSIS**
 58867 #include <sys/socket.h>
 58868 int setsockopt(int socket, int level, int option_name,
 58869 const void *option_value, socklen_t option_len);

58870 **DESCRIPTION**
 58871 The *setsockopt()* function shall set the option specified by the *option_name* argument, at the
 58872 protocol level specified by the *level* argument, to the value pointed to by the *option_value*
 58873 argument for the socket associated with the file descriptor specified by the *socket* argument.

58874 The *level* argument specifies the protocol level at which the option resides. To set options at the
 58875 socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply
 58876 the appropriate *level* identifier for the protocol controlling the option. For example, to indicate
 58877 that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP
 58878 as defined in the **<netinet/in.h>** header.

58879 The *option_name* argument specifies a single option to set. The *option_name* argument and any
 58880 specified options are passed uninterpreted to the appropriate protocol module for
 58881 interpretations. The **<sys/socket.h>** header defines the socket-level options. The options are as
 58882 follows:

58883 SO_DEBUG Turns on recording of debugging information. This option enables or
 58884 disables debugging in the underlying protocol modules. This option takes
 58885 an **int** value. This is a Boolean option.

58886 SO_BROADCAST Permits sending of broadcast messages, if this is supported by the
 58887 protocol. This option takes an **int** value. This is a Boolean option.

58888 SO_REUSEADDR Specifies that the rules used in validating addresses supplied to *bind()*
 58889 should allow reuse of local addresses, if this is supported by the protocol.
 58890 This option takes an **int** value. This is a Boolean option.

58891 SO_KEEPALIVE Keeps connections active by enabling the periodic transmission of
 58892 messages, if this is supported by the protocol. This option takes an **int**
 58893 value.

58894 If the connected socket fails to respond to these messages, the connection
 58895 is broken and threads writing to that socket are notified with a SIGPIPE
 58896 signal. This is a Boolean option.

58897 SO_LINGER Lingers on a *close()* if data is present. This option controls the action taken
 58898 when unsent messages queue on a socket and *close()* is performed. If
 58899 SO_LINGER is set, the system shall block the calling thread during *close()*
 58900 until it can transmit the data or until the time expires. If SO_LINGER is
 58901 not specified, and *close()* is issued, the system handles the call in a way
 58902 that allows the calling thread to continue as quickly as possible. This
 58903 option takes a **linger** structure, as defined in the **<sys/socket.h>** header, to
 58904 specify the state of the option and linger interval.

58905 SO_OOBINLINE Leaves received out-of-band data (data marked urgent) inline. This option
 58906 takes an **int** value. This is a Boolean option.

58907	SO_SNDBUF	Sets send buffer size. This option takes an int value.
58908	SO_RCVBUF	Sets receive buffer size. This option takes an int value.
58909	SO_DONTROUTE	Requests that outgoing messages bypass the standard routing facilities. The destination shall be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an int value. This is a Boolean option.
58910		
58911		
58912		
58913		
58914	SO_RCVLOWAT	Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different from that returned; for example, out-of-band data.) This option takes an int value. Note that not all implementations allow this option to be set.
58915		
58916		
58917		
58918		
58919		
58920		
58921		
58922	SO_RCVTIMEO	Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a timeval structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it shall return with a partial count or <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is received. The default for this option is zero, which indicates that a receive operation shall not time out. This option takes a timeval structure. Note that not all implementations allow this option to be set.
58923		
58924		
58925		
58926		
58927		
58928		
58929		
58930		
58931		
58932	SO_SNDLOWAT	Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations shall process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an int value. Note that not all implementations allow this option to be set.
58933		
58934		
58935		
58936		
58937	SO_SNDTIMEO	Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it shall return with a partial count or with <i>errno</i> set to [EAGAIN] or [EWOULDBLOCK] if no data is sent. The default for this option is zero, which indicates that a send operation shall not time out. This option stores a timeval structure. Note that not all implementations allow this option to be set.
58938		
58939		
58940		
58941		
58942		
58943		
58944		
58945		
58946		
58947		
58948		
58949		
58950		
58951		
58952		

For Boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.

Options at other protocol levels vary in format and name.

RETURN VALUE

Upon successful completion, *setsockopt()* shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setsockopt()* function shall fail if:

[EBADF] The *socket* argument is not a valid file descriptor.

58953	[EDOM]	The send and receive timeout values are too big to fit into the timeout fields in the socket structure.
58954		
58955	[EINVAL]	The specified option is invalid at the specified socket level or the socket has been shut down.
58956		
58957	[EISCONN]	The socket is already connected, and a specified option cannot be set while the socket is connected.
58958		
58959	[ENOPROTOOPT]	
58960		The option is not supported by the protocol.
58961	[ENOTSOCK]	The <i>socket</i> argument does not refer to a socket.
58962		The <i>setsockopt()</i> function may fail if:
58963	[ENOMEM]	There was insufficient memory available for the operation to complete.
58964	[ENOBUFS]	Insufficient resources are available in the system to complete the call.

EXAMPLES

None.

APPLICATION USAGE

The *setsockopt()* function provides an application program with the means to control socket behavior. An application program can use *setsockopt()* to allocate buffer space, control timeouts, or permit socket data broadcasts. The `<sys/socket.h>` header defines the socket-level options available to *setsockopt()*.

Options may exist at multiple protocol levels. The `SO_` options are always present at the uppermost socket level.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10 (on page 496), *bind()*, *endprotoent()*, *getsockopt()*, *socket()*

XBD `<netinet/in.h>`, `<sys/socket.h>`

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/125 is applied, updating the `SO_LINGER` option in the DESCRIPTION to refer to the calling thread rather than the process.

setstate()

58985 **NAME**
58986 setstate — switch pseudo-random number generator state arrays

58987 **SYNOPSIS**
58988 XSI #include <stdlib.h>
58989 char *setstate(const char *state);

58990 **DESCRIPTION**
58991 Refer to *initstate()*.

58992 **NAME**

58993 setuid — set user ID

58994 **SYNOPSIS**58995 #include <unistd.h>
58996 int setuid(uid_t uid);58997 **DESCRIPTION**58998 If the process has appropriate privileges, *setuid()* shall set the real user ID, effective user ID, and
58999 the saved set-user-ID of the calling process to *uid*.59000 If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the
59001 saved set-user-ID, *setuid()* shall set the effective user ID to *uid*; the real user ID and saved set-
59002 user-ID shall remain unchanged.59003 The *setuid()* function shall not affect the supplementary group list in any way.59004 **RETURN VALUE**59005 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
59006 indicate the error.59007 **ERRORS**59008 The *setuid()* function shall fail, return -1, and set *errno* to the corresponding value if one or more
59009 of the following are true:

- 59010 [EINVAL] The value of the
- uid*
- argument is invalid and not supported by the
-
- 59011 implementation.
-
- 59012 [EPERM] The process does not have appropriate privileges and
- uid*
- does not match the
-
- 59013 real user ID or the saved set-user-ID.

59014 **EXAMPLES**

59015 None.

59016 **APPLICATION USAGE**

59017 None.

59018 **RATIONALE**59019 The various behaviors of the *setuid()* and *setgid()* functions when called by non-privileged
59020 processes reflect the behavior of different historical implementations. For portability, it is
59021 recommended that new non-privileged applications use the *seteuid()* and *setegid()* functions
59022 instead.59023 The saved set-user-ID capability allows a program to regain the effective user ID established at
59024 the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the
59025 effective group ID established at the last *exec* call. These capabilities are derived from System V.
59026 Without them, a program might have to run as superuser in order to perform the same
59027 functions, because superuser can write on the user's files. This is a problem because such a
59028 program can write on any user's files, and so must be carefully written to emulate the
59029 permissions of the calling process properly. In System V, these capabilities have traditionally
59030 been implemented only via the *setuid()* and *setgid()* functions for non-privileged processes. The
59031 fact that the behavior of those functions was different for privileged processes made them
59032 difficult to use. The POSIX.1-1990 standard defined the *setuid()* function to behave differently
59033 for privileged and unprivileged users. When the caller had the appropriate privilege, the
59034 function set the real user ID, effective user ID, and saved set-user ID of the calling process on
59035 implementations that supported it. When the caller did not have the appropriate privilege, the
59036 function set only the effective user ID, subject to permission checks. The former use is generally

59037
59038
59039
59040

needed for utilities like *login* and *su*, which are not conforming applications and thus outside the scope of POSIX.1-200x. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

59041
59042
59043
59044
59045
59046
59047
59048
59049
59050

POSIX.1-200x augments the latter functionality with a mandatory feature named `_POSIX_SAVED_IDS`. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with the (implementation-defined) appropriate privilege. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-200x. However, there are implementors who have been reluctant to support it given the limitation described above.

59051
59052
59053
59054
59055
59056
59057
59058
59059
59060

The 4.3BSD system handles the problem by supporting separate functions: *setuid()* (which always sets both the real and effective user IDs, like *setuid()* in POSIX.1-200x for privileged users), and *seteuid()* (which always sets just the effective user ID, like *setuid()* in POSIX.1-200x for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid()*, which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid()* and *setegid()* as described above. The *setreuid()* and *setregid()* functions will be deprecated or removed.

59061

The solution here is:

59062
59063
59064
59065

- Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid()*.
- Add the *seteuid()* and *setegid()* functions as portable alternatives to *setuid()* and *setgid()* for non-privileged and privileged processes.

59066
59067
59068
59069
59070
59071
59072
59073
59074
59075
59076
59077
59078
59079
59080
59081
59082
59083
59084
59085
59086
59087

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid()* function in the presence of a saved set-user-ID, or the use of the BSD *setreuid()* function, which was able to swap the real and effective user IDs. The changes included in POSIX.1-200x provide a new mechanism using *seteuid()* in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid()* mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by `_POSIX_SAVED_IDS` has been changed to agree with the case where the option was defined. The *kill()* function is an exception. Implementors of the new *seteuid()* mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid()* and with the `_POSIX_SAVED_IDS` behavior of *kill()* is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid()* to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill()* behavior of `_POSIX_SAVED_IDS` was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid()*. The standard developers considered that alternative to be less correct than the retention of the old behavior of *kill()* in such systems. Current conforming applications shall accommodate either behavior from *kill()*, and there appears to be no strong reason for *kill()* to

59088 check the saved set-user-ID rather than the effective user ID.

59089 FUTURE DIRECTIONS

59090 None.

59091 SEE ALSO

59092 *exec*, *getegid()*, *geteuid()*, *getgid()*, *getuid()*, *setegid()*, *seteuid()*, *setgid()*, *setregid()*, *setreuid()*

59093 XBD [<sys/types.h>](#), [<unistd.h>](#)

59094 CHANGE HISTORY

59095 First released in Issue 1. Derived from Issue 1 of the SVID.

59096 Issue 6

59097 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.

59098 The following new requirements on POSIX implementations derive from alignment with the
59099 Single UNIX Specification:

- 59100 • The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
59101 required for conforming implementations of previous POSIX specifications, it was not
59102 required for UNIX applications.
- 59103 • The functionality associated with `_POSIX_SAVED_IDS` is now mandatory. This is a FIPS
59104 requirement.

59105 The following changes were made to align with the IEEE P1003.1a draft standard:

- 59106 • The effects of *setuid()* in processes without appropriate privileges are changed.
- 59107 • A requirement that the supplementary group list is not affected is added.

59108 **NAME**
59109 setutxent — reset the user accounting database to the first entry

59110 **SYNOPSIS**

59111 XSI #include <utmpx.h>
59112 void setutxent(void);

59113 **DESCRIPTION**

59114 Refer to *endutxent()*.

59115 **NAME**

59116 setvbuf — assign buffering to a stream

59117 **SYNOPSIS**

59118 #include <stdio.h>

59119 int setvbuf(FILE *restrict stream, char *restrict buf, int type,
59120 size_t size);59121 **DESCRIPTION**59122 CX The functionality described on this reference page is aligned with the ISO C standard. Any
59123 conflict between the requirements described here and the ISO C standard is unintentional. This
59124 volume of POSIX.1-200x defers to the ISO C standard.59125 The *setvbuf()* function may be used after the stream pointed to by *stream* is associated with an
59126 open file but before any other operation (other than an unsuccessful call to *setvbuf()*) is
59127 performed on the stream. The argument *type* determines how *stream* shall be buffered, as
59128 follows:

- 59129
- {_IOFBF} shall cause input/output to be fully buffered.
59130 - {_IOLBF} shall cause input/output to be line buffered.
59131 - {_IONBF} shall cause input/output to be unbuffered.

59132 If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by
59133 *setvbuf()* and the argument *size* specifies the size of the array; otherwise, *size* may determine the
59134 size of a buffer allocated by the *setvbuf()* function. The contents of the array at any time are
59135 unspecified.59136 For information about streams, see [Section 2.5](#) (on page 469).59137 **RETURN VALUE**59138 Upon successful completion, *setvbuf()* shall return 0. Otherwise, it shall return a non-zero value
59139 CX if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to
59140 indicate the error.59141 **ERRORS**59142 The *setvbuf()* function may fail if:59143 CX [EBADF] The file descriptor underlying *stream* is not valid.59144 **EXAMPLES**

59145 None.

59146 **APPLICATION USAGE**59147 A common source of error is allocating buffer space as an “automatic” variable in a code block,
59148 and then failing to close the stream in the same block.59149 With *setvbuf()*, allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are
59150 used for the buffer area.59151 Applications should note that many implementations only provide line buffering on input from
59152 terminal devices.59153 **RATIONALE**

59154 None.

setvbuf()

59155

FUTURE DIRECTIONS

59156

None.

59157

SEE ALSO

59158

[Section 2.5](#) (on page 469), [fopen\(\)](#), [setbuf\(\)](#)

59159

XBD [<stdio.h>](#)

59160

CHANGE HISTORY

59161

First released in Issue 1. Derived from Issue 1 of the SVID.

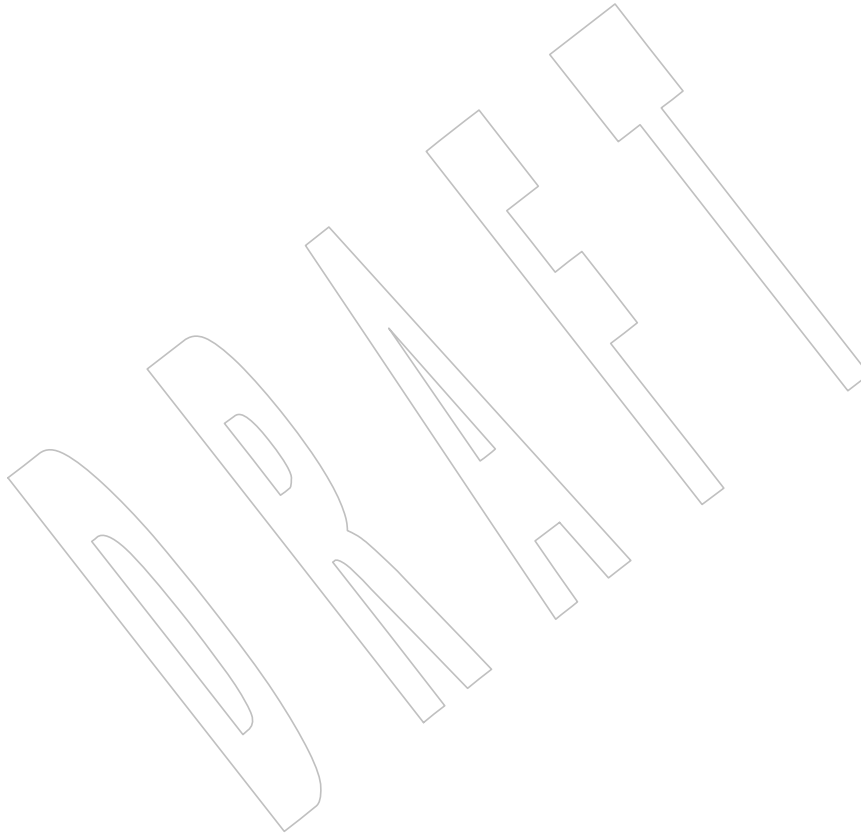
59162

Issue 6

59163

Extensions beyond the ISO C standard are marked.

59164

The *setvbuf()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

59165 **NAME**59166 shm_open — open a shared memory object (**REALTIME**)59167 **SYNOPSIS**

```
59168 SHM #include <sys/mman.h>
59169 int shm_open(const char *name, int oflag, mode_t mode);
```

59170 **DESCRIPTION**

59171 The *shm_open()* function shall establish a connection between a shared memory object and a file
 59172 descriptor. It shall create an open file description that refers to the shared memory object and a
 59173 file descriptor that refers to that open file description. The file descriptor is used by other
 59174 functions to refer to that shared memory object. The *name* argument points to a string naming a
 59175 shared memory object. It is unspecified whether the name appears in the file system and is
 59176 visible to other functions that take pathnames as arguments. The *name* argument conforms to the
 59177 construction rules for a pathname, except that the interpretation of slash characters other than
 59178 the leading slash character in *name* is implementation-defined, and that the length limits for the
 59179 *name* argument are implementation-defined and need not be the same as the pathname limits
 59180 {PATH_MAX} and {NAME_MAX}. If *name* begins with the slash character, then processes
 59181 calling *shm_open()* with the same value of *name* refer to the same shared memory object, as long
 59182 as that name has not been removed. If *name* does not begin with the slash character, the effect is
 59183 implementation-defined.

59184 If successful, *shm_open()* shall return a file descriptor for the shared memory object that is the
 59185 lowest numbered file descriptor not currently open for that process. The open file description is
 59186 new, and therefore the file descriptor does not share it with any other processes. It is unspecified
 59187 whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file
 59188 descriptor is set.

59189 The file status flags and file access modes of the open file description are according to the value
 59190 of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the
 59191 **<fcntl.h>** header. Applications specify exactly one of the first two values (access modes) below
 59192 in the value of *oflag*:

59193 O_RDONLY Open for read access only.

59194 O_RDWR Open for read or write access.

59195 Any combination of the remaining flags may be specified in the value of *oflag*:

59196 O_CREAT If the shared memory object exists, this flag has no effect, except as noted |
 59197 under O_EXCL below. Otherwise, the shared memory object is created. The |
 59198 user ID of the shared memory object shall be set to the effective user ID of the |
 59199 process. The group ID of the shared memory object shall be set to the effective |
 59200 group ID of the process; however, if the *name* argument is visible in the file |
 59201 system, the group ID may be set to the group ID of the containing directory. |
 59202 The permission bits of the shared memory object shall be set to the value of |
 59203 the *mode* argument except those set in the file mode creation mask of the |
 59204 process. When bits in *mode* other than the file permission bits are set, the effect |
 59205 is unspecified. The *mode* argument does not affect whether the shared memory |
 59206 object is opened for reading, for writing, or for both. The shared memory |
 59207 object has a size of zero.

59208 O_EXCL If O_EXCL and O_CREAT are set, *shm_open()* fails if the shared memory
 59209 object exists. The check for the existence of the shared memory object and the
 59210 creation of the object if it does not exist is atomic with respect to other

shm_open()

59211 processes executing *shm_open()* naming the same shared memory object with
 59212 O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the
 59213 result is undefined.

59214 O_TRUNC If the shared memory object exists, and it is successfully opened O_RDWR, the
 59215 object shall be truncated to zero length and the mode and owner shall be
 59216 unchanged by this function call. The result of using O_TRUNC with
 59217 O_RDONLY is undefined.

59218 When a shared memory object is created, the state of the shared memory object, including all
 59219 data associated with the shared memory object, persists until the shared memory object is
 59220 unlinked and all other references are gone. It is unspecified whether the name and shared
 59221 memory object state remain valid after a system reboot.

RETURN VALUE

59222 Upon successful completion, the *shm_open()* function shall return a non-negative integer
 59223 representing the lowest numbered unused file descriptor. Otherwise, it shall return -1 and set
 59224 *errno* to indicate the error.

ERRORS

59226 The *shm_open()* function shall fail if:

59228 [EACCES] The shared memory object exists and the permissions specified by *oflag* are
 59229 denied, or the shared memory object does not exist and permission to create
 59230 the shared memory object is denied, or O_TRUNC is specified and write
 59231 permission is denied.

59232 [EEXIST] O_CREAT and O_EXCL are set and the named shared memory object already
 59233 exists.

59234 [EINTR] The *shm_open()* operation was interrupted by a signal.

59235 [EINVAL] The *shm_open()* operation is not supported for the given name.

59236 [EMFILE] All file descriptors available to the process are currently open.

59237 [ENFILE] Too many shared memory objects are currently open in the system.

59238 [ENOENT] O_CREAT is not set and the named shared memory object does not exist.

59239 [ENOSPC] There is insufficient space for the creation of the new shared memory object.

59240 The *shm_open()* function may fail if:

59241 [ENAMETOOLONG]

59242 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
 59243 XSI that do not support the XSI option or exceeds `{_XOPEN_PATH_MAX}` on XSI
 59244 systems, or has a pathname component that is longer than
 59245 XSI `{_POSIX_NAME_MAX}` on systems that do not support the XSI option or
 59246 longer than `{_XOPEN_NAME_MAX}` on XSI systems.

EXAMPLES**Creating and Mapping a Shared Memory Object**

59248 The following code segment demonstrates the use of *shm_open()* to create a shared memory
 59249 object which is then sized using *ftruncate()* before being mapped into the process address space
 59250 using *mmap()*:

```
59252 #include <unistd.h>
59253 #include <sys/mman.h>
59254 ...
```

```

59255     #define MAX_LEN 10000
59256     struct region {           /* Defines "structure" of shared memory */
59257         int len;
59258         char buf[MAX_LEN];
59259     };
59260     struct region *rptr;
59261     int fd;

59262     /* Create shared memory object and set its size */

59263     fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
59264     if (fd == -1)
59265         /* Handle error */;

59266     if (ftruncate(fd, sizeof(struct region)) == -1)
59267         /* Handle error */;

59268     /* Map shared memory object */

59269     rptr = mmap(NULL, sizeof(struct region),
59270                PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
59271     if (rptr == MAP_FAILED)
59272         /* Handle error */;

59273     /* Now we can refer to mapped region using fields of rptr;
59274        for example, rptr->len */
59275     ...

```

APPLICATION USAGE

None.

RATIONALE

When the Memory Mapped Files option is supported, the normal *open()* call is used to obtain a descriptor to a file to be mapped according to existing practice with *mmap()*. When the Shared Memory Objects option is supported, the *shm_open()* function shall obtain a descriptor to the shared memory object to be mapped.

There is ample precedent for having a file descriptor represent several types of objects. In the POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory. Many implementations simply have an operations vector, which is indexed by the file descriptor type and does very different operations. Note that in some cases the file descriptor passed to generic operations on file descriptors is returned by *open()* or *creat()* and in some cases returned by alternate functions, such as *pipe()*. The latter technique is used by *shm_open()*.

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using *ftruncate()*. The *shm_open()* function itself does not create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open()* function may be implemented using a macro that invokes *open()*, and the *shm_unlink()* function may be implemented using a macro that invokes *unlink()*.

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open()* followed by an *ftruncate()* and *close()* can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual

59303 memory hardware support in order to ensure that the memory is contiguous.

59304 The set of valid open flags to *shm_open()* was restricted to *O_RDONLY*, *O_RDWR*, *O_CREAT*,
59305 and *O_TRUNC* because these could be easily implemented on most memory mapping systems.
59306 This volume of POSIX.1-200x is silent on the results if the implementation cannot supply the
59307 requested file access because of implementation-defined reasons, including hardware ones.

59308 The error conditions [EACCES] and [ENOTSUP] are provided to inform the application that the
59309 implementation cannot complete a request.

59310 [EACCES] indicates for implementation-defined reasons, probably hardware-related, that the
59311 implementation cannot comply with a requested mode because it conflicts with another
59312 requested mode. An example might be that an application desires to open a memory object two
59313 times, mapping different areas with different access modes. If the implementation cannot map a
59314 single area into a process space in two places, which would be required if different access modes
59315 were required for the two areas, then the implementation may inform the application at the time
59316 of the second open.

59317 [ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the
59318 implementation cannot comply with a requested mode at all. An example would be that the
59319 hardware of the implementation cannot support write-only shared memory areas.

59320 On all implementations, it may be desirable to restrict the location of the memory objects to
59321 specific file systems for performance (such as a RAM disk) or implementation-defined reasons
59322 (shared memory supported directly only on certain file systems). The *shm_open()* function may
59323 be used to enforce these restrictions. There are a number of methods available to the application
59324 to determine an appropriate name of the file or the location of an appropriate directory. One way
59325 is from the environment via *getenv()*. Another would be from a configuration file.

59326 This volume of POSIX.1-200x specifies that memory objects have initial contents of zero when
59327 created. This is consistent with current behavior for both files and newly allocated memory. For
59328 those implementations that use physical memory, it would be possible that such
59329 implementations could simply use available memory and give it to the process uninitialized.
59330 This, however, is not consistent with standard behavior for the uninitialized data area, the stack,
59331 and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security
59332 reasons. Thus, initializing memory objects to zero is required.

59333 FUTURE DIRECTIONS

59334 None.

59335 SEE ALSO

59336 *close()*, *dup()*, *exec*, *fcntl()*, *mmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_unlink()*, *umask*

59337 XBD <fcntl.h>, <sys/mman.h>

59338 CHANGE HISTORY

59339 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

59340 Issue 6

59341 The *shm_open()* function is marked as part of the Shared Memory Objects option.

59342 The [ENOSYS] error condition has been removed as stubs need not be provided if an
59343 implementation does not support the Shared Memory Objects option.

59344 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/126 is applied, adding the example to the
59345 EXAMPLES section.

59346

Issue 7

59347

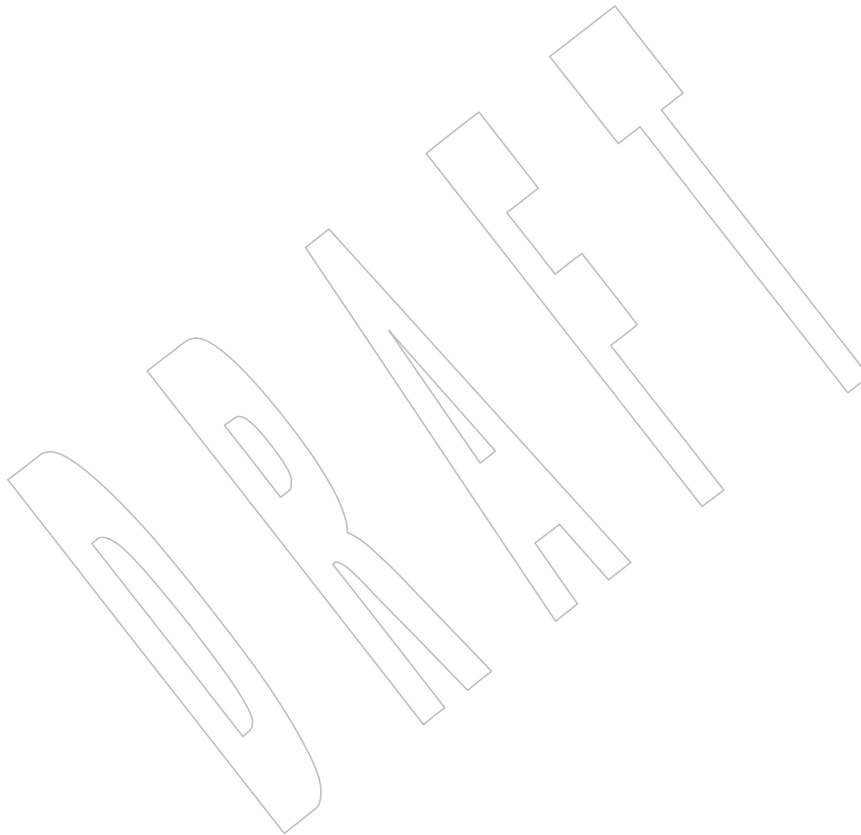
Austin Group Interpretation 1003.1-2001 #077 is applied, clarifying the *name* argument and changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

59348

59349

SD5-XSH-ERN-170 is applied, updating the DESCRIPTION to clarify the wording for setting the user ID and group ID of the shared memory object. +

59350



59351 **NAME**59352 shm_unlink — remove a shared memory object (**REALTIME**)59353 **SYNOPSIS**

```
59354 SHM #include <sys/mman.h>
59355 int shm_unlink(const char *name);
```

59356 **DESCRIPTION**

59357 The *shm_unlink()* function shall remove the name of the shared memory object named by the
59358 string pointed to by *name*.

59359 If one or more references to the shared memory object exist when the object is unlinked, the
59360 name shall be removed before *shm_unlink()* returns, but the removal of the memory object
59361 contents shall be postponed until all open and map references to the shared memory object have
59362 been removed.

59363 Even if the object continues to exist after the last *shm_unlink()*, reuse of the name shall
59364 subsequently cause *shm_open()* to behave as if no shared memory object of this name exists (that
59365 is, *shm_open()* will fail if `O_CREAT` is not set, or will create a new shared memory object if
59366 `O_CREAT` is set).

59367 **RETURN VALUE**

59368 Upon successful completion, a value of zero shall be returned. Otherwise, a value of `-1` shall be
59369 returned and *errno* set to indicate the error. If `-1` is returned, the named shared memory object
59370 shall not be changed by this function call.

59371 **ERRORS**59372 The *shm_unlink()* function shall fail if:

59373 [EACCES] Permission is denied to unlink the named shared memory object.

59374 [ENOENT] The named shared memory object does not exist.

59375 The *shm_unlink()* function may fail if:

59376 [ENAMETOOLONG]

59377 The length of the *name* argument exceeds `{_POSIX_PATH_MAX}` on systems
59378 that do not support the XSI option `or exceeds {_XOPEN_PATH_MAX}` on XSI
59379 systems, `or has a pathname component that is longer than`
59380 `{_POSIX_NAME_MAX}` on systems that do not support the XSI option `or`
59381 `longer than {_XOPEN_NAME_MAX}` on XSI systems. A call to *shm_unlink()*
59382 with a *name* argument that contains the same shared memory object name as
59383 was previously used in a successful *shm_open()* call shall not give an
59384 [ENAMETOOLONG] error.

59385 **EXAMPLES**

59386 None.

59387 **APPLICATION USAGE**

59388 Names of memory objects that were allocated with *open()* are deleted with *unlink()* in the usual
59389 fashion. Names of memory objects that were allocated with *shm_open()* are deleted with
59390 *shm_unlink()*. Note that the actual memory object is not destroyed until the last close and
59391 unmap on it have occurred if it was already in use.

59392
59393
59394
59395
59396
59397
59398
59399
59400
59401
59402
59403
59404
59405
59406
59407
59408
59409

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

close(), *mmap()*, *munmap()*, *shmat()*, *shmctl()*, *shmdt()*, *shm_open()*

XBD <sys/mman.h>

CHANGE HISTORY

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *shm_unlink()* function is marked as part of the Shared Memory Objects option.

In the DESCRIPTION, text is added to clarify that reusing the same name after a *shm_unlink()* will not attach to the old shared memory object.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Shared Memory Objects option.

Issue 7

Austin Group Interpretation 1003.1-2001 #077 is applied, changing [ENAMETOOLONG] from a “shall fail” to a “may fail” error.

DRAFT

59410 **NAME**

59411 shmat — XSI shared memory attach operation

59412 **SYNOPSIS**

```
59413 XSI #include <sys/shm.h>
59414 void *shmat(int shmid, const void *shmaddr, int shmflg);
```

59415 **DESCRIPTION**

59416 The *shmat()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 80). It is
 59417 unspecified whether this function interoperates with the realtime interprocess communication
 59418 facilities defined in [Section 2.8](#) (on page 476).

59419 The *shmat()* function attaches the shared memory segment associated with the shared memory
 59420 identifier specified by *shmid* to the address space of the calling process. The segment is attached
 59421 at the address specified by one of the following criteria:

- 59422 • If *shmaddr* is a null pointer, the segment is attached at the first available address as selected
 59423 by the system.
- 59424 • If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is non-zero, the segment is
 59425 attached at the address given by (*shmaddr* - ((*uintptr_t*)*shmaddr* %SHMLBA)). The character
 59426 ‘%’ is the C-language remainder operator.
- 59427 • If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is 0, the segment is attached at
 59428 the address given by *shmaddr*.
- 59429 • The segment is attached for reading if (*shmflg* &SHM_RDONLY) is non-zero and the
 59430 calling process has read permission; otherwise, if it is 0 and the calling process has read
 59431 and write permission, the segment is attached for reading and writing.

59432 **RETURN VALUE**

59433 Upon successful completion, *shmat()* shall increment the value of *shm_nattch* in the data
 59434 structure associated with the shared memory ID of the attached shared memory segment and
 59435 return the segment’s start address.

59436 Otherwise, the shared memory segment shall not be attached, *shmat()* shall return -1, and *errno*
 59437 shall be set to indicate the error.

59438 **ERRORS**

59439 The *shmat()* function shall fail if:

- | | | |
|-------|----------|--|
| 59440 | [EACCES] | Operation permission is denied to the calling process; see Section 2.7 (on page 474). |
| 59442 | [EINVAL] | The value of <i>shmid</i> is not a valid shared memory identifier, the <i>shmaddr</i> is not a null pointer, and the value of (<i>shmaddr</i> - ((<i>uintptr_t</i>) <i>shmaddr</i> %SHMLBA)) is an illegal address for attaching shared memory; or the <i>shmaddr</i> is not a null pointer, (<i>shmflg</i> &SHM_RND) is 0, and the value of <i>shmaddr</i> is an illegal address for attaching shared memory. |
| 59447 | [EMFILE] | The number of shared memory segments attached to the calling process would exceed the system-imposed limit. |
| 59449 | [ENOMEM] | The available data space is not large enough to accommodate the shared memory segment. |

59451

EXAMPLES

59452

None.

59453

APPLICATION USAGE

59454

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

59455

59456

59457

59458

RATIONALE

59459

None.

59460

FUTURE DIRECTIONS

59461

None.

59462

SEE ALSO

59463

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), *exec*, *exit()*, *fork()*, *shmctl()*, *shmdt()*, *shmget()*, *shm_open()*, *shm_unlink()*

59464

59465

XBD [Section 3.340](#) (on page 80), [<sys/shm.h>](#)

59466

CHANGE HISTORY

59467

First released in Issue 2. Derived from Issue 2 of the SVID.

59468

Issue 5

59469

Moved from SHARED MEMORY to BASE.

59470

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

59471

59472

Issue 6

59473

The Open Group Corrigendum U021/13 is applied.

59474 **NAME**
 59475 shmctl — XSI shared memory control operations

59476 **SYNOPSIS**

```
59477 XSI #include <sys/shm.h>
59478 int shmctl(int shmid, int cmd, struct shm_id_ds *buf);
```

59479 **DESCRIPTION**

59480 The *shmctl()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 80). It is
 59481 unspecified whether this function interoperates with the realtime interprocess communication
 59482 facilities defined in [Section 2.8](#) (on page 476).

59483 The *shmctl()* function provides a variety of shared memory control operations as specified by
 59484 *cmd*. The following values for *cmd* are available:

59485 **IPC_STAT** Place the current value of each member of the **shm_id_ds** data structure
 59486 associated with *shmid* into the structure pointed to by *buf*. The contents of the
 59487 structure are defined in **<sys/shm.h>**.

59488 **IPC_SET** Set the value of the following members of the **shm_id_ds** data structure
 59489 associated with *shmid* to the corresponding value found in the structure
 59490 pointed to by *buf*:

59491 shm_perm.uid
 59492 shm_perm.gid
 59493 shm_perm.mode Low-order nine bits.

59494 IPC_SET can only be executed by a process that has an effective user ID equal
 59495 to either that of a process with appropriate privileges or to the value of
 59496 *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated with
 59497 *shmid*.

59498 **IPC_RMID** Remove the shared memory identifier specified by *shmid* from the system and
 59499 destroy the shared memory segment and **shm_id_ds** data structure associated
 59500 with it. IPC_RMID can only be executed by a process that has an effective user
 59501 ID equal to either that of a process with appropriate privileges or to the value
 59502 of *shm_perm.cuid* or *shm_perm.uid* in the **shm_id_ds** data structure associated
 59503 with *shmid*.

59504 **RETURN VALUE**

59505 Upon successful completion, *shmctl()* shall return 0; otherwise, it shall return -1 and set *errno* to
 59506 indicate the error.

59507 **ERRORS**

59508 The *shmctl()* function shall fail if:

59509 [EACCES] The argument *cmd* is equal to IPC_STAT and the calling process does not have
 59510 read permission; see [Section 2.7](#) (on page 474).

59511 [EINVAL] The value of *shmid* is not a valid shared memory identifier, or the value of *cmd*
 59512 is not a valid command.

59513 [EPERM] The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID
 59514 of the calling process is not equal to that of a process with appropriate
 59515 privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in
 59516 the data structure associated with *shmid*.

59517 The *shmctl()* function may fail if:

59518 [EOVERFLOW] The *cmd* argument is `IPC_STAT` and the *gid* or *uid* value is too large to be
59519 stored in the structure pointed to by the *buf* argument.

59520 EXAMPLES

59521 None.

59522 APPLICATION USAGE

59523 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
59524 Application developers who need to use IPC should design their applications so that modules
59525 using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the
59526 alternative interfaces.

59527 RATIONALE

59528 None.

59529 FUTURE DIRECTIONS

59530 None.

59531 SEE ALSO

59532 [Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), *shmat()*, *shmdt()*, *shmget()*, *shm_open()*,
59533 *shm_unlink()*

59534 XBD [Section 3.340](#) (on page 80), [<sys/shm.h>](#)

59535 CHANGE HISTORY

59536 First released in Issue 2. Derived from Issue 2 of the SVID.

59537 Issue 5

59538 Moved from SHARED MEMORY to BASE.

59539 The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE
59540 DIRECTIONS to a new APPLICATION USAGE section.

59541 **NAME**
 59542 `shmdt` — XSI shared memory detach operation

59543 **SYNOPSIS**

```
59544 XSI #include <sys/shm.h>
59545 int shmdt(const void *shmaddr);
```

59546 **DESCRIPTION**

59547 The `shmdt()` function operates on XSI shared memory (see XBD [Section 3.340](#), on page 80). It is
 59548 unspecified whether this function interoperates with the realtime interprocess communication
 59549 facilities defined in [Section 2.8](#) (on page 476).

59550 The `shmdt()` function detaches the shared memory segment located at the address specified by
 59551 `shmaddr` from the address space of the calling process.

59552 **RETURN VALUE**

59553 Upon successful completion, `shmdt()` shall decrement the value of `shm_nattch` in the data
 59554 structure associated with the shared memory ID of the attached shared memory segment and
 59555 return 0.

59556 Otherwise, the shared memory segment shall not be detached, `shmdt()` shall return `-1`, and `errno`
 59557 shall be set to indicate the error.

59558 **ERRORS**

59559 The `shmdt()` function shall fail if:

59560 [EINVAL] The value of `shmaddr` is not the data segment start address of a shared memory
 59561 segment.

59562 **EXAMPLES**

59563 None.

59564 **APPLICATION USAGE**

59565 The POSIX Realtime Extension defines alternative interfaces for interprocess communication.
 59566 Application developers who need to use IPC should design their applications so that modules
 59567 using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the
 59568 alternative interfaces.

59569 **RATIONALE**

59570 None.

59571 **FUTURE DIRECTIONS**

59572 None.

59573 **SEE ALSO**

59574 [Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), `exec`, `exit()`, `fork()`, `shmat()`, `shmctl()`,
 59575 `shmget()`, `shm_open()`, `shm_unlink()`

59576 XBD [Section 3.340](#) (on page 80), `<sys/shm.h>`

59577 **CHANGE HISTORY**

59578 First released in Issue 2. Derived from Issue 2 of the SVID.

59579

Issue 5

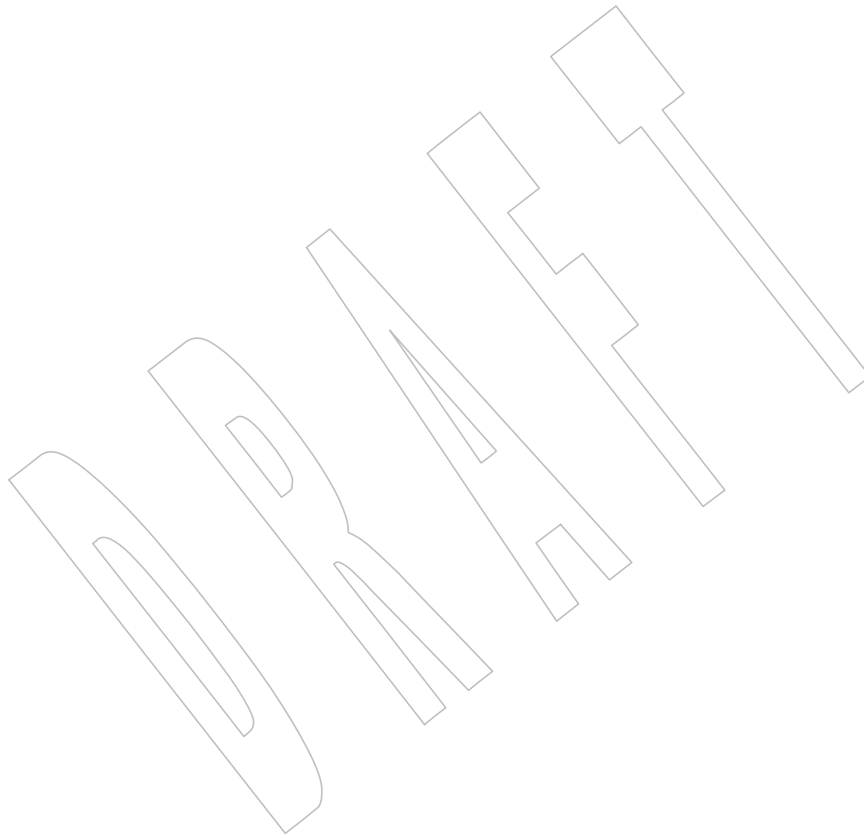
59580

Moved from SHARED MEMORY to BASE.

59581

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

59582



59583 **NAME**
 59584 shmget — get an XSI shared memory segment

59585 **SYNOPSIS**

```
59586 XSI #include <sys/shm.h>
59587 int shmget(key_t key, size_t size, int shmflg);
```

59588 **DESCRIPTION**

59589 The *shmget()* function operates on XSI shared memory (see XBD [Section 3.340](#), on page 80). It is
 59590 unspecified whether this function interoperates with the realtime interprocess communication
 59591 facilities defined in [Section 2.8](#) (on page 476).

59592 The *shmget()* function shall return the shared memory identifier associated with *key*.

59593 A shared memory identifier, associated data structure, and shared memory segment of at least
 59594 *size* bytes (see [<sys/shm.h>](#)) are created for *key* if one of the following is true:

- 59595 • The argument *key* is equal to `IPC_PRIVATE`.
- 59596 • The argument *key* does not already have a shared memory identifier associated with it and
 59597 (*shmflg* & `IPC_CREAT`) is non-zero.

59598 Upon creation, the data structure associated with the new shared memory identifier shall be
 59599 initialized as follows:

- 59600 • The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set equal
 59601 to the effective user ID and effective group ID, respectively, of the calling process.
- 59602 • The low-order nine bits of *shm_perm.mode* are set equal to the low-order nine bits of *shmflg*.
- 59603 • The value of *shm_segsz* is set equal to the value of *size*.
- 59604 • The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set equal to 0.
- 59605 • The value of *shm_ctime* is set equal to the current time.

59606 When the shared memory segment is created, it shall be initialized with all zero values.

59607 **RETURN VALUE**

59608 Upon successful completion, *shmget()* shall return a non-negative integer, namely a shared
 59609 memory identifier; otherwise, it shall return `-1` and set *errno* to indicate the error.

59610 **ERRORS**

59611 The *shmget()* function shall fail if:

- | | | |
|-------|----------|---|
| 59612 | [EACCES] | A shared memory identifier exists for <i>key</i> but operation permission as specified by the low-order nine bits of <i>shmflg</i> would not be granted; see Section 2.7 (on page 474). |
| 59613 | | |
| 59614 | | |
| 59615 | [EEXIST] | A shared memory identifier exists for the argument <i>key</i> but (<i>shmflg</i> & <code>IPC_CREAT</code>) && (<i>shmflg</i> & <code>IPC_EXCL</code>) is non-zero. |
| 59616 | | |
| 59617 | [EINVAL] | A shared memory segment is to be created and the value of <i>size</i> is less than the system-imposed minimum or greater than the system-imposed maximum. |
| 59618 | | |
| 59619 | [EINVAL] | No shared memory segment is to be created and a shared memory segment exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not 0. |
| 59620 | | |
| 59621 | | |

59622	[ENOENT]	A shared memory identifier does not exist for the argument <i>key</i> and (<i>shmflg</i> &IPC_CREAT) is 0.
59623		
59624	[ENOMEM]	A shared memory identifier and associated shared memory segment shall be created, but the amount of available physical memory is not sufficient to fill the request.
59625		
59626		
59627	[ENOSPC]	A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.
59628		
59629		

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in [Section 2.7](#) (on page 474) can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.7](#) (on page 474), [Section 2.8](#) (on page 476), [shmatt\(\)](#), [shmctl\(\)](#), [shmdt\(\)](#), [shm_open\(\)](#), [shm_unlink\(\)](#)

XBD [Section 3.340](#) (on page 80), [<sys/shm.h>](#)

CHANGE HISTORY

First released in Issue 2. Derived from Issue 2 of the SVID.

Issue 5

Moved from SHARED MEMORY to BASE.

The note about use of POSIX Realtime Extension IPC routines has been moved from FUTURE DIRECTIONS to a new APPLICATION USAGE section.

59651 **NAME**

59652 shutdown — shut down socket send and receive operations

59653 **SYNOPSIS**

59654 #include <sys/socket.h>

59655 int shutdown(int *socket*, int *how*);59656 **DESCRIPTION**59657 The *shutdown()* function shall cause all or part of a full-duplex connection on the socket
59658 associated with the file descriptor *socket* to be shut down.59659 The *shutdown()* function takes the following arguments:59660 *socket* Specifies the file descriptor of the socket.59661 *how* Specifies the type of shutdown. The values are as follows:

59662 SHUT_RD Disables further receive operations.

59663 SHUT_WR Disables further send operations.

59664 SHUT_RDWR Disables further send and receive operations.

59665 The *shutdown()* function disables subsequent send and/or receive operations on a socket,
59666 depending on the value of the *how* argument.59667 **RETURN VALUE**59668 Upon successful completion, *shutdown()* shall return 0; otherwise, -1 shall be returned and *errno*
59669 set to indicate the error.59670 **ERRORS**59671 The *shutdown()* function shall fail if:59672 [EBADF] The *socket* argument is not a valid file descriptor.59673 [EINVAL] The *how* argument is invalid.

59674 [ENOTCONN] The socket is not connected.

59675 [ENOTSOCK] The *socket* argument does not refer to a socket.59676 The *shutdown()* function may fail if:

59677 [ENOBUFS] Insufficient resources were available in the system to perform the operation.

59678 **EXAMPLES**

59679 None.

59680 **APPLICATION USAGE**

59681 None.

59682 **RATIONALE**

59683 None.

59684 **FUTURE DIRECTIONS**

59685 None.

59686 **SEE ALSO**59687 *getsockopt()*, *pselect()*, *read*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendto()*, *setsockopt()*, *socket()*,
59688 *write*

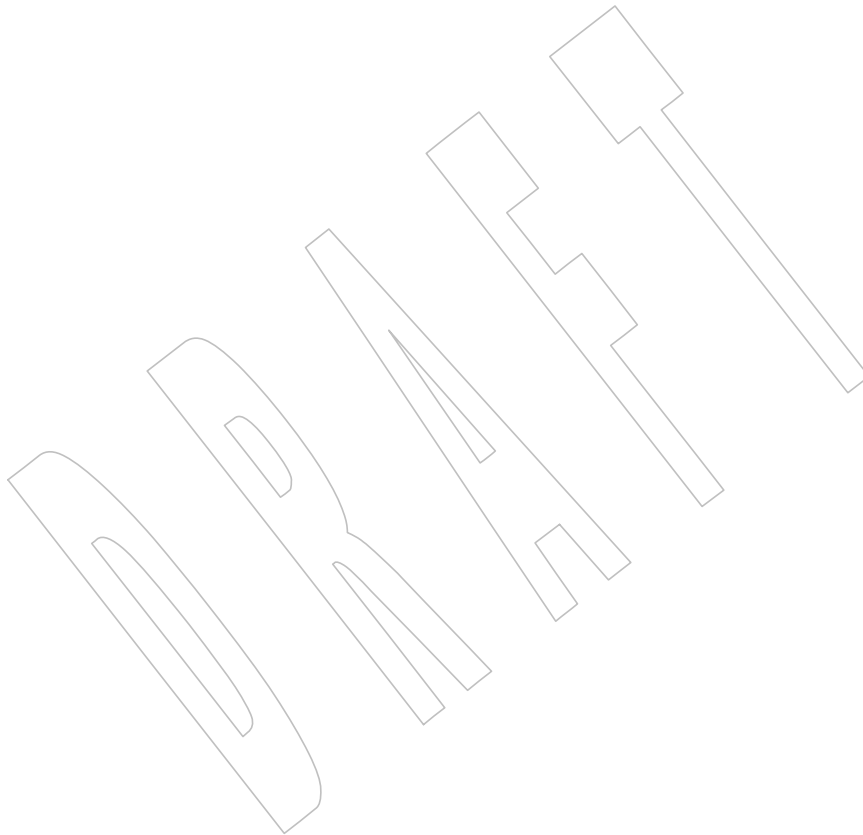
59689 XBD <sys/socket.h>

59690

59691

CHANGE HISTORY

First released in Issue 6. Derived from the XNS, Issue 5.2 specification.



59692 **NAME**

59693 sigaction — examine and change a signal action

59694 **SYNOPSIS**

```
59695 CX #include <signal.h>
59696 int sigaction(int sig, const struct sigaction *restrict act,
59697 struct sigaction *restrict oact);
```

59698 **DESCRIPTION**

59699 The *sigaction()* function allows the calling process to examine and/or specify the action to be
 59700 associated with a specific signal. The argument *sig* specifies the signal; acceptable values are
 59701 defined in **<signal.h>**.

59702 The structure **sigaction**, used to describe an action to be taken, is defined in the **<signal.h>**
 59703 header to include at least the following members:

Member Type	Member Name	Description
void(*) (int)	<i>sa_handler</i>	Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL.
sigset_t	<i>sa_mask</i>	Additional set of signals to be blocked during execution of signal-catching function.
int	<i>sa_flags</i>	Special flags to affect behavior of signal.
void(*) (int, siginfo_t *, void *)	<i>sa_sigaction</i>	Pointer to a signal-catching function.

59714 The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application
 59715 shall not use both simultaneously.

59716 If the argument *act* is not a null pointer, it points to a structure specifying the action to be
 59717 associated with the specified signal. If the argument *oact* is not a null pointer, the action
 59718 previously associated with the signal is stored in the location pointed to by the argument *oact*. If
 59719 the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to
 59720 enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall
 59721 not be added to the signal mask using this mechanism; this restriction shall be enforced by the
 59722 system without causing an error to be indicated.

59723 If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the
 59724 *sa_handler* field identifies the action to be associated with the specified signal. If the
 59725 SA_SIGINFO flag is set in the *sa_flags* field, the *sa_sigaction* field specifies a signal-catching
 59726 function.

59727 The *sa_flags* field can be used to modify the behavior of the specified signal.

59728 The following flags, defined in the **<signal.h>** header, can be set in *sa_flags*:

59729 XSI	SA_NOCLDSTOP	Do not generate SIGCHLD when children stop or stopped children continue.
59730		
59731		If <i>sig</i> is SIGCHLD and the SA_NOCLDSTOP flag is not set in <i>sa_flags</i> , and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If <i>sig</i>
59732		
59733		
59734 XSI		
59735		

59736			is SIGCHLD and the SA_NOCLDSTOP flag is set in <i>sa_flags</i> , then the implementation shall not generate a SIGCHLD signal in this way.
59737			
59738	XSI	SA_ONSTACK	If set and an alternate signal stack has been declared with <i>sigaltstack()</i> , the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.
59739			
59740			
59741		SA_RESETHAND	If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.
59742			
59743			Note: SIGILL and SIGTRAP cannot be automatically reset when delivered;
59744			the system silently enforces this restriction.
59745			Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.
59746			
59747			In addition, if this flag is set, <i>sigaction()</i> may behave as if the SA_NODEFER flag were also set.
59748			
59749		SA_RESTART	This flag affects the behavior of interruptible functions; that is, those specified to fail with <i>errno</i> set to [EINTR]. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with [EINTR] unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with <i>errno</i> set to [EINTR].
59750			
59751			
59752			
59753			
59754			
59755			
59756			
59757		SA_SIGINFO	If cleared and the signal is caught, the signal-catching function shall be entered as:
59758			
59759			<pre>void func(int signo);</pre>
59760			where <i>signo</i> is the only argument to the signal-catching function. In this case, the application shall use the <i>sa_handler</i> member to describe the signal-catching function and the application shall not modify the <i>sa_sigaction</i> member.
59761			
59762			
59763			
59764			If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:
59765			
59766			<pre>void func(int signo, siginfo_t *info, void *context);</pre>
59767			where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type siginfo_t explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type ucontext_t to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the <i>sa_sigaction</i> member to describe the signal-catching function and the application shall not modify the <i>sa_handler</i> member.
59768			
59769			
59770			
59771			
59772			
59773			
59774			
59775			The <i>si_signo</i> member contains the system-generated signal number.
59776	XSI		The <i>si_errno</i> member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.
59777			
59778			
59779			The <i>si_code</i> member contains a code identifying the cause of the signal.
59780			If the value of <i>si_code</i> is less than or equal to 0, then the signal was generated by a process and <i>si_pid</i> and <i>si_uid</i> , respectively, indicate the process ID and the real user ID of the sender. The <signal.h> header
59781			
59782			

59783 description contains information about the signal-specific contents of the
59784 elements of the **siginfo_t** type.

59785 **SA_NOCLDWAIT** If set, and *sig* equals SIGCHLD, child processes of the calling processes
59786 shall not be transformed into zombie processes when they terminate. If
59787 the calling process subsequently waits for its children, and the process has
59788 no unwaited-for children that were transformed into zombie processes, it
59789 shall block until all of its children terminate, and *wait()*, *waitid()*, and
59790 *waitpid()* shall fail and set *errno* to [ECHILD]. Otherwise, terminating
59791 child processes shall be transformed into zombie processes, unless
59792 SIGCHLD is set to SIG_IGN.

59793 **SA_NODEFER** If set and *sig* is caught, *sig* shall not be added to the thread's signal mask
59794 on entry to the signal handler unless it is included in *sa_mask*. Otherwise,
59795 *sig* shall always be added to the thread's signal mask on entry to the
59796 signal handler.

59797 When a signal is caught by a signal-catching function installed by *sigaction()*, a new signal mask
59798 is calculated and installed for the duration of the signal-catching function (or until a call to either
59799 *sigprocmask()* or *sigsuspend()* is made). This mask is formed by taking the union of the current
59800 signal mask and the value of the *sa_mask* for the signal being delivered, and unless
59801 SA_NODEFER or SA_RESETHAND is set, then including the signal being delivered. If and
59802 when the user's signal handler returns normally, the original signal mask is restored.

59803 Once an action is installed for a specific signal, it shall remain installed until another action is
59804 explicitly requested (by another call to *sigaction()*), until the SA_RESETHAND flag causes
59805 resetting of the handler, or until one of the *exec* functions is called.

59806 If the previous action for *sig* had been established by *signal()*, the values of the fields returned in
59807 the structure pointed to by *oact* are unspecified, and in particular *oact->sa_handler* is not
59808 necessarily the same value passed to *signal()*. However, if a pointer to the same structure or a
59809 copy thereof is passed to a subsequent call to *sigaction()* via the *act* argument, handling of the
59810 signal shall be as if the original call to *signal()* were repeated.

59811 If *sigaction()* fails, no new signal handler is installed.

59812 It is unspecified whether an attempt to set the action for a signal that cannot be caught or
59813 ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to [EINVAL].

59814 If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when
59815 it is already pending is implementation-defined; the signal-catching function shall be invoked
59816 with a single argument. If SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig*
59817 generated by *sigqueue()* or as a result of any signal-generating function that supports the
59818 specification of an application-defined value (when *sig* is already pending) shall be queued in
59819 FIFO order until delivered or accepted; the signal-catching function shall be invoked with three
59820 arguments. The application specified value is passed to the signal-catching function as the
59821 *si_value* member of the **siginfo_t** structure.

59822 The result of the use of *sigaction()* and a *sigwait()* function concurrently within a process on the
59823 same signal is unspecified.

59824 **RETURN VALUE**

59825 Upon successful completion, *sigaction()* shall return 0; otherwise, -1 shall be returned, *errno* shall
59826 be set to indicate the error, and no new signal-catching function shall be installed.

59827 **ERRORS**59828 The *sigaction()* function shall fail if:59829 [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
59830 signal that cannot be caught or ignore a signal that cannot be ignored.59831 [ENOTSUP] The SA_SIGINFO bit flag is set in the *sa_flags* field of the **sigaction** structure.59832 The *sigaction()* function may fail if:59833 [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
59834 caught or ignored (or both).59835 In addition, the *sigaction()* function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of
59836 the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.59837 **EXAMPLES**59838 **Establishing a Signal Handler**59839 The following example demonstrates the use of *sigaction()* to establish a handler for the SIGINT
59840 signal.

```

59841 #include <signal.h>
59842 static void handler(int signum)
59843 {
59844     /* Take appropriate actions for signal delivery */
59845 }
59846 int main()
59847 {
59848     struct sigaction sa;
59849     sa.sa_handler = handler;
59850     sigemptyset(&sa.sa_mask);
59851     sa.sa_flags = SA_RESTART; /* Restart functions if
59852                             interrupted by handler */
59853     if (sigaction(SIGINT, &sa, NULL) == -1)
59854         /* Handle error */;
59855     /* Further code */
59856 }

```

59857 **APPLICATION USAGE**

59858 The *sigaction()* function supersedes the *signal()* function, and should be used in preference. In
59859 particular, *sigaction()* and *signal()* should not be used in the same process to control the same
59860 signal. The behavior of reentrant functions, as defined in the DESCRIPTION, is as specified by
59861 this volume of POSIX.1-200x, regardless of invocation from a signal-catching function. This is the
59862 only intended meaning of the statement that reentrant functions may be used in signal-catching
59863 functions without restrictions. Applications must still consider all effects of such functions on
59864 such things as data structures, files, and process state. In particular, application developers need
59865 to consider the restrictions on interactions when interrupting *sleep()* and interactions among
59866 multiple handles for a file description. The fact that any specific function is listed as reentrant
59867 does not necessarily mean that invocation of that function from a signal-catching function is
59868 recommended.

59869 In order to prevent errors arising from interrupting non-reentrant function calls, applications
59870 should protect calls to these functions either by blocking the appropriate signals or through the
59871 use of some programmatic semaphore (see *semget()*, *sem_init()*, *sem_open()*, and so on). Note in
59872 particular that even the "safe" functions may modify *errno*; the signal-catching function, if not

executing as an independent thread, should save and restore its value in order to avoid the possibility that delivery of a signal in between an error return from a function that sets *errno* and the subsequent examination of *errno* could result in the signal-catching function changing the value of *errno*. Naturally, the same principles apply to the reentrancy of application routines and asynchronous data access. Note that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the code executing after *longjmp()* and *siglongjmp()* can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp()* and *siglongjmp()* from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either *malloc()* or *free()* functions or the standard I/O library, both of which traditionally use data structures in a non-reentrant manner. Since any combination of different functions using a common data structure can cause reentrancy problems, this volume of POSIX.1-200x does not define the behavior when any unsafe function is called in a signal handler that interrupts an unsafe function.

If the signal occurs other than as the result of calling *abort()*, *kill()*, or *raise()*, the behavior is undefined if the signal handler calls any function in the standard library other than one of the functions listed in the table above or refers to any object other than *errno* with static storage duration other than by assigning a value to a static storage duration variable of type **volatile sig_atomic_t**. Unless all signal handlers have *errno* set on return as it was on entry, the value of *errno* is unspecified.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving thread resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp()* or *_longjmp()* is used to leave the signal handler, then the signal mask must be explicitly restored.

This volume of POSIX.1-200x defines the third argument of a signal handling function when SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t ***.

The BSD optional four argument signal handling function is not supported by this volume of POSIX.1-200x. The BSD declaration would be:

```
void handler(int sig, int code, struct sigcontext *scp,
             char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

Since the *sigaction()* function is allowed but not required to set SA_NODEFER when the application sets the SA_RESETHAND flag, applications which depend on the SA_RESETHAND functionality for the newly installed signal handler must always explicitly set SA_NODEFER when they set SA_RESETHAND in order to be portable.

RATIONALE

Although this volume of POSIX.1-200x requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction()* reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction()* returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act->sa_flags* parameter, allows overloading

59922 SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated
 59923 child. Most conforming applications that catch SIGCHLD are expected to install signal-catching
 59924 functions that repeatedly call the *waitpid()* function with the WNOHANG flag set, acting on
 59925 each child for which status is returned, until *waitpid()* returns zero. If stopped children are not of
 59926 interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the
 59927 signal-catching routine when they stop.

59928 Some historical implementations also define other mechanisms for stopping processes, such as
 59929 the *ptrace()* function. These implementations usually do not generate a SIGCHLD signal when
 59930 processes stop due to this mechanism; however, that is beyond the scope of this volume of
 59931 POSIX.1-200x.

59932 This volume of POSIX.1-200x requires that calls to *sigaction()* that supply a NULL *act* argument
 59933 succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or
 59934 SIGSTOP). The System V *signal()* and BSD *sigvec()* functions return [EINVAL] in these cases
 59935 and, in this respect, their behavior varies from *sigaction()*.

59936 This volume of POSIX.1-200x requires that *sigaction()* properly save and restore a signal action
 59937 set up by the ISO C standard *signal()* function. However, there is no guarantee that the reverse is
 59938 true, nor could there be given the greater amount of information conveyed by the **sigaction**
 59939 structure. Because of this, applications should avoid using both functions for the same signal in
 59940 the same process. Since this cannot always be avoided in case of general-purpose library
 59941 routines, they should always be implemented with *sigaction()*.

59942 It was intended that the *signal()* function should be implementable as a library routine using
 59943 *sigaction()*.

59944 The POSIX Realtime Extension extends the *sigaction()* function as specified by the POSIX.1-1990
 59945 standard to allow the application to request on a per-signal basis via an additional signal action
 59946 flag that the extra parameters, including the application-defined signal value, if any, be passed to
 59947 the signal-catching function.

59948 FUTURE DIRECTIONS

59949 None.

59950 SEE ALSO

59951 [Section 2.4](#) (on page 463), *exec*, *kill*, *_longjmp()*, *longjmp()*, *pthread_sigmask()*, *raise()*, *semget()*,
 59952 *sem_init()*, *sem_open()*, *sigaddset()*, *sigaltstack()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
 59953 *sigismember()*, *signal()*, *sigsuspend()*, *wait*, *waitid()*

59954 XBD [<signal.h>](#)

59955 CHANGE HISTORY

59956 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

59957 Issue 5

59958 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and POSIX
 59959 Threads Extension.

59960 In the DESCRIPTION, the second argument to *func* when SA_SIGINFO is set is no longer
 59961 permitted to be NULL, and the description of permitted **siginfo_t** contents is expanded by
 59962 reference to [<signal.h>](#).

59963 Since the X/OPEN UNIX Extension functionality is now folded into the BASE, the [ENOTSUP]
 59964 error is deleted.

59965 Issue 6

59966 The Open Group Corrigendum U028/7 is applied. In the paragraph entitled “Signal Effects on
 59967 Other Functions”, a reference to *sigpending()* is added.

59968 In the DESCRIPTION, the text “Signal Generation and Delivery”, “Signal Actions”, and “Signal

59969	Effects on Other Functions” are moved to a separate section of this volume of POSIX.1-200x.	
59970	Text describing functionality from the Realtime Signals Extension option is marked.	
59971	The following changes are made for alignment with the ISO POSIX-1: 1996 standard:	
59972	<ul style="list-style-type: none"> • The [ENOTSUP] error condition is added. 	
59973	The normative text is updated to avoid use of the term “must” for application requirements.	
59974	The restrict keyword is added to the <i>sigaction()</i> prototype for alignment with the	
59975	ISO/IEC 9899: 1999 standard.	
59976	References to the <i>wait3()</i> function are removed.	
59977	The SYNOPSIS is marked CX since the presence of this function in the <signal.h> header is an	
59978	extension over the ISO C standard.	
59979	IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/57 is applied, changing text in the table	
59980	describing the sigaction structure.	
59981	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/127 is applied, removing text from the	
59982	DESCRIPTION duplicated later in the same section.	
59983	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/128 is applied, updating the	
59984	DESCRIPTION and APPLICATION USAGE sections. Changes are made to refer to the thread	
59985	rather than the process.	
59986	IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/129 is applied, adding the example to the	
59987	EXAMPLES section.	
59988	Issue 7	
59989	Austin Group Interpretations 1003.1-2001 #065 and #084 are applied, clarifying the role of the	
59990	SA_NODEFER flag with respect to the signal mask, and clarifying the SA_RESTART flag for	
59991	interrupted functions which use timeouts.	
59992	Austin Group Interpretation 1003.1-2001 #004 is applied.	
59993	SD5-XSH-ERN-167 is applied, updating the APPLICATION USAGE section.	+
59994	SD5-XSH-ERN-172 is applied, updating the DESCRIPTION to make optional the requirement	+
59995	that when the SA_RESETHAND flag is set, <i>sigaction()</i> shall behave as if the SA_NODEFER flag	+
59996	were also set.	+
59997	Functionality relating to the Realtime Signals Extension option is moved to the Base.	

59998 **NAME**
 59999 sigaddset — add a signal to a signal set

60000 **SYNOPSIS**

60001 CX

```
#include <signal.h>
```


 60002

```
int sigaddset(sigset_t *set, int signo);
```

60003 **DESCRIPTION**

60004 The *sigaddset()* function adds the individual signal specified by the *signo* to the signal set pointed
 60005 to by *set*.

60006 Applications shall call either *sigemptyset()* or *sigfillset()* at least once for each object of type
 60007 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
 60008 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
 60009 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
 60010 *sigwaitinfo()*, the results are undefined.

60011 **RETURN VALUE**

60012 Upon successful completion, *sigaddset()* shall return 0; otherwise, it shall return -1 and set *errno*
 60013 to indicate the error.

60014 **ERRORS**

60015 The *sigaddset()* function may fail if:

60016 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

60017 **EXAMPLES**

60018 None.

60019 **APPLICATION USAGE**

60020 None.

60021 **RATIONALE**

60022 None.

60023 **FUTURE DIRECTIONS**

60024 None.

60025 **SEE ALSO**

60026 Section 2.4 (on page 463), *pthread_sigmask()*, *sigaction()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*,
 60027 *sigismember()*, *sigpending()*, *sigsuspend()*

60028 XBD **<signal.h>**

60029 **CHANGE HISTORY**

60030 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60031 **Issue 5**

60032 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 60033 previous issues.

60034 **Issue 6**

60035 The normative text is updated to avoid use of the term “must” for application requirements.

60036 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
 60037 extension over the ISO C standard.

60038 **NAME**
 60039 sigaltstack — set and get signal alternate stack context

60040 **SYNOPSIS**

```
60041 XSI #include <signal.h>
60042 int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

60043 **DESCRIPTION**

60044 The *sigaltstack()* function allows a process to define and examine the state of an alternate stack
 60045 for signal handlers for the current thread. Signals that have been explicitly declared to execute
 60046 on the alternate stack shall be delivered on the alternate stack.

60047 If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack
 60048 that shall take effect upon return from *sigaltstack()*. The *ss_flags* member specifies the new stack
 60049 state. If it is set to *SS_DISABLE*, the stack is disabled and *ss_sp* and *ss_size* are ignored.
 60050 Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address
 60051 and size of the stack.

60052 The range of addresses starting at *ss_sp* up to but not including *ss_sp+ss_size* is available to the
 60053 implementation for use as the stack. This function makes no assumptions regarding which end
 60054 is the stack base and in which direction the stack grows as items are pushed.

60055 If *oss* is not a null pointer, on successful completion it shall point to a **stack_t** structure that
 60056 specifies the alternate signal stack that was in effect prior to the call to *sigaltstack()*. The *ss_sp*
 60057 and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the
 60058 stack's state, and may contain one of the following values:

60059 **SS_ONSTACK** The process is currently executing on the alternate signal stack. Attempts to
 60060 modify the alternate signal stack while the process is executing on it fail. This
 60061 flag shall not be modified by processes.

60062 **SS_DISABLE** The alternate signal stack is currently disabled.

60063 The value **SIGSTKSZ** is a system default specifying the number of bytes that would be used to
 60064 cover the usual case when manually allocating an alternate stack area. The value **MINSIGSTKSZ**
 60065 is defined to be the minimum stack size for a signal handler. In computing an alternate stack
 60066 size, a program should add that amount to its stack requirements to allow for the system
 60067 implementation overhead. The constants **SS_ONSTACK**, **SS_DISABLE**, **SIGSTKSZ**, and
 60068 **MINSIGSTKSZ** are defined in **<signal.h>**.

60069 After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new
 60070 process image.

60071 In some implementations, a signal (whether or not indicated to execute on the alternate stack)
 60072 shall always execute on the alternate stack if it is delivered while another signal is being caught
 60073 using the alternate stack.

60074 Use of this function by library threads that are not bound to kernel-scheduled entities results in
 60075 undefined behavior.

60076 **RETURN VALUE**

60077 Upon successful completion, *sigaltstack()* shall return 0; otherwise, it shall return -1 and set *errno*
 60078 to indicate the error.

60079 ERRORS

60080 The *sigaltstack()* function shall fail if:

- 60081 [EINVAL] The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss*
60082 contains flags other than SS_DISABLE.
- 60083 [ENOMEM] The size of the alternate stack area is less than MINSIGSTKSZ.
- 60084 [EPERM] An attempt was made to modify an active stack.

60085 EXAMPLES**60086 Allocating Memory for an Alternate Stack**

60087 The following example illustrates a method for allocating memory for an alternate stack.

```
60088 #include <signal.h>
60089 ...
60090 if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
60091     /* Error return. */
60092     sigstk.ss_size = SIGSTKSZ;
60093     sigstk.ss_flags = 0;
60094     if (sigaltstack(&sigstk, (stack_t *)0) < 0)
60095         perror("sigaltstack");
```

60096 APPLICATION USAGE

60097 On some implementations, stack space is automatically extended as needed. On those
60098 implementations, automatic extension is typically not available for an alternate stack. If the stack
60099 overflows, the behavior is undefined.

60100 RATIONALE

60101 None.

60102 FUTURE DIRECTIONS

60103 None.

60104 SEE ALSO

60105 [Section 2.4](#) (on page 463), *exec*, *sigaction()*, *sigsetjmp()*

60106 XBD [<signal.h>](#)

60107 CHANGE HISTORY

60108 First released in Issue 4, Version 2.

60109 Issue 5

60110 Moved from X/OPEN UNIX extension to BASE.

60111 The last sentence of the DESCRIPTION was included as an APPLICATION USAGE note in
60112 previous issues.

60113 Issue 6

60114 The normative text is updated to avoid use of the term “must” for application requirements.

60115 The **restrict** keyword is added to the *sigaltstack()* prototype for alignment with the
60116 ISO/IEC 9899:1999 standard.

60117 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/58 is applied, updating the first sentence
60118 to include “for the current thread”.

60119 NAME

60120 sigdelset — delete a signal from a signal set

60121 SYNOPSIS

```
60122 CX #include <signal.h>
60123 int sigdelset(sigset_t *set, int signo);
```

60124 DESCRIPTION

60125 The *sigdelset()* function deletes the individual signal specified by *signo* from the signal set
60126 pointed to by *set*.

60127 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
60128 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
60129 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
60130 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
60131 *sigwaitinfo()*, the results are undefined.

60132 RETURN VALUE

60133 Upon successful completion, *sigdelset()* shall return 0; otherwise, it shall return -1 and set *errno*
60134 to indicate the error.

60135 ERRORS

60136 The *sigdelset()* function may fail if:

60137	[EINVAL]	The <i>signo</i> argument is not a valid signal number, or is an unsupported signal
60138		number.

60139 EXAMPLES

60140 None.

60141 APPLICATION USAGE

60142 None.

60143 RATIONALE

60144 None.

60145 FUTURE DIRECTIONS

60146 None.

60147 SEE ALSO

60148 [Section 2.4](#) (on page 463), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigemptyset()*, *sigfillset()*,
60149 *sigismember()*, *sigpending()*, *sigsuspend()*

60150 XBD [<signal.h>](#)

60151 CHANGE HISTORY

60152 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60153 Issue 5

60154 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
60155 previous issues.

60156 Issue 6

60157 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
60158 extension over the ISO C standard.

60159 **NAME**

60160 sigemptyset — initialize and empty a signal set

60161 **SYNOPSIS**

```
60162 CX #include <signal.h>
60163 int sigemptyset(sigset_t *set);
```

60164 **DESCRIPTION**

60165 The *sigemptyset()* function initializes the signal set pointed to by *set*, such that all signals defined
60166 in POSIX.1-200x are excluded.

60167 **RETURN VALUE**

60168 Upon successful completion, *sigemptyset()* shall return 0; otherwise, it shall return -1 and set
60169 *errno* to indicate the error.

60170 **ERRORS**

60171 No errors are defined.

60172 **EXAMPLES**

60173 None.

60174 **APPLICATION USAGE**

60175 None.

60176 **RATIONALE**

60177 The implementation of the *sigemptyset()* (or *sigfillset()*) function could quite trivially clear (or set)
60178 all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the
60179 structure, such as a version field, to permit binary-compatibility between releases where the size
60180 of the set varies. For such reasons, either *sigemptyset()* or *sigfillset()* must be called prior to any
60181 other use of the signal set, even if such use is read-only (for example, as an argument to
60182 *sigpending()*). This function is not intended for dynamic allocation.

60183 The *sigfillset()* and *sigemptyset()* functions require that the resulting signal set include (or
60184 exclude) all the signals defined in this volume of POSIX.1-200x. Although it is outside the scope
60185 of this volume of POSIX.1-200x to place this requirement on signals that are implemented as
60186 extensions, it is recommended that implementation-defined signals also be affected by these
60187 functions. However, there may be a good reason for a particular signal not to be affected. For
60188 example, blocking or ignoring an implementation-defined signal may have undesirable side
60189 effects, whereas the default action for that signal is harmless. In such a case, it would be
60190 preferable for such a signal to be excluded from the signal set returned by *sigfillset()*.

60191 In early proposals there was no distinction between invalid and unsupported signals (the names
60192 of optional signals that were not supported by an implementation were not defined by that
60193 implementation). The [EINVAL] error was thus specified as a required error for invalid signals.
60194 With that distinction, it is not necessary to require implementations of these functions to
60195 determine whether an optional signal is actually supported, as that could have a significant
60196 performance impact for little value. The error could have been required for invalid signals and
60197 optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is
60198 optional in both cases.

60199 **FUTURE DIRECTIONS**

60200 None.

sigemptyset()60201
60202
60203
60204
60205
60206
60207
60208
60209**SEE ALSO**

Section 2.4 (on page 463), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*, *sigismember()*, *sigpending()*, *sigsuspend()*

XBD **<signal.h>**

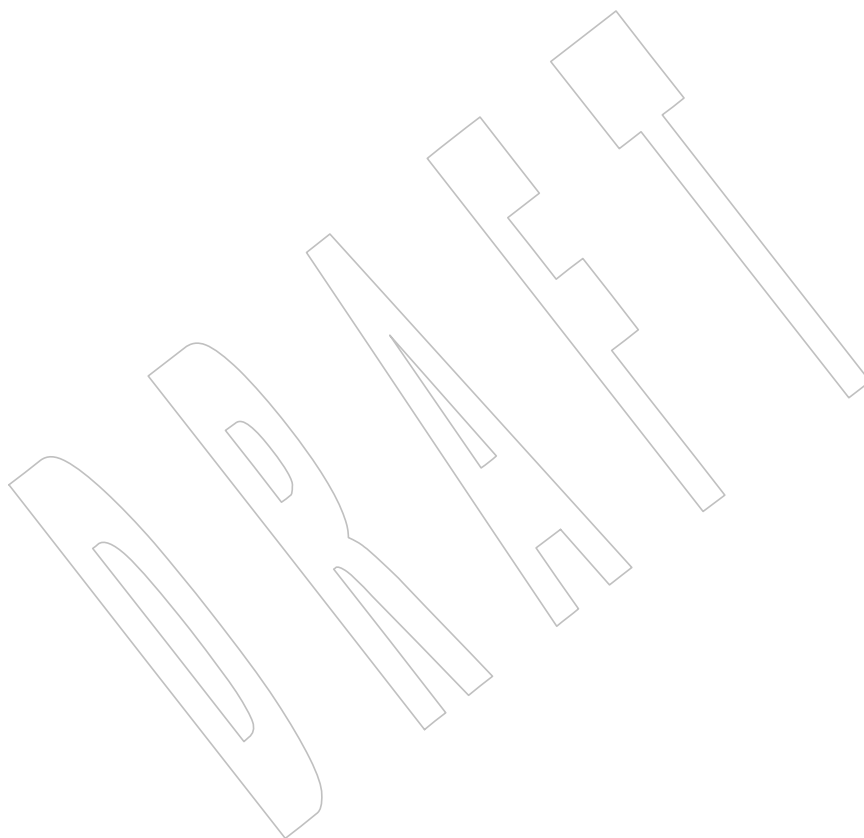
+

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an extension over the ISO C standard.



60210 **NAME**
 60211 sigfillset — initialize and fill a signal set

60212 **SYNOPSIS**

60213 CX `#include <signal.h>`
 60214 `int sigfillset(sigset_t *set);`

60215 **DESCRIPTION**

60216 The *sigfillset()* function shall initialize the signal set pointed to by *set*, such that all signals
 60217 defined in this volume of POSIX.1-200x are included.

60218 **RETURN VALUE**

60219 Upon successful completion, *sigfillset()* shall return 0; otherwise, it shall return -1 and set *errno*
 60220 to indicate the error.

60221 **ERRORS**

60222 No errors are defined.

60223 **EXAMPLES**

60224 None.

60225 **APPLICATION USAGE**

60226 None.

60227 **RATIONALE**

60228 Refer to *sigemptyset()* (on page 1881).

60229 **FUTURE DIRECTIONS**

60230 None.

60231 **SEE ALSO**

60232 Section 2.4 (on page 463), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*,
 60233 *sigismember()*, *sigpending()*, *sigsuspend()*

60234 XBD [<signal.h>](#)

60235 **CHANGE HISTORY**

60236 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60237 **Issue 6**

60238 The SYNOPSIS is marked CX since the presence of this function in the `<signal.h>` header is an
 60239 extension over the ISO C standard.

60240 **NAME**

60241 sighold, sigignore, sigpause, sigrelse, sigset — signal management

60242 **SYNOPSIS**

```
60243 OB XSI #include <signal.h>
60244 int sighold(int sig);
60245 int sigignore(int sig);
60246 int sigpause(int sig);
60247 int sigrelse(int sig);
60248 void (*sigset(int sig, void (*disp)(int)))(int);
```

60249 **DESCRIPTION**

60250 Use of any of these functions is unspecified in a multi-threaded process.

60251 The *sighold()*, *sigignore()*, *sigpause()*, *sigrelse()*, and *sigset()* functions provide simplified signal
60252 management.

60253 The *sigset()* function shall modify signal dispositions. The *sig* argument specifies the signal,
60254 which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's
60255 disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset()* is
60256 used, and *disp* is the address of a signal handler, the system shall add *sig* to the signal mask of
60257 the calling process before executing the signal handler; when the signal handler returns, the
60258 system shall restore the signal mask of the calling process to its state prior to the delivery of the
60259 signal. In addition, if *sigset()* is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the
60260 signal mask of the calling process and *sig*'s disposition shall remain unchanged. If *sigset()* is
60261 used, and *disp* is not equal to SIG_HOLD, *sig* shall be removed from the signal mask of the
60262 calling process.

60263 The *sighold()* function shall add *sig* to the signal mask of the calling process.60264 The *sigrelse()* function shall remove *sig* from the signal mask of the calling process.60265 The *sigignore()* function shall set the disposition of *sig* to SIG_IGN.

60266 The *sigpause()* function shall remove *sig* from the signal mask of the calling process and suspend
60267 the calling process until a signal is received. The *sigpause()* function shall restore the signal mask
60268 of the process to its original state before returning.

60269 If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes
60270 shall not be transformed into zombie processes when they terminate. If the calling process
60271 subsequently waits for its children, and the process has no unwaited-for children that were
60272 transformed into zombie processes, it shall block until all of its children terminate, and *wait()*,
60273 *waitid()*, and *waitpid()* shall fail and set *errno* to [ECHILD].

60274 **RETURN VALUE**

60275 Upon successful completion, *sigset()* shall return SIG_HOLD if the signal had been blocked and
60276 the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be
60277 returned and *errno* set to indicate the error.

60278 The *sigpause()* function shall suspend execution of the thread until a signal is received,
60279 whereupon it shall return -1 and set *errno* to [EINTR].

60280 For all other functions, upon successful completion, 0 shall be returned. Otherwise, -1 shall be
60281 returned and *errno* set to indicate the error.

60282 **ERRORS**

60283 These functions shall fail if:

60284 [EINVAL] The *sig* argument is an illegal signal number.60285 The *sigset()* and *sigignore()* functions shall fail if:60286 [EINVAL] An attempt is made to catch a signal that cannot be caught, or to ignore a
60287 signal that cannot be ignored.60288 **EXAMPLES**

60289 None.

60290 **APPLICATION USAGE**60291 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
60292 signals; new applications should use the *sigaction()* function instead of the obsolescent *sigset()*
60293 function.60294 The *sighold()* function, in conjunction with *sigrelse()* or *sigpause()*, may be used to establish
60295 critical regions of code that require the delivery of a signal to be temporarily deferred. For
60296 broader portability, the *pthread_sigmask()* or *sigprocmask()* functions should be used instead of
60297 the obsolescent *sighold()* and *sigrelse()* functions.60298 For broader portability, the *sigsuspend()* function should be used instead of the obsolescent
60299 *sigpause()* function.60300 **RATIONALE**60301 Each of these historic functions has a direct analog in the other functions which are required to
60302 be per-thread and thread-safe (aside from *sigprocmask()*, which is replaced by *pthread_sigmask()*).
60303 The *sigset()* function can be implemented as a simple wrapper for *sigaction()*. The *sighold()*
60304 function is equivalent to *sigprocmask()* or *pthread_sigmask()* with SIG_BLOCK set. The *sigignore()*
60305 function is equivalent to *sigaction()* with SIG_IGN set. The *sigpause()* function is equivalent to
60306 *sigsuspend()*. The *sigrelse()* function is equivalent to *sigprocmask()* or *pthread_sigmask()* with
60307 SIG_UNBLOCK set.60308 **FUTURE DIRECTIONS**

60309 These functions may be removed in a future version.

60310 **SEE ALSO**60311 Section 2.4 (on page 463), *exec*, *pause()*, *pthread_sigmask()*, *sigaction()*, *signal()*, *sigsuspend()*, *wait*,
60312 *waitid()*

60313 XBD <signal.h>

60314 **CHANGE HISTORY**

60315 First released in Issue 4, Version 2.

60316 **Issue 5**

60317 Moved from X/OPEN UNIX extension to BASE.

60318 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of
60319 the process to its original state before returning.60320 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
60321 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
60322 [EINTR].60323 **Issue 6**

60324 The normative text is updated to avoid use of the term “must” for application requirements.

60325 References to the *wait3()* function are removed.

60326 The XSI functions are split out into their own reference page.

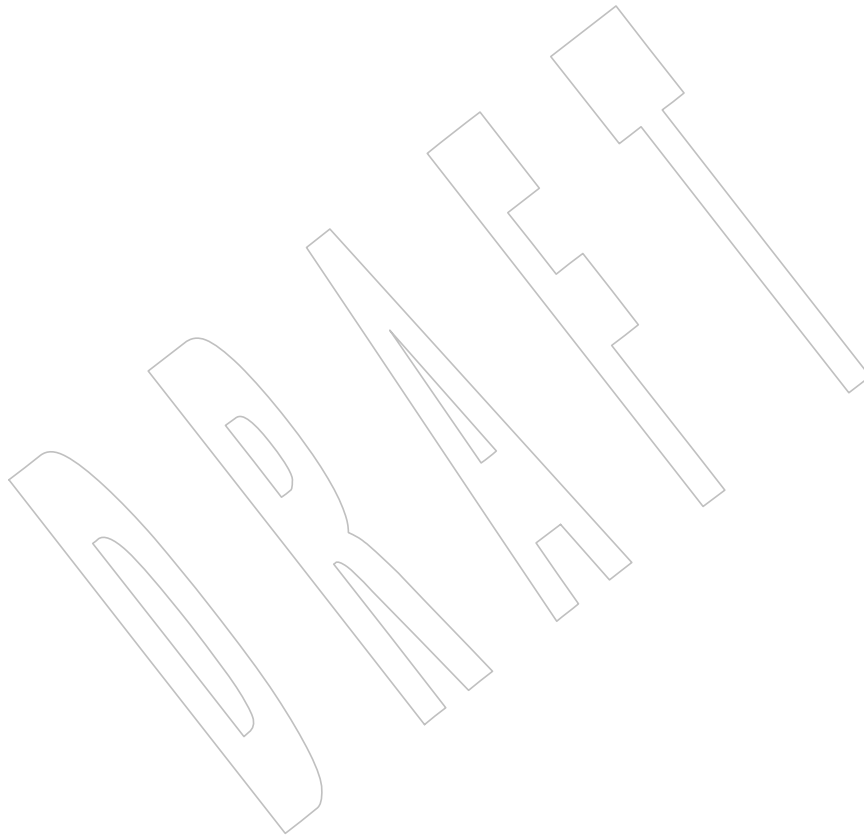
60327

Issue 7

60328

SD5-XSH-ERN-113 and SD5-XSH-ERN-42 are applied, marking these functions obsolescent and updating the APPLICATION USAGE and RATIONALE sections.

60329



60330 **NAME**
 60331 siginterrupt — allow signals to interrupt functions

60332 **SYNOPSIS**

60333 OB XSI `#include <signal.h>`
 60334 `int siginterrupt(int sig, int flag);`

60335 **DESCRIPTION**

60336 The *siginterrupt()* function shall change the restart behavior when a function is interrupted by
 60337 the specified signal. The function *siginterrupt(sig, flag)* has an effect as if implemented as:

```
60338 int siginterrupt(int sig, int flag) {
60339     int ret;
60340     struct sigaction act;
60341
60342     (void) sigaction(sig, NULL, &act);
60343     if (flag)
60344         act.sa_flags &= ~SA_RESTART;
60345     else
60346         act.sa_flags |= SA_RESTART;
60347     ret = sigaction(sig, &act, NULL);
60348     return ret;
60349 }
```

60349 **RETURN VALUE**

60350 Upon successful completion, *siginterrupt()* shall return 0; otherwise, -1 shall be returned and
 60351 *errno* set to indicate the error.

60352 **ERRORS**

60353 The *siginterrupt()* function shall fail if:

60354 [EINVAL] The *sig* argument is not a valid signal number.

60355 **EXAMPLES**

60356 None.

60357 **APPLICATION USAGE**

60358 The *siginterrupt()* function supports programs written to historical system interfaces.
 60359 Applications should use the *sigaction()* with the SA_RESTART flag instead of the obsolescent
 60360 *siginterrupt()* function.

60361 **RATIONALE**

60362 None.

60363 **FUTURE DIRECTIONS**

60364 None.

60365 **SEE ALSO**

60366 [Section 2.4](#) (on page 463), *sigaction()*

60367 XBD [<signal.h>](#)

60368 **CHANGE HISTORY**

60369 First released in Issue 4, Version 2.

siginterrupt()

60370

Issue 5

60371

Moved from X/OPEN UNIX extension to BASE.

60372

Issue 6

60373

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/59 is applied, correcting the declaration in the sample implementation given in the DESCRIPTION.

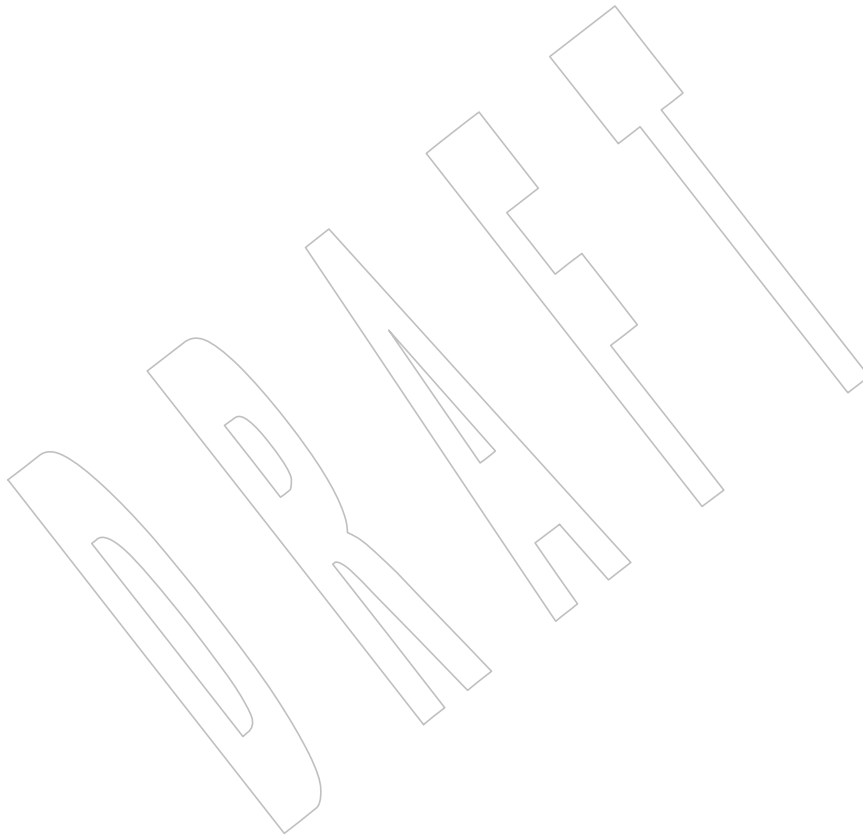
60374

60375

Issue 7

60376

The *siginterrupt()* function is marked obsolescent.



60377 **NAME**

60378 sigismember — test for a signal in a signal set

60379 **SYNOPSIS**

```
60380 CX #include <signal.h>
60381 int sigismember(const sigset_t *set, int signo);
```

60382 **DESCRIPTION**

60383 The *sigismember()* function shall test whether the signal specified by *signo* is a member of the set
 60384 pointed to by *set*.

60385 Applications should call either *sigemptyset()* or *sigfillset()* at least once for each object of type
 60386 **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is
 60387 nonetheless supplied as an argument to any of *pthread_sigmask()*, *sigaction()*, *sigaddset()*,
 60388 *sigdelset()*, *sigismember()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigtimedwait()*, *sigwait()*, or
 60389 *sigwaitinfo()*, the results are undefined.

60390 **RETURN VALUE**

60391 Upon successful completion, *sigismember()* shall return 1 if the specified signal is a member of
 60392 the specified set, or 0 if it is not. Otherwise, it shall return -1 and set *errno* to indicate the error.

60393 **ERRORS**

60394 The *sigismember()* function may fail if:

60395 [EINVAL] The *signo* argument is not a valid signal number, or is an unsupported signal
 60396 number.

60397 **EXAMPLES**

60398 None.

60399 **APPLICATION USAGE**

60400 None.

60401 **RATIONALE**

60402 None.

60403 **FUTURE DIRECTIONS**

60404 None.

60405 **SEE ALSO**

60406 Section 2.4 (on page 463), *pthread_sigmask()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigfillset()*,
 60407 *sigemptyset()*, *sigpending()*, *sigsuspend()*

60408 XBD **<signal.h>**

60409 **CHANGE HISTORY**

60410 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60411 **Issue 5**

60412 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 60413 previous issues.

60414 **Issue 6**

60415 The SYNOPSIS is marked CX since the presence of this function in the **<signal.h>** header is an
 60416 extension over the ISO C standard.

60417 **NAME**
 60418 siglongjmp — non-local goto with signal handling

60419 **SYNOPSIS**
 60420 CX `#include <setjmp.h>`
 60421 `void siglongjmp(sigjmp_buf env, int val);`

60422 **DESCRIPTION**
 60423 The *siglongjmp()* function shall be equivalent to the *longjmp()* function, except as follows:

- 60424 • References to *setjmp()* shall be equivalent to *sigsetjmp()*.
- 60425 • The *siglongjmp()* function shall restore the saved signal mask if and only if the *env*
 60426 argument was initialized by a call to *sigsetjmp()* with a non-zero *savemask* argument.

60427 **RETURN VALUE**
 60428 After *siglongjmp()* is completed, program execution shall continue as if the corresponding
 60429 invocation of *sigsetjmp()* had just returned the value specified by *val*. The *siglongjmp()* function
 60430 shall not cause *sigsetjmp()* to return 0; if *val* is 0, *sigsetjmp()* shall return the value 1.

60431 **ERRORS**
 60432 No errors are defined.

60433 **EXAMPLES**
 60434 None.

60435 **APPLICATION USAGE**
 60436 The distinction between *setjmp()* or *longjmp()* and *sigsetjmp()* or *siglongjmp()* is only significant
 60437 for programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

60438 **RATIONALE**
 60439 None.

60440 **FUTURE DIRECTIONS**
 60441 None.

60442 **SEE ALSO**
 60443 *longjmp()*, *pthread_sigmask()*, *setjmp()*, *sigsetjmp()*, *sigsuspend()*
 60444 XBD `<setjmp.h>`

60445 **CHANGE HISTORY**
 60446 First released in Issue 3. Included for alignment with the ISO POSIX-1 standard.

60447 **Issue 5**
 60448 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60449 **Issue 6**
 60450 The DESCRIPTION is rewritten in terms of *longjmp()*.
 60451 The SYNOPSIS is marked CX since the presence of this function in the `<setjmp.h>` header is an
 60452 extension over the ISO C standard.

60453 **NAME**60454 `signal` — signal management60455 **SYNOPSIS**60456 `#include <signal.h>`60457 `void (*signal(int sig, void (*func)(int)))(int);`60458 **DESCRIPTION**60459 CX The functionality described on this reference page is aligned with the ISO C standard. Any
60460 conflict between the requirements described here and the ISO C standard is unintentional. This
60461 volume of POSIX.1-200x defers to the ISO C standard.

60462 Use of this function is unspecified in a multi-threaded process.

60463 The `signal()` function chooses one of three ways in which receipt of the signal number `sig` is to be
60464 subsequently handled. If the value of `func` is `SIG_DFL`, default handling for that signal shall
60465 occur. If the value of `func` is `SIG_IGN`, the signal shall be ignored. Otherwise, the application
60466 shall ensure that `func` points to a function to be called when that signal occurs. An invocation of
60467 such a function because of a signal, or (recursively) of any further functions called by that
60468 invocation (other than functions in the standard library), is called a “signal handler”.60469 When a signal occurs, and `func` points to a function, it is implementation-defined whether the
60470 equivalent of a:60471 `signal(sig, SIG_DFL);`60472 is executed or the implementation prevents some implementation-defined set of signals (at least
60473 including `sig`) from occurring until the current signal handling has completed. (If the value of `sig`
60474 is `SIGILL`, the implementation may alternatively define that no action is taken.) Next the
60475 equivalent of:60476 `(*func)(sig);`60477 is executed. If and when the function returns, if the value of `sig` was `SIGFPE`, `SIGILL`, or
60478 `SIGSEGV` or any other implementation-defined value corresponding to a computational
60479 exception, the behavior is undefined. Otherwise, the program shall resume execution at the
60480 point it was interrupted. If the signal occurs as the result of calling the `abort()`, `raise()`, `kill()`,
60481 `pthread_kill()`, or `sigqueue()` function, the signal handler shall not call the `raise()` function.60482 CX If the signal occurs other than as the result of calling `abort()`, `raise()`, `kill()`, `pthread_kill()`, or
60483 `sigqueue()`, the behavior is undefined if the signal handler refers to any object with static storage
60484 duration other than by assigning a value to an object declared as volatile `sig_atomic_t`, or if the
60485 signal handler calls any function in the standard library other than one of the functions listed in
60486 [Section 2.4](#) (on page 463). Furthermore, if such a call fails, the value of `errno` is unspecified.

60487 At program start-up, the equivalent of:

60488 `signal(sig, SIG_IGN);`

60489 is executed for some signals, and the equivalent of:

60490 `signal(sig, SIG_DFL);`60491 CX is executed for all other signals (see [exec](#)).

60492 RETURN VALUE

60493 If the request can be honored, *signal()* shall return the value of *func* for the most recent call to
 60494 *signal()* for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value
 60495 shall be stored in *errno*.

60496 ERRORS

60497 The *signal()* function shall fail if:

60498 CX [EINVAL] The *sig* argument is not a valid signal number or an attempt is made to catch a
 60499 signal that cannot be caught or ignore a signal that cannot be ignored.

60500 The *signal()* function may fail if:

60501 CX [EINVAL] An attempt was made to set the action to SIG_DFL for a signal that cannot be
 60502 caught or ignored (or both).

60503 EXAMPLES

60504 None.

60505 APPLICATION USAGE

60506 The *sigaction()* function provides a more comprehensive and reliable mechanism for controlling
 60507 signals; new applications should use *sigaction()* rather than *signal()*.

60508 RATIONALE

60509 None.

60510 FUTURE DIRECTIONS

60511 None.

60512 SEE ALSO

60513 [Section 2.4](#) (on page 463), *exec*, *pause()*, *sigaction()*, *sigsuspend()*, *waitid()*

60514 XBD [<signal.h>](#)

60515 CHANGE HISTORY

60516 First released in Issue 1. Derived from Issue 1 of the SVID.

60517 Issue 5

60518 Moved from X/OPEN UNIX extension to BASE.

60519 The DESCRIPTION is updated to indicate that the *sigpause()* function restores the signal mask of
 60520 the process to its original state before returning.

60521 The RETURN VALUE section is updated to indicate that the *sigpause()* function suspends
 60522 execution of the process until a signal is received, whereupon it returns -1 and sets *errno* to
 60523 [EINTR].

60524 Issue 6

60525 Extensions beyond the ISO C standard are marked.

60526 The normative text is updated to avoid use of the term “must” for application requirements.

60527 The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

60528 References to the *wait3()* function are removed.

60529 The *sighold()*, *sigignore()*, *sigelse()*, and *sigset()* functions are split out onto their own reference
 60530 page.

60531 **NAME**

60532 signbit — test sign

60533 **SYNOPSIS**

60534 #include <math.h>

60535 int signbit(real-floating x);

60536 **DESCRIPTION**

60537 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 60538 conflict between the requirements described here and the ISO C standard is unintentional. This
 60539 volume of POSIX.1-200x defers to the ISO C standard.

60540 The *signbit()* macro shall determine whether the sign of its argument value is negative. NaNs,
 60541 zeros, and infinities have a sign bit.

60542 **RETURN VALUE**

60543 The *signbit()* macro shall return a non-zero value if and only if the sign of its argument value is
 60544 negative.

60545 **ERRORS**

60546 No errors are defined.

60547 **EXAMPLES**

60548 None.

60549 **APPLICATION USAGE**

60550 None.

60551 **RATIONALE**

60552 None.

60553 **FUTURE DIRECTIONS**

60554 None.

60555 **SEE ALSO**60556 *fpclassify(), isfinite(), isinf(), isnan(), isnormal()*

60557 XBD <math.h>

60558 **CHANGE HISTORY**

60559 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

sigpause()

60560 **NAME**
60561 sigpause — remove a signal from the signal mask and suspend the thread

SYNOPSIS

60562 OB XSI #include <signal.h>
60563
60564 int sigpause(int *sig*);

DESCRIPTION

60565 Refer to *sighold()*.
60566

60567 **NAME**
 60568 sigpending — examine pending signals

60569 **SYNOPSIS**

60570 CX `#include <signal.h>`
 60571 `int sigpending(sigset_t *set);`

60572 **DESCRIPTION**

60573 The *sigpending()* function shall store, in the location referenced by the *set* argument, the set of
 60574 signals that are blocked from delivery to the calling thread and that are pending on the process
 60575 or the calling thread.

60576 **RETURN VALUE**

60577 Upon successful completion, *sigpending()* shall return 0; otherwise, -1 shall be returned and
 60578 *errno* set to indicate the error.

60579 **ERRORS**

60580 No errors are defined.

60581 **EXAMPLES**

60582 None.

60583 **APPLICATION USAGE**

60584 None.

60585 **RATIONALE**

60586 None.

60587 **FUTURE DIRECTIONS**

60588 None.

60589 **SEE ALSO**

60590 *exec*, *pthread_sigmask()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*, *sigismember()*

60591 XBD [<signal.h>](#)

60592 **CHANGE HISTORY**

60593 First released in Issue 3.

60594 **Issue 5**

60595 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60596 **Issue 6**

60597 The SYNOPSIS is marked CX since the presence of this function in the `<signal.h>` header is an
 60598 extension over the ISO C standard.

60599 **NAME**
60600 sigprocmask — examine and change blocked signals

60601 **SYNOPSIS**

```
60602 CX #include <signal.h>  
60603 int sigprocmask(int how, const sigset_t *restrict set,  
60604 sigset_t *restrict oset);
```

60605 **DESCRIPTION**

60606 Refer to *pthread_sigmask()*.

60607 **NAME**
 60608 sigqueue — queue a signal to a process

60609 **SYNOPSIS**

```
60610 CX #include <signal.h>
60611 int sigqueue(pid_t pid, int signo, const union sigval value);
```

60612 **DESCRIPTION**

60613 The *sigqueue()* function shall cause the signal specified by *signo* to be sent with the value
 60614 specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking
 60615 is performed but no signal is actually sent. The null signal can be used to check the validity of
 60616 *pid*.

60617 The conditions required for a process to have permission to queue a signal to another process are
 60618 the same as for the *kill()* function.

60619 The *sigqueue()* function shall return immediately. If SA_SIGINFO is set for *signo* and if the
 60620 resources were available to queue the signal, the signal shall be queued and sent to the receiving
 60621 process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving
 60622 process; it is unspecified whether *value* shall be sent to the receiving process as a result of this
 60623 call.

60624 If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked
 60625 for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait()*
 60626 function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the
 60627 calling thread before the *sigqueue()* function returns. Should any multiple pending signals in the
 60628 range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one.
 60629 The selection order between realtime and non-realtime signals, or between multiple pending
 60630 non-realtime signals, is unspecified.

60631 **RETURN VALUE**

60632 Upon successful completion, the specified signal shall have been queued, and the *sigqueue()*
 60633 function shall return a value of zero. Otherwise, the function shall return a value of -1 and set
 60634 *errno* to indicate the error.

60635 **ERRORS**

60636 The *sigqueue()* function shall fail if:

60637 [EAGAIN] No resources are available to queue the signal. The process has already
 60638 queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or
 60639 a system-wide resource limit has been exceeded.

60640 [EINVAL] The value of the *signo* argument is an invalid or unsupported signal number.

60641 [EPERM] The process does not have the appropriate privilege to send the signal to the
 60642 receiving process.

60643 [ESRCH] The process *pid* does not exist.

EXAMPLES

60644 None.
60645

APPLICATION USAGE

60646 None.
60647

RATIONALE

60648 The *sigqueue()* function allows an application to queue a realtime signal to itself or to another
60649 process, specifying the application-defined value. This is common practice in realtime
60650 applications on existing realtime systems. It was felt that specifying another function in the
60651 *sig...* name space already carved out for signals was preferable to extending the interface to
60652 *kill()*.
60653

60654 Such a function became necessary when the put/get event function of the message queues was
60655 removed. It should be noted that the *sigqueue()* function implies reduced performance in a
60656 security-conscious implementation as the access permissions between the sender and receiver
60657 have to be checked on each send when the *pid* is resolved into a target process. Such access
60658 checks were necessary only at message queue open in the previous interface.

60659 The standard developers required that *sigqueue()* have the same semantics with respect to the
60660 null signal as *kill()*, and that the same permission checking be used. But because of the difficulty
60661 of implementing the “broadcast” semantic of *kill()* (for example, to process groups) and the
60662 interaction with resource allocation, this semantic was not adopted. The *sigqueue()* function
60663 queues a signal to a single process specified by the *pid* argument.

60664 The *sigqueue()* function can fail if the system has insufficient resources to queue the signal. An
60665 explicit limit on the number of queued signals that a process could send was introduced. While
60666 the limit is “per-sender”, this volume of POSIX.1-200x does not specify that the resources be part
60667 of the state of the sender. This would require either that the sender be maintained after exit until
60668 all signals that it had sent to other processes were handled or that all such signals that had not
60669 yet been acted upon be removed from the queue(s) of the receivers. This volume of
60670 POSIX.1-200x does not preclude this behavior, but an implementation that allocated queuing
60671 resources from a system-wide pool (with per-sender limits) and that leaves queued signals
60672 pending after the sender exits is also permitted.

FUTURE DIRECTIONS

60673 None.
60674

SEE ALSO

60675 [Section 2.8.1](#) (on page 476)
60676

60677 XBD [<signal.h>](#)

CHANGE HISTORY

60678 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
60679 POSIX Threads Extension.
60680

Issue 6

60681 The *sigqueue()* function is marked as part of the Realtime Signals Extension option.
60682

60683 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60684 implementation does not support the Realtime Signals Extension option.

Issue 7

60685 The *sigqueue()* function is moved from the Realtime Signals Extension option to the Base.
60686

60687 **NAME**
60688 sigrelse, sigset — signal management

60689 **SYNOPSIS**

60690 OB XSI `#include <signal.h>`
60691 `int sigrelse(int sig);`
60692 `void (*sigset(int sig, void (*disp)(int)))(int);`

60693 **DESCRIPTION**

60694 Refer to *sighold()*.

60695 **NAME**

60696 sigsetjmp — set jump point for a non-local goto

60697 **SYNOPSIS**

```
60698 CX #include <setjmp.h>
60699 int sigsetjmp(sigjmp_buf env, int savemask);
```

60700 **DESCRIPTION**60701 The *sigsetjmp()* function shall be equivalent to the *setjmp()* function, except as follows:

- 60702 • References to *setjmp()* are equivalent to *sigsetjmp()*.
- 60703 • References to *longjmp()* are equivalent to *siglongjmp()*.
- 60704 • If the value of the *savemask* argument is not 0, *sigsetjmp()* shall also save the current signal mask of the calling thread as part of the calling environment.

60706 **RETURN VALUE**

60707 If the return is from a successful direct invocation, *sigsetjmp()* shall return 0. If the return is from
 60708 a call to *siglongjmp()*, *sigsetjmp()* shall return a non-zero value.

60709 **ERRORS**

60710 No errors are defined.

60711 **EXAMPLES**

60712 None.

60713 **APPLICATION USAGE**

60714 The distinction between *setjmp()/longjmp()* and *sigsetjmp()/siglongjmp()* is only significant for
 60715 programs which use *sigaction()*, *sigprocmask()*, or *sigsuspend()*.

60716 Note that since this function is defined in terms of *setjmp()*, if *savemask* is zero, it is unspecified
 60717 whether the signal mask is saved.

60718 **RATIONALE**

60719 The ISO C standard specifies various restrictions on the usage of the *setjmp()* macro in order to
 60720 permit implementors to recognize the name in the compiler and not implement an actual
 60721 function. These same restrictions apply to the *sigsetjmp()* macro.

60722 There are processors that cannot easily support these calls, but this was not considered a
 60723 sufficient reason to exclude them.

60724 4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp()* and
 60725 *_longjmp()* that, together with *setjmp()* and *longjmp()*, provide the same functionality as
 60726 *sigsetjmp()* and *siglongjmp()*. On those systems, *setjmp()* and *longjmp()* save and restore signal
 60727 masks, while *_setjmp()* and *_longjmp()* do not. On System V Release 3 and in corresponding
 60728 issues of the SVID, *setjmp()* and *longjmp()* are explicitly defined not to save and restore signal
 60729 masks. In order to permit existing practice in both cases, the relation of *setjmp()* and *longjmp()* to
 60730 signal masks is not specified, and a new set of functions is defined instead.

60731 The *longjmp()* and *siglongjmp()* functions operate as in the previous issue provided the matching
 60732 *setjmp()* or *sigsetjmp()* has been performed in the same thread. Non-local jumps into contexts
 60733 saved by other threads would be at best a questionable practice and were not considered worthy
 60734 of standardization.

60735

FUTURE DIRECTIONS

60736

None.

60737

SEE ALSO

60738

pthread_sigmask(), *siglongjmp()*, *signal()*, *sigsuspend()*

60739

XBD <setjmp.h>

60740

CHANGE HISTORY

60741

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

60742

Issue 5

60743

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

60744

Issue 6

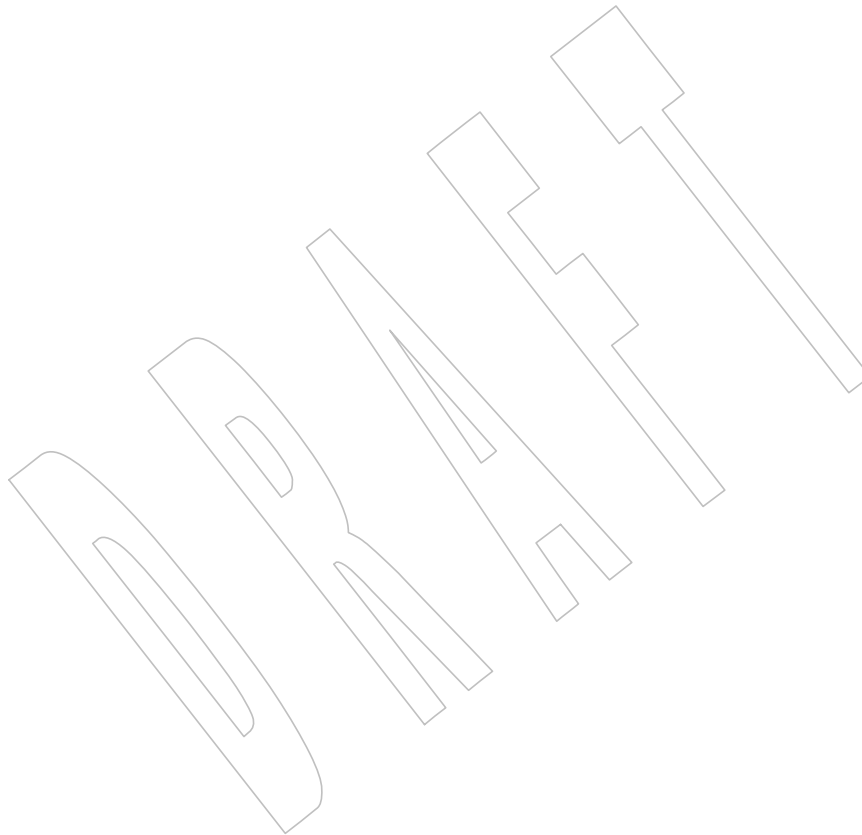
60745

The DESCRIPTION is reworded in terms of *setjmp()*.

60746

The SYNOPSIS is marked CX since the presence of this function in the <setjmp.h> header is an extension over the ISO C standard.

60747



60748 **NAME**

60749 sigsuspend — wait for a signal

60750 **SYNOPSIS**

```
60751 CX #include <signal.h>
60752 int sigsuspend(const sigset_t *sigmask);
```

60753 **DESCRIPTION**

60754 The *sigsuspend()* function shall replace the current signal mask of the calling thread with the set
 60755 of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose
 60756 action is either to execute a signal-catching function or to terminate the process. This shall not
 60757 cause any other signals that may have been pending on the process to become pending on the
 60758 thread.

60759 If the action is to terminate the process then *sigsuspend()* shall never return. If the action is to
 60760 execute a signal-catching function, then *sigsuspend()* shall return after the signal-catching
 60761 function returns, with the signal mask restored to the set that existed prior to the *sigsuspend()*
 60762 call.

60763 It is not possible to block signals that cannot be ignored. This is enforced by the system without
 60764 causing an error to be indicated.

60765 **RETURN VALUE**

60766 Since *sigsuspend()* suspends thread execution indefinitely, there is no successful completion
 60767 return value. If a return occurs, -1 shall be returned and *errno* set to indicate the error.

60768 **ERRORS**

60769 The *sigsuspend()* function shall fail if:

60770 [EINTR] A signal is caught by the calling process and control is returned from the
 60771 signal-catching function.

60772 **EXAMPLES**

60773 None.

60774 **APPLICATION USAGE**

60775 Normally, at the beginning of a critical code section, a specified set of signals is blocked using
 60776 the *sigprocmask()* function. When the thread has completed the critical section and needs to wait
 60777 for the previously blocked signal(s), it pauses by calling *sigsuspend()* with the mask that was
 60778 returned by the *sigprocmask()* call.

60779 **RATIONALE**

60780 Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers
 60781 can install an additional cancellation handler which resets the signal mask to the expected value.

```
60782 void cleanup(void *arg)
60783 {
60784     sigset_t *ss = (sigset_t *) arg;
60785     pthread_sigmask(SIG_SETMASK, ss, NULL);
60786 }
60787 int call_sigsuspend(const sigset_t *mask)
60788 {
60789     sigset_t oldmask;
60790     int result;
60791     pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
```

```

60792         pthread_cleanup_push(cleanup, &oldmask);
60793         result = sigsuspend(sigmask);
60794         pthread_cleanup_pop(0);
60795         return result;
60796     }

```

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4 (on page 463), *pause()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*, *sigfillset()*

XBD [<signal.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The text in the RETURN VALUE section has been changed from “suspends process execution” to “suspends thread execution”. This reflects IEEE PASC Interpretation 1003.1c #40.

Text in the APPLICATION USAGE section has been replaced.

The SYNOPSIS is marked CX since the presence of this function in the [<signal.h>](#) header is an extension over the ISO C standard.

Issue 7

SD5-XSH-ERN-122 is applied, adding the example code in the RATIONALE.

60814 **NAME**

60815 sigtimedwait, sigwaitinfo — wait for queued signals

60816 **SYNOPSIS**

```
60817 CX      #include <signal.h>
60818
60818      int sigtimedwait(const sigset_t *restrict set,
60819                    siginfo_t *restrict info,
60820                    const struct timespec *restrict timeout);
60821      int sigwaitinfo(const sigset_t *restrict set,
60822                    siginfo_t *restrict info);
```

60823 **DESCRIPTION**

60824 The *sigtimedwait()* function shall be equivalent to *sigwaitinfo()* except that if none of the signals
60825 specified by *set* are pending, *sigtimedwait()* shall wait for the time interval specified in the
60826 **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-
60827 valued and if none of the signals specified by *set* are pending, then *sigtimedwait()* shall return
60828 immediately with an error. If *timeout* is the NULL pointer, the behavior is unspecified. If the
60829 Monotonic Clock option is supported, the CLOCK_MONOTONIC clock shall be used to
60830 measure the time interval specified by the *timeout* argument.

60831 The *sigwaitinfo()* function selects the pending signal from the set specified by *set*. Should any of
60832 multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the
60833 lowest numbered one. The selection order between realtime and non-realtime signals, or
60834 between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at
60835 the time of the call, the calling thread shall be suspended until one or more signals in *set* become
60836 pending or until it is interrupted by an unblocked, caught signal.

60837 The *sigwaitinfo()* function shall be equivalent to the *sigwait()* function if the *info* argument is
60838 NULL. If the *info* argument is non-NULL, the *sigwaitinfo()* function shall be equivalent to
60839 *sigwait()*, except that the selected signal number shall be stored in the *si_signo* member, and the
60840 cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected
60841 signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the
60842 value shall be stored in the *si_value* member of *info*. The system resource used to queue the
60843 signal shall be released and returned to the system for other use. If no value is queued, the
60844 content of the *si_value* member is undefined. If no further signals are queued for the selected
60845 signal, the pending indication for that signal shall be reset.

60846 **RETURN VALUE**

60847 Upon successful completion (that is, one of the signals specified by *set* is pending or is
60848 generated) *sigwaitinfo()* and *sigtimedwait()* shall return the selected signal number. Otherwise,
60849 the function shall return a value of -1 and set *errno* to indicate the error.

60850 **ERRORS**

60851 The *sigtimedwait()* function shall fail if:

60852 [EAGAIN] No signal specified by *set* was generated within the specified timeout period.

60853 The *sigtimedwait()* and *sigwaitinfo()* functions may fail if:

60854 [EINTR] The wait was interrupted by an unblocked, caught signal. It shall be
60855 documented in system documentation whether this error causes these
60856 functions to fail.

60857 The *sigtimedwait()* function may also fail if:

60858 [EINVAL] The *timeout* argument specified a *tv_nsec* value less than zero or greater than
60859 or equal to 1 000 million.

60860 An implementation should only check for this error if no signal is pending in *set* and it is
60861 necessary to wait.

60862 EXAMPLES

60863 None.

60864 APPLICATION USAGE

60865 The *sigtimedwait()* function times out and returns an [EAGAIN] error. Application developers
60866 should note that this is inconsistent with other functions such as *pthread_cond_timedwait()* that
60867 return [ETIMEDOUT].

60868 RATIONALE

60869 Existing programming practice on realtime systems uses the ability to pause waiting for a
60870 selected set of events and handle the first event that occurs in-line instead of in a signal-handling
60871 function. This allows applications to be written in an event-directed style similar to a state
60872 machine. This style of programming is useful for largescale transaction processing in which the
60873 overall throughput of an application and the ability to clearly track states are more important
60874 than the ability to minimize the response time of individual event handling.

60875 It is possible to construct a signal-waiting macro function out of the realtime signal function
60876 mechanism defined in this volume of POSIX.1-200x. However, such a macro has to include the
60877 definition of a generalized handler for all signals to be waited on. A significant portion of the
60878 overhead of handler processing can be avoided if the signal-waiting function is provided by the
60879 kernel. This volume of POSIX.1-200x therefore provides two signal-waiting functions—one that
60880 waits indefinitely and one with a timeout—as part of the overall realtime signal function
60881 specification.

60882 The specification of a function with a timeout allows an application to be written that can be
60883 broken out of a wait after a set period of time if no event has occurred. It was argued that setting
60884 a timer event before the wait and recognizing the timer event in the wait would also implement
60885 the same functionality, but at a lower performance level. Because of the performance
60886 degradation associated with the user-level specification of a timer event and the subsequent
60887 cancellation of that timer event after the wait completes for a valid event, and the complexity
60888 associated with handling potential race conditions associated with the user-level method, the
60889 separate function has been included.

60890 Note that the semantics of the *sigwaitinfo()* function are nearly identical to that of the *sigwait()*
60891 function defined by this volume of POSIX.1-200x. The only difference is that *sigwaitinfo()* returns
60892 the queued signal value in the *value* argument. The return of the queued value is required so that
60893 applications can differentiate between multiple events queued to the same signal number.

60894 The two distinct functions are being maintained because some implementations may choose to
60895 implement the POSIX Threads Extension functions and not implement the queued signals
60896 extensions. Note, though, that *sigwaitinfo()* does not return the queued value if the *value*
60897 argument is NULL, so the POSIX Threads Extension *sigwait()* function can be implemented as a
60898 macro on *sigwaitinfo()*.

60899 The *sigtimedwait()* function was separated from the *sigwaitinfo()* function to address concerns
60900 regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed
60901 wait, and immediate return, and concerns regarding consistency with other functions where the
60902 conditional and timed waits were separate functions from the pure blocking function. The
60903 semantics of *sigtimedwait()* are specified such that *sigwaitinfo()* could be implemented as a macro
60904 with a NULL pointer for *timeout*.

60905 The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-

60906 generated signals. One important question was how many threads that are suspended in a call
 60907 to a *sigwait()* function for a signal should return from the call when the signal is sent. Four
 60908 choices were considered:

- 60909 1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
- 60910 2. One or more threads return.
- 60911 3. All waiting threads return.
- 60912 4. Exactly one thread returns.

60913 Prohibiting multiple calls to *sigwait()* for the same signal was felt to be overly restrictive. The
 60914 “one or more” behavior made implementation of conforming packages easy at the expense of
 60915 forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait()* in
 60916 application code in order to achieve predictable behavior. There was concern that the “all
 60917 waiting threads” behavior would result in “signal broadcast storms”, consuming excessive CPU
 60918 resources by replicating the signals in the general case. Furthermore, no convincing examples
 60919 could be presented that delivery to all was either simpler or more powerful than delivery to one.

60920 Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait*
 60921 function for a signal should return when that signal occurs. This is not an onerous restriction as:

- 60922 • A multi-way signal wait can be built from the single-way wait.
- 60923 • Signals should only be handled by application-level code, as library routines cannot guess
 60924 what the application wants to do with signals generated for the entire process.
- 60925 • Applications can thus arrange for a single thread to wait for any given signal and call any
 60926 needed routines upon its arrival.

60927 In an application that is using signals for interprocess communication, signal processing is
 60928 typically done in one place. Alternatively, if the signal is being caught so that process cleanup
 60929 can be done, the signal handler thread can call separate process cleanup routines for each
 60930 portion of the application. Since the application main line started each portion of the application,
 60931 it is at the right abstraction level to tell each portion of the application to clean up.

60932 Certainly, there exist programming styles where it is logical to consider waiting for a single
 60933 signal in multiple threads. A simple *sigwait_multiple()* routine can be constructed to achieve this
 60934 goal. A possible implementation would be to have each *sigwait_multiple()* caller registered as
 60935 having expressed interest in a set of signals. The caller then waits on a thread-specific condition
 60936 variable. A single server thread calls a *sigwait()* function on the union of all registered signals.
 60937 When the *sigwait()* function returns, the appropriate state is set and condition variables are
 60938 broadcast. New *sigwait_multiple()* callers may cause the pending *sigwait()* call to be canceled
 60939 and reissued in order to update the set of signals being waited for.

60940 FUTURE DIRECTIONS

60941 None.

60942 SEE ALSO

60943 [Section 2.8.1](#) (on page 476), [pause\(\)](#), [pthread_sigmask\(\)](#), [sigaction\(\)](#), [sigpending\(\)](#), [sigsuspend\(\)](#),
 60944 [sigwait\(\)](#)

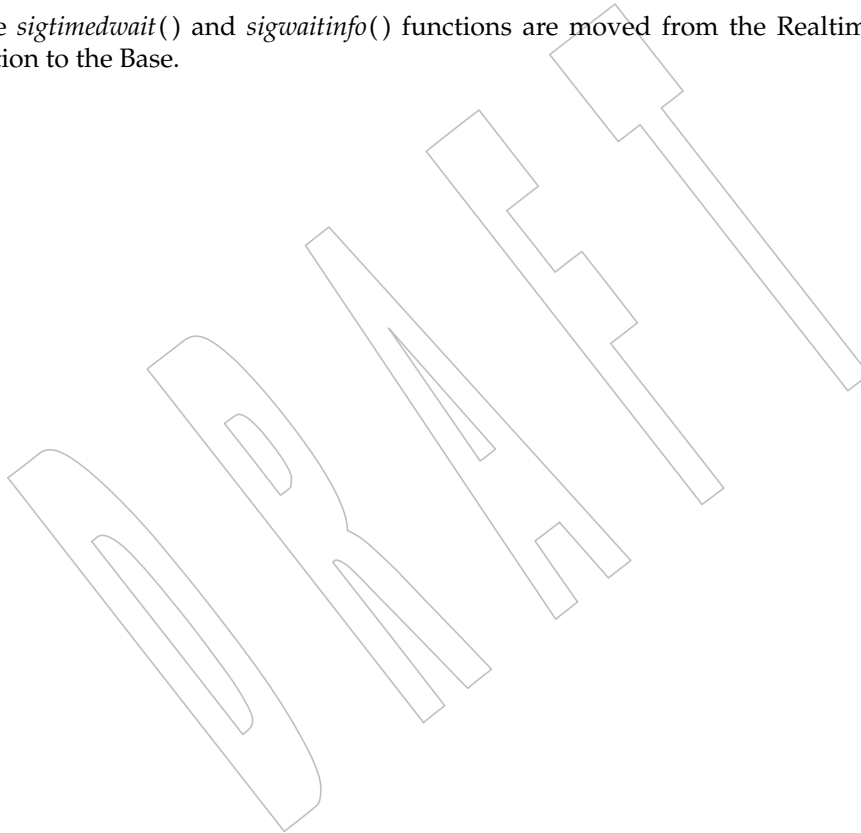
60945 XBD [<signal.h>](#), [<time.h>](#)

60946 CHANGE HISTORY

60947 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
 60948 POSIX Threads Extension.

60949 **Issue 6**

60950 These functions are marked as part of the Realtime Signals Extension option.

60951 The Open Group Corrigendum U035/3 is applied. The SYNOPSIS of the *sigwaitinfo()* function
60952 has been corrected so that the second argument is of type **siginfo_t** *.60953 The [ENOSYS] error condition has been removed as stubs need not be provided if an
60954 implementation does not support the Realtime Signals Extension option.60955 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that the
60956 CLOCK_MONOTONIC clock, if supported, is used to measure timeout intervals.60957 The **restrict** keyword is added to the *sigtimedwait()* and *sigwaitinfo()* prototypes for alignment
60958 with the ISO/IEC 9899:1999 standard.60959 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/130 is applied, restoring wording in the
60960 RETURN VALUE section to that in the original base document (“An implementation should
60961 only check for this error if no signal is pending in *set* and it is necessary to wait”).60962 **Issue 7**60963 The *sigtimedwait()* and *sigwaitinfo()* functions are moved from the Realtime Signals Extension
60964 option to the Base.

60965 **NAME**60966 `sigwait` — wait for queued signals60967 **SYNOPSIS**

```
60968 CX #include <signal.h>
60969 int sigwait(const sigset_t *restrict set, int *restrict sig);
```

60970 **DESCRIPTION**

60971 The `sigwait()` function shall select a pending signal from `set`, atomically clear it from the system's
 60972 set of pending signals, and return that signal number in the location referenced by `sig`. If prior to
 60973 the call to `sigwait()` there are multiple pending instances of a single signal number, it is
 60974 implementation-defined whether upon successful return there are any remaining pending
 60975 signals for that signal number. If the implementation supports queued signals and there are
 60976 multiple signals queued for the signal number selected, the first such queued signal shall cause a
 60977 return from `sigwait()` and the remainder shall remain queued. If no signal in `set` is pending at the
 60978 time of the call, the thread shall be suspended until one or more becomes pending. The signals
 60979 defined by `set` shall have been blocked at the time of the call to `sigwait()`; otherwise, the behavior
 60980 is undefined. The effect of `sigwait()` on the signal actions for the signals in `set` is unspecified.

60981 If more than one thread is using `sigwait()` to wait for the same signal, no more than one of these
 60982 threads shall return from `sigwait()` with the signal number. If more than a single thread is
 60983 blocked in `sigwait()` for a signal when that signal is generated for the process, it is unspecified
 60984 which of the waiting threads returns from `sigwait()`. If the signal is generated for a specific
 60985 thread, as by `pthread_kill()`, only that thread shall return.

60986 Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it
 60987 shall be the lowest numbered one. The selection order between realtime and non-realtime
 60988 signals, or between multiple pending non-realtime signals, is unspecified.

60989 **RETURN VALUE**

60990 Upon successful completion, `sigwait()` shall store the signal number of the received signal at the
 60991 location referenced by `sig` and return zero. Otherwise, an error number shall be returned to
 60992 indicate the error.

60993 **ERRORS**60994 The `sigwait()` function may fail if:60995 [EINVAL] The `set` argument contains an invalid or unsupported signal number.60996 **EXAMPLES**

60997 None.

60998 **APPLICATION USAGE**

60999 None.

61000 **RATIONALE**

61001 To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-200x
 61002 provides the `sigwait()` function. For most cases where a thread has to wait for a signal, the
 61003 `sigwait()` function should be quite convenient, efficient, and adequate.

61004 However, requests were made for a lower-level primitive than `sigwait()` and for semaphores that
 61005 could be used by threads. After some consideration, threads were allowed to use semaphores
 61006 and `sem_post()` was defined to be async-signal and async-cancel-safe.

61007 In summary, when it is necessary for code run in response to an asynchronous signal to notify a
 61008 thread, `sigwait()` should be used to handle the signal. Alternatively, if the implementation

61009 provides semaphores, they also can be used, either following *sigwait()* or from within a signal
 61010 handling routine previously registered with *sigaction()*.

61011 FUTURE DIRECTIONS

61012 None.

61013 SEE ALSO

61014 [Section 2.4](#) (on page 463), [Section 2.8.1](#) (on page 476), [pause\(\)](#), [pthread_sigmask\(\)](#), [sigaction\(\)](#),
 61015 [sigpending\(\)](#), [sigsuspend\(\)](#), [sigtimedwait\(\)](#)

61016 XBD [<signal.h>](#), [<time.h>](#)

61017 CHANGE HISTORY

61018 First released in Issue 5. Included for alignment with the POSIX Realtime Extension and the
 61019 POSIX Threads Extension.

61020 Issue 6

61021 The **restrict** keyword is added to the *sigwait()* prototype for alignment with the
 61022 ISO/IEC 9899:1999 standard.

61023 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/131 is applied, updating the
 61024 DESCRIPTION to state that if more than a single thread is blocked in *sigwait()*, it is unspecified
 61025 which of the waiting threads returns, and that if a signal is generated for a specific thread only
 61026 that thread shall return.

61027 Issue 7

61028 Functionality relating to the Realtime Signals Extension option is moved to the Base.

sigwaitinfo()*System Interfaces*

61029 **NAME**
61030 sigwaitinfo — wait for queued signals

61031 **SYNOPSIS**

```
61032 #include <signal.h>  
61033 int sigwaitinfo(const sigset_t *restrict set, siginfo_t *restrict info);
```

61034 **DESCRIPTION**

61035 Refer to [sigtimedwait\(\)](#).

61036 **NAME**61037 `sin, sinf, sinl` — sine function61038 **SYNOPSIS**

```
61039 #include <math.h>
61040 double sin(double x);
61041 float sinf(float x);
61042 long double sinl(long double x);
```

61043 **DESCRIPTION**

61044 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 61045 conflict between the requirements described here and the ISO C standard is unintentional. This
 61046 volume of POSIX.1-200x defers to the ISO C standard.

61047 These functions shall compute the sine of their argument x , measured in radians.

61048 An application wishing to check for error situations should set *errno* to zero and call
 61049 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 61050 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 61051 zero, an error has occurred.

61052 **RETURN VALUE**

61053 Upon successful completion, these functions shall return the sine of x .

61054 MX If x is NaN, a NaN shall be returned.

61055 If x is ± 0 , x shall be returned.

61056 If x is subnormal, a range error may occur and x should be returned.

61057 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 61058 defined value shall be returned.

61059 **ERRORS**

61060 These functions shall fail if:

61061 MX **Domain Error** The x argument is $\pm\text{Inf}$.

61062 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 61063 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 61064 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 61065 shall be raised.

61066 These functions may fail if:

61067 MX **Range Error** The value of x is subnormal

61068 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 61069 then *errno* shall be set to [ERANGE]. If the integer expression
 61070 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 61071 floating-point exception shall be raised.

61072 **EXAMPLES**61073 **Taking the Sine of a 45-Degree Angle**

```

61074 #include <math.h>
61075 ...
61076 double radians = 45.0 * M_PI / 180;
61077 double result;
61078 ...
61079 result = sin(radians);

```

61080 **APPLICATION USAGE**

61081 These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

61082 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
61083 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

61084 **RATIONALE**

61085 None.

61086 **FUTURE DIRECTIONS**

61087 None.

61088 **SEE ALSO**

61089 *asin()*, *feclearexcept()*, *fetetestexcept()*, *isnan()*

61090 XBD Section 4.19 (on page 104), **<math.h>**

61091 **CHANGE HISTORY**

61092 First released in Issue 1. Derived from Issue 1 of the SVID.

61093 **Issue 5**

61094 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
61095 in previous issues.

61096 **Issue 6**

61097 The *sinf()* and *sinl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

61098 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
61099 revised to align with the ISO/IEC 9899:1999 standard.

61100 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
61101 marked.

61102 **NAME**

61103 sinh, sinhf, sinhl — hyperbolic sine functions

61104 **SYNOPSIS**

```
61105     #include <math.h>
61106
61106     double sinh(double x);
61107     float  sinhf(float x);
61108     long double sinhl(long double x);
```

61109 **DESCRIPTION**

61110 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
61111 conflict between the requirements described here and the ISO C standard is unintentional. This
61112 volume of POSIX.1-200x defers to the ISO C standard.

61113 These functions shall compute the hyperbolic sine of their argument x .

61114 An application wishing to check for error situations should set *errno* to zero and call
61115 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
61116 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
61117 zero, an error has occurred.

61118 **RETURN VALUE**

61119 Upon successful completion, these functions shall return the hyperbolic sine of x .

61120 If the result would cause an overflow, a range error shall occur and \pm HUGE_VAL,
61121 \pm HUGE_VALF, and \pm HUGE_VALL (with the same sign as x) shall be returned as appropriate for
61122 the type of the function.

61123 **MX** If x is NaN, a NaN shall be returned.

61124 If x is ± 0 or \pm Inf, x shall be returned.

61125 If x is subnormal, a range error may occur and x should be returned.

61126 **ERRORS**

61127 These functions shall fail if:

61128 Range Error The result would cause an overflow.

61129 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
61130 then *errno* shall be set to [ERANGE]. If the integer expression
61131 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
61132 floating-point exception shall be raised.

61133 These functions may fail if:

61134 **MX** Range Error The value x is subnormal.

61135 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
61136 then *errno* shall be set to [ERANGE]. If the integer expression
61137 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
61138 floating-point exception shall be raised.

61139
61140

61141
61142
61143

61144
61145

61146
61147

61148
61149
61150

61151
61152

61153
61154
61155

61156
61157
61158
61159
61160
61161

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asinh(), *cosh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tanh()*

XBD Section 4.19 (on page 104), **<math.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *sinhf()* and *sinhl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

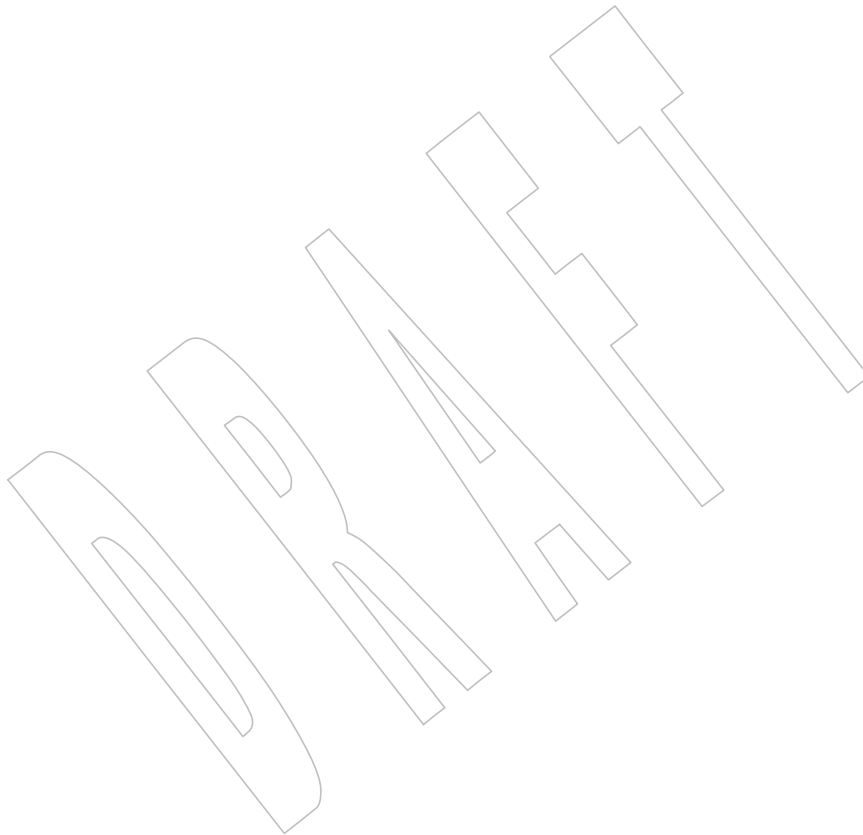
The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.

61162 **NAME**
61163 sinl — sine function

61164 **SYNOPSIS**
61165 #include <math.h>
61166 long double sinl(long double x);

61167 **DESCRIPTION**
61168 Refer to *sin()*.



61169 **NAME**61170 `sleep` — suspend execution for an interval of time61171 **SYNOPSIS**61172 `#include <unistd.h>`61173 `unsigned sleep(unsigned seconds);`61174 **DESCRIPTION**

61175 The `sleep()` function shall cause the calling thread to be suspended from execution until either
 61176 the number of realtime seconds specified by the argument `seconds` has elapsed or a signal is
 61177 delivered to the calling thread and its action is to invoke a signal-catching function or to
 61178 terminate the process. The suspension time may be longer than requested due to the scheduling
 61179 of other activity by the system.

61180 If a SIGALRM signal is generated for the calling process during execution of `sleep()` and if the
 61181 SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether `sleep()`
 61182 returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also
 61183 unspecified whether it remains pending after `sleep()` returns or it is discarded.

61184 If a SIGALRM signal is generated for the calling process during execution of `sleep()`, except as a
 61185 result of a prior call to `alarm()`, and if the SIGALRM signal is not being ignored or blocked from
 61186 delivery, it is unspecified whether that signal has any effect other than causing `sleep()` to return.

61187 If a signal-catching function interrupts `sleep()` and examines or changes either the time a
 61188 SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or
 61189 whether the SIGALRM signal is blocked from delivery, the results are unspecified.

61190 If a signal-catching function interrupts `sleep()` and calls `siglongjmp()` or `longjmp()` to restore an
 61191 environment saved prior to the `sleep()` call, the action associated with the SIGALRM signal and
 61192 the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also
 61193 unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is
 61194 restored as part of the environment.

61195 XSI Interactions between `sleep()` and `setitimer()` are unspecified.61196 **RETURN VALUE**

61197 If `sleep()` returns because the requested time has elapsed, the value returned shall be 0. If `sleep()`
 61198 returns due to delivery of a signal, the return value shall be the “unslept” amount (the requested
 61199 time minus the time actually slept) in seconds.

61200 **ERRORS**

61201 No errors are defined.

61202 **EXAMPLES**

61203 None.

61204 **APPLICATION USAGE**

61205 None.

61206 **RATIONALE**

61207 There are two general approaches to the implementation of the `sleep()` function. One is to use the
 61208 `alarm()` function to schedule a SIGALRM signal and then suspend the calling thread waiting for
 61209 that signal. The other is to implement an independent facility. This volume of POSIX.1-200x
 61210 permits either approach.

61211 In order to comply with the requirement that no primitive shall change a process attribute unless
 61212 explicitly described by this volume of POSIX.1-200x, an implementation using SIGALRM must
 61213 carefully take into account any SIGALRM signal scheduled by previous `alarm()` calls, the action

61214 previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has
 61215 been scheduled before the *sleep()* would ordinarily complete, the *sleep()* must be shortened to
 61216 that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-
 61217 catching function) before *sleep()* returns. If a SIGALRM has been scheduled after the *sleep()*
 61218 would ordinarily complete, it must be rescheduled for the same time before *sleep()* returns. The
 61219 action and blocking for SIGALRM must be saved and restored.

61220 Historical implementations often implement the SIGALRM-based version using *alarm()* and
 61221 *pause()*. One such implementation is prone to infinite hangups, as described in *pause()*.
 61222 Another such implementation uses the C-language *setjmp()* and *longjmp()* functions to avoid
 61223 that window. That implementation introduces a different problem: when the SIGALRM signal
 61224 interrupts a signal-catching function installed by the user to catch a different signal, the
 61225 *longjmp()* aborts that signal-catching function. An implementation based on *sigprocmask()*,
 61226 *alarm()*, and *sigsuspend()* can avoid these problems.

61227 Despite all reasonable care, there are several very subtle, but detectable and unavoidable,
 61228 differences between the two types of implementations. These are the cases mentioned in this
 61229 volume of POSIX.1-200x where some other activity relating to SIGALRM takes place, and the
 61230 results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of
 61231 concern to most applications.

61232 See also the discussion of the term *realtime* in *alarm()*.

61233 Since *sleep()* can be implemented using *alarm()*, the discussion about alarms occurring early
 61234 under *alarm()* applies to *sleep()* as well.

61235 Application developers should note that the type of the argument *seconds* and the return value of
 61236 *sleep()* is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application
 61237 cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the
 61238 ISO C standard sets as 65535, and any application passing a larger value is restricting its
 61239 portability. A different type was considered, but historical implementations, including those
 61240 with a 16-bit **int** type, consistently use either **unsigned** or **int**.

61241 Scheduling delays may cause the process to return from the *sleep()* function significantly after
 61242 the requested time. In such cases, the return value should be set to zero, since the formula
 61243 (requested time minus the time actually spent) yields a negative number and *sleep()* returns an
 61244 **unsigned**.

61245 FUTURE DIRECTIONS

61246 None.

61247 SEE ALSO

61248 *alarm()*, *getitimer()*, *nanosleep()*, *pause()*, *sigaction()*, *sigsetjmp()*

61249 XBD <**unistd.h**>

61250 CHANGE HISTORY

61251 First released in Issue 1. Derived from Issue 1 of the SVID.

61252 Issue 5

61253 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

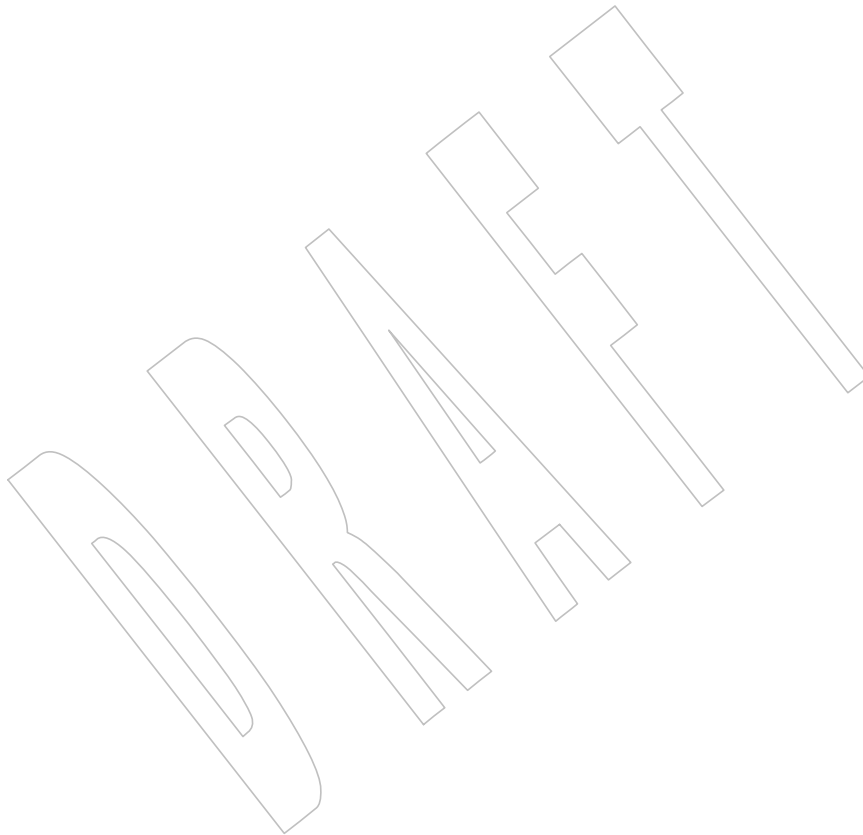
61254 Issue 6

61255 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/132 is applied, making a correction in the
 61256 RATIONALE section.

61257 **NAME**
61258 **snprintf** — print formatted output

61259 **SYNOPSIS**
61260 #include <stdio.h>
61261 int snprintf(char *restrict *s*, size_t *n*,
61262 const char *restrict *format*, ...);

61263 **DESCRIPTION**
61264 Refer to *fprintf()*.



61265 **NAME**

61266 socketmark — determine whether a socket is at the out-of-band mark

61267 **SYNOPSIS**61268 #include <sys/socket.h>
61269 int socketmark(int s);61270 **DESCRIPTION**61271 The *socketmark()* function shall determine whether the socket specified by the descriptor *s* is at
61272 the out-of-band data mark (see [Section 2.10.12](#), on page 499). If the protocol for the socket
61273 supports out-of-band data by marking the stream with an out-of-band data mark, the
61274 *socketmark()* function shall return 1 when all data preceding the mark has been read and the out-
61275 of-band data mark is the first element in the receive queue. The *socketmark()* function shall not
61276 remove the mark from the stream.61277 **RETURN VALUE**61278 Upon successful completion, the *socketmark()* function shall return a value indicating whether
61279 the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data
61280 preceding the mark has been read, the return value shall be 1; if there is no mark, or if data
61281 precedes the mark in the receive queue, the *socketmark()* function shall return 0. Otherwise, it
61282 shall return a value of -1 and set *errno* to indicate the error.61283 **ERRORS**61284 The *socketmark()* function shall fail if:

- 61285 [EBADF] The
- s*
- argument is not a valid file descriptor.
-
- 61286 [ENOTTY] The file associated with the
- s*
- argument is not a socket.

61287 **EXAMPLES**

61288 None.

61289 **APPLICATION USAGE**61290 The use of this function between receive operations allows an application to determine which
61291 received data precedes the out-of-band data and which follows the out-of-band data.61292 There is an inherent race condition in the use of this function. On an empty receive queue, the
61293 current read of the location might well be at the “mark”, but the system has no way of knowing
61294 that the next data segment that will arrive from the network will carry the mark, and
61295 *socketmark()* will return false, and the next read operation will silently consume the mark.61296 Hence, this function can only be used reliably when the application already knows that the out-
61297 of-band data has been seen by the system or that it is known that there is data waiting to be read
61298 at the socket (via SIGURG or *select()*). See [Section 2.10.11](#) (on page 498), [Section 2.10.12](#) (on page
61299 499), [Section 2.10.14](#) (on page 499), and *pselect()* for details.61300 **RATIONALE**61301 The *socketmark()* function replaces the historical SIOCATMARK command to *ioctl()* which
61302 implemented the same functionality on many implementations. Using a wrapper function
61303 follows the adopted conventions to avoid specifying commands to the *ioctl()* function, other
61304 than those now included to support XSI STREAMS. The *socketmark()* function could be
61305 implemented as follows:61306 #include <sys/ioctl.h>
61307 int socketmark(int s)
61308 {
61309 int val;

socketmark()

```
61310         if (ioctl(s,SIOCATMARK,&val)==-1)
61311             return(-1);
61312         return(val);
61313     }
```

61314 The use of [ENOTTY] to indicate an incorrect descriptor type matches the historical behavior of
61315 SIOCATMARK.

FUTURE DIRECTIONS

61316 None.

SEE ALSO

61318 [Section 2.10.12](#) (on page 499), [pselect\(\)](#), [recv\(\)](#), [recvmsg\(\)](#)

61320 XBD [<sys/socket.h>](#)

CHANGE HISTORY

61321 First released in Issue 6. Derived from IEEE Std 1003.1g-2000.

Issue 7

61323 SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.
61324

61325 **NAME**
 61326 socket — create an endpoint for communication

61327 **SYNOPSIS**
 61328 #include <sys/socket.h>

61329 int socket(int domain, int type, int protocol);

61330 DESCRIPTION

61331 The *socket()* function shall create an unbound socket in a communications domain, and return a
 61332 file descriptor that can be used in later function calls that operate on sockets.

61333 The *socket()* function takes the following arguments:

61334 *domain* Specifies the communications domain in which a socket is to be created.

61335 *type* Specifies the type of socket to be created.

61336 *protocol* Specifies a particular protocol to be used with the socket. Specifying a *protocol*
 61337 of 0 causes *socket()* to use an unspecified default protocol appropriate for the
 61338 requested socket type.

61339 The *domain* argument specifies the address family used in the communications domain. The
 61340 address families supported by the system are implementation-defined.

61341 Symbolic constants that can be used for the domain argument are defined in the <sys/socket.h>
 61342 header.

61343 The *type* argument specifies the socket type, which determines the semantics of communication
 61344 over the socket. The following socket types are defined; implementations may specify additional
 61345 socket types:

61346 SOCK_STREAM Provides sequenced, reliable, bidirectional, connection-mode byte
 61347 streams, and may provide a transmission mechanism for out-of-band
 61348 data.

61349 SOCK_DGRAM Provides datagrams, which are connectionless-mode, unreliable messages
 61350 of fixed maximum length.

61351 SOCK_SEQPACKET Provides sequenced, reliable, bidirectional, connection-mode transmission
 61352 paths for records. A record can be sent using one or more output
 61353 operations and received using one or more input operations, but a single
 61354 operation never transfers part of more than one record. Record
 61355 boundaries are visible to the receiver via the MSG_EOR flag.

61356 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 61357 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 61358 be used. The protocols supported by the system are implementation-defined.

61359 The process may need to have appropriate privileges to use the *socket()* function or to create
 61360 some sockets.

61361 RETURN VALUE

61362 Upon successful completion, *socket()* shall return a non-negative integer, the socket file
 61363 descriptor. Otherwise, a value of -1 shall be returned and *errno* set to indicate the error.

socket()**61364 ERRORS**

61365 The *socket()* function shall fail if:

61366 [EAFNOSUPPORT]

61367 The implementation does not support the specified address family.

61368 [EMFILE]

All file descriptors available to the process are currently open.

61369 [ENFILE]

No more file descriptors are available for the system.

61370 [EPROTONOSUPPORT]

61371 The protocol is not supported by the address family, or the protocol is not
61372 supported by the implementation.

61373 [EPROTOTYPE] The socket type is not supported by the protocol.

61374 The *socket()* function may fail if:

61375 [EACCES]

The process does not have appropriate privileges.

61376 [ENOBUFS]

Insufficient resources were available in the system to perform the operation.

61377 [ENOMEM]

Insufficient memory was available to fulfill the request.

61378 EXAMPLES

61379 None.

61380 APPLICATION USAGE

61381 The documentation for specific address families specifies which protocols each address family
61382 supports. The documentation for specific protocols specifies which socket types each protocol
61383 supports.

61384 The application can determine whether an address family is supported by trying to create a
61385 socket with *domain* set to the protocol in question.

61386 RATIONALE

61387 None.

61388 FUTURE DIRECTIONS

61389 None.

61390 SEE ALSO

61391 *accept()*, *bind()*, *connect()*, *getsockname()*, *getsockopt()*, *listen()*, *recv()*, *recvfrom()*, *recvmsg()*,
61392 *send()*, *sendmsg()*, *setsockopt()*, *shutdown()*, *socketpair()*

61393 XBD <[netinet/in.h](#)>, <[sys/socket.h](#)>

61394 CHANGE HISTORY

61395 First released in Issue 6. Derived from the XNS, Issue 5.2 specification.

61396 **NAME**

61397 socketpair — create a pair of connected sockets

61398 **SYNOPSIS**

```
61399 #include <sys/socket.h>
61400 int socketpair(int domain, int type, int protocol,
61401               int socket_vector[2]);
```

61402 **DESCRIPTION**

61403 The *socketpair()* function shall create an unbound pair of connected sockets in a specified *domain*,
 61404 of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two
 61405 sockets shall be identical. The file descriptors used in referencing the created sockets shall be
 61406 returned in *socket_vector*[0] and *socket_vector*[1].

61407 The *socketpair()* function takes the following arguments:

61408	<i>domain</i>	Specifies the communications domain in which the sockets are to be created.
61409	<i>type</i>	Specifies the type of sockets to be created.
61410	<i>protocol</i>	Specifies a particular protocol to be used with the sockets. Specifying a 61411 <i>protocol</i> of 0 causes <i>socketpair()</i> to use an unspecified default protocol 61412 appropriate for the requested socket type.
61413	<i>socket_vector</i>	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

61414 The *type* argument specifies the socket type, which determines the semantics of communications
 61415 over the socket. The following socket types are defined; implementations may specify additional
 61416 socket types:

61417	SOCK_STREAM	Provides sequenced, reliable, bidirectional, connection-mode byte 61418 streams, and may provide a transmission mechanism for out-of-band 61419 data.
61420	SOCK_DGRAM	Provides datagrams, which are connectionless-mode, unreliable messages 61421 of fixed maximum length.
61422	SOCK_SEQPACKET	Provides sequenced, reliable, bidirectional, connection-mode transmission 61423 paths for records. A record can be sent using one or more output 61424 operations and received using one or more input operations, but a single 61425 operation never transfers part of more than one record. Record 61426 boundaries are visible to the receiver via the MSG_EOR flag.

61427 If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address
 61428 family. If the *protocol* argument is zero, the default protocol for this address family and type shall
 61429 be used. The protocols supported by the system are implementation-defined.

61430 The process may need to have appropriate privileges to use the *socketpair()* function or to create
 61431 some sockets.

61432 **RETURN VALUE**

61433 Upon successful completion, this function shall return 0; otherwise, -1 shall be returned and
 61434 *errno* set to indicate the error.

61435 **ERRORS**

61436 The *socketpair()* function shall fail if:

socketpair()

61437	[EAFNOSUPPORT]		
61438		The implementation does not support the specified address family.	
61439	[EMFILE]	All, or all but one, of the file descriptors available to the process are currently	
61440		open.	
61441	[ENFILE]	No more file descriptors are available for the system.	
61442	[EOPNOTSUPP]	The specified protocol does not permit creation of socket pairs.	
61443	[EPROTONOSUPPORT]		
61444		The protocol is not supported by the address family, or the protocol is not	
61445		supported by the implementation.	
61446	[EPROTOTYPE]	The socket type is not supported by the protocol.	
61447		The <i>socketpair()</i> function may fail if:	
61448	[EACCES]	The process does not have appropriate privileges.	
61449	[ENOBUFS]	Insufficient resources were available in the system to perform the operation.	
61450	[ENOMEM]	Insufficient memory was available to fulfill the request.	
61451	EXAMPLES		
61452		None.	
61453	APPLICATION USAGE		
61454		The documentation for specific address families specifies which protocols each address family	
61455		supports. The documentation for specific protocols specifies which socket types each protocol	
61456		supports.	
61457		The <i>socketpair()</i> function is used primarily with UNIX domain sockets and need not be	
61458		supported for other domains.	
61459	RATIONALE		
61460		None.	
61461	FUTURE DIRECTIONS		
61462		None.	
61463	SEE ALSO		
61464		socket()	
61465		XBD <sys/socket.h>	
61466	CHANGE HISTORY		
61467		First released in Issue 6. Derived from the XNS, Issue 5.2 specification.	
61468	Issue 7		
61469		The description of the [EMFILE] error condition is aligned with the <i>pipe()</i> function.	+

61470 **NAME**
61471 sprintf — print formatted output

61472 **SYNOPSIS**
61473 #include <stdio.h>
61474 int sprintf(char *restrict *s*, const char *restrict *format*, ...);

61475 **DESCRIPTION**
61476 Refer to *fprintf()*.



61477 **NAME**
 61478 sqrt, sqrtf, sqrtl — square root function

61479 **SYNOPSIS**
 61480 #include <math.h>
 61481 double sqrt(double x);
 61482 float sqrtf(float x);
 61483 long double sqrtl(long double x);

61484 **DESCRIPTION**
 61485 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 61486 conflict between the requirements described here and the ISO C standard is unintentional. This
 61487 volume of POSIX.1-200x defers to the ISO C standard.

61488 These functions shall compute the square root of their argument x , \sqrt{x} .

61489 An application wishing to check for error situations should set *errno* to zero and call
 61490 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 61491 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 61492 zero, an error has occurred.

61493 **RETURN VALUE**
 61494 Upon successful completion, these functions shall return the square root of x .

61495 MX For finite values of $x < -0$, a domain error shall occur, and either a NaN (if supported), or an
 61496 implementation-defined value shall be returned.

61497 MX If x is NaN, a NaN shall be returned.

61498 If x is ± 0 or $+\text{Inf}$, x shall be returned.

61499 If x is $-\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 61500 defined value shall be returned.

61501 **ERRORS**
 61502 These functions shall fail if:

61503 MX Domain Error The finite value of x is < -0 , or x is $-\text{Inf}$.

61504 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 61505 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 61506 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 61507 shall be raised.

61508 EXAMPLES

61509 Taking the Square Root of 9.0

```
61510 #include <math.h>
61511 ...
61512 double x = 9.0;
61513 double result;
61514 ...
61515 result = sqrt(x);
```

61516
61517
61518

61519
61520

61521
61522

61523
61524
61525

61526
61527

61528
61529
61530

61531
61532
61533
61534

61535
61536

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept(), *fetestexcept()*, *isnan()*

XBD Section 4.19 (on page 104), `<math.h>`, `<stdio.h>`

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

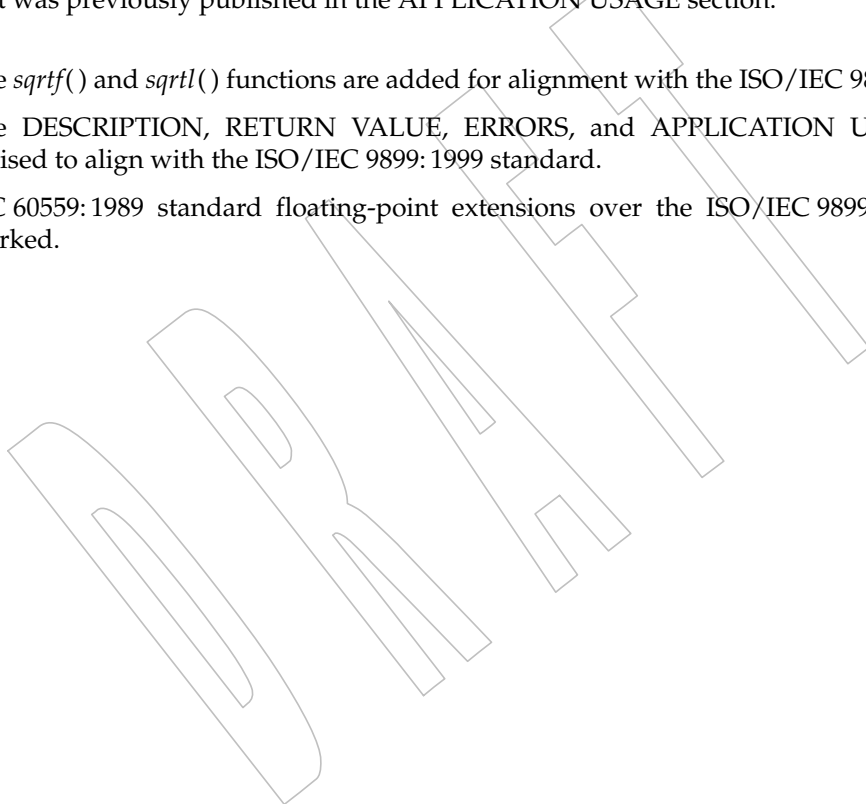
The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *sqrftf()* and *sqrftl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

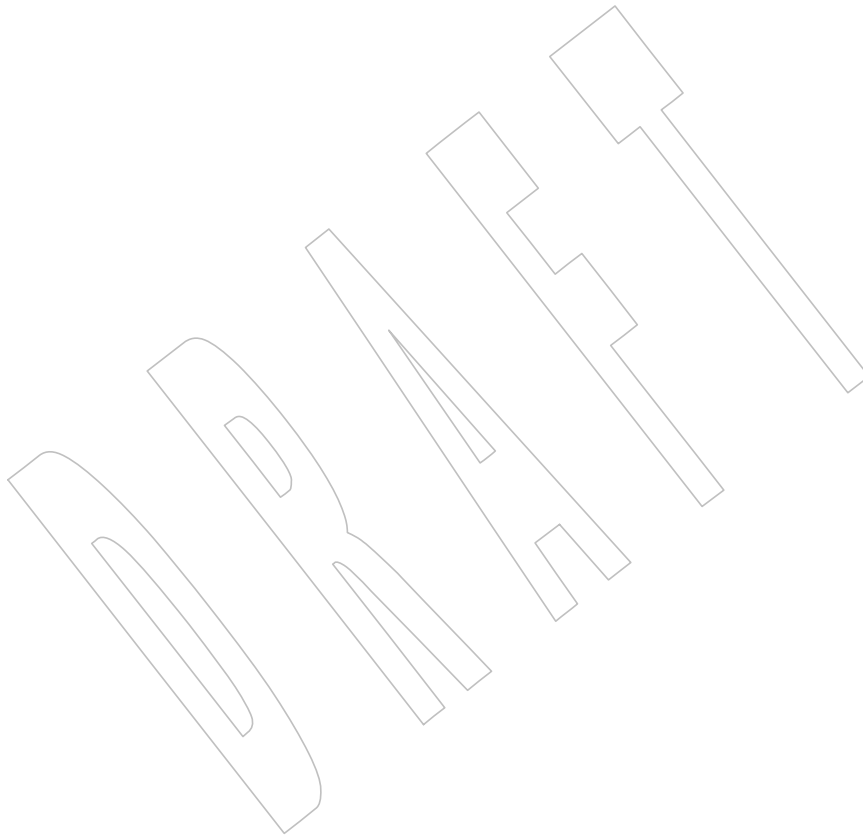
IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.



61537 **NAME**
61538 `srand` — pseudo-random number generator

61539 **SYNOPSIS**
61540 `#include <stdlib.h>`
61541 `void srand(unsigned seed);`

61542 **DESCRIPTION**
61543 Refer to *rand()*.



61544 **NAME**
61545 `srand48` — seed the uniformly distributed double-precision pseudo-random number generator

61546 **SYNOPSIS**

```
61547 XSI #include <stdlib.h>  
61548 void srand48(long seedval);
```

61549 **DESCRIPTION**

61550 Refer to [drand48\(\)](#).

srandom()

61551 **NAME**
61552 `srandom` — seed pseudo-random number generator

SYNOPSIS

61553 XSI `#include <stdlib.h>`
61554 `void srandom(unsigned seed);`

DESCRIPTION

61556 Refer to *initstate()*.
61557

61558 **NAME**
61559 scanf — convert formatted input

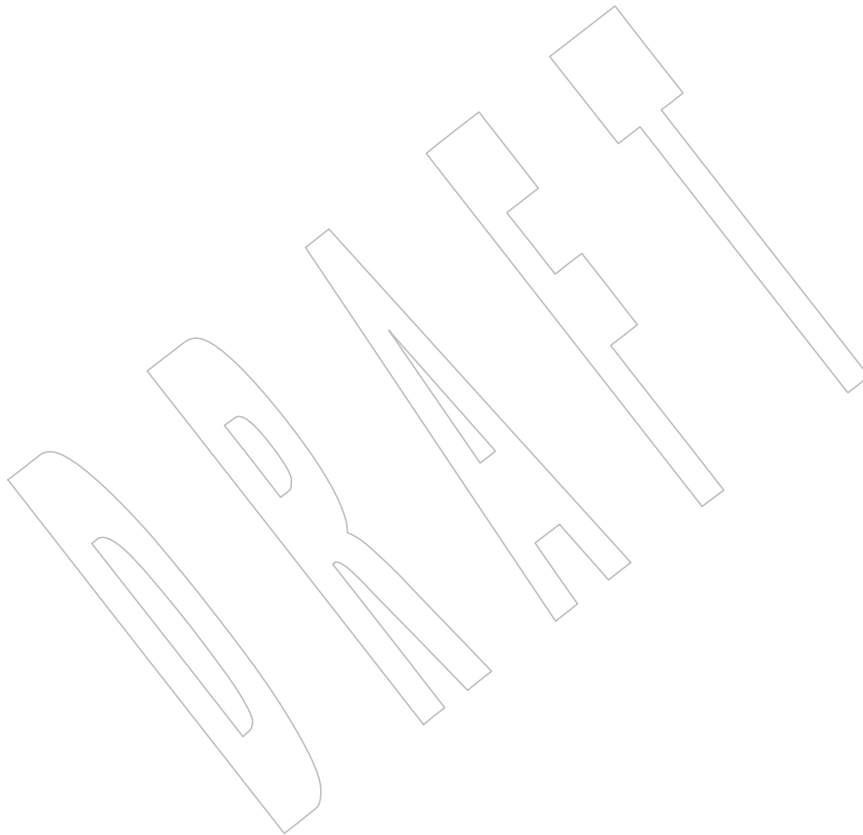
61560 **SYNOPSIS**

61561 #include <stdio.h>

61562 int sscanf(const char *restrict *s*, const char *restrict *format*, ...);

61563 **DESCRIPTION**

61564 Refer to *fscanf()*.

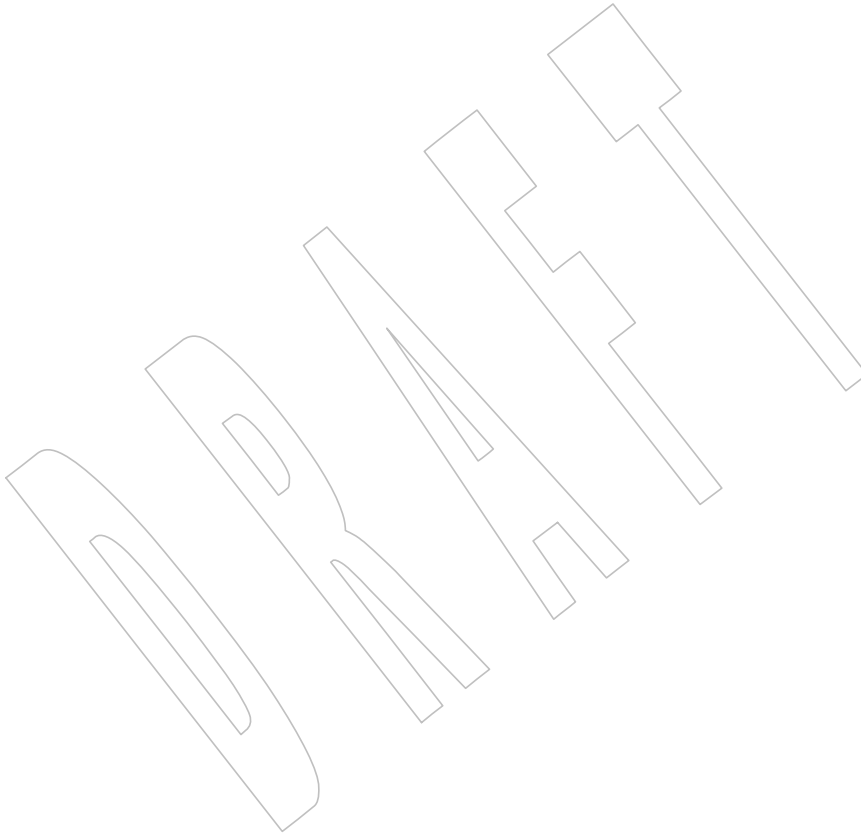


61565 **NAME**
61566 `stat` — get file status

61567 **SYNOPSIS**
61568 `#include <sys/stat.h>`

61569 `int stat(const char *restrict path, struct stat *restrict buf);`

61570 **DESCRIPTION**
61571 Refer to *[fstatat\(\)](#)*.



61572 **NAME**
61573 statvfs — get file system information

61574 **SYNOPSIS**
61575 #include <sys/statvfs.h>
61576 int statvfs(const char *restrict path, struct statvfs *restrict buf);

61577 **DESCRIPTION**
61578 Refer to *fstatvfs()*.



61579 **NAME**61580 `stderr, stdin, stdout` — standard I/O streams61581 **SYNOPSIS**61582 `#include <stdio.h>`61583 `extern FILE *stderr, *stdin, *stdout;`61584 **DESCRIPTION**61585 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61586 conflict between the requirements described here and the ISO C standard is unintentional. This
61587 volume of POSIX.1-200x defers to the ISO C standard.61588 A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type
61589 **FILE**. The *fopen()* function shall create certain descriptive data for a stream and return a pointer
61590 to designate the stream in all further transactions. Normally, there are three open streams with
61591 constant pointers declared in the **<stdio.h>** header and associated with the standard open files.61592 At program start-up, three streams shall be predefined and need not be opened explicitly:
61593 *standard input* (for reading conventional input), *standard output* (for writing conventional output),
61594 and *standard error* (for writing diagnostic output). When opened, the standard error stream is not
61595 fully buffered; the standard input and standard output streams are fully buffered if and only if
61596 the stream can be determined not to refer to an interactive device.61597 CX The following symbolic values in **<unistd.h>** define the file descriptors that shall be associated
61598 with the C-language *stdin*, *stdout*, and *stderr* when the application is started:61599 `STDIN_FILENO` Standard input value, *stdin*. Its value is 0.61600 `STDOUT_FILENO` Standard output value, *stdout*. Its value is 1.61601 `STDERR_FILENO` Standard error value, *stderr*. Its value is 2.61602 The *stderr* stream is expected to be open for reading and writing.61603 **RETURN VALUE**

61604 None.

61605 **ERRORS**

61606 No errors are defined.

61607 **EXAMPLES**

61608 None.

61609 **APPLICATION USAGE**

61610 None.

61611 **RATIONALE**

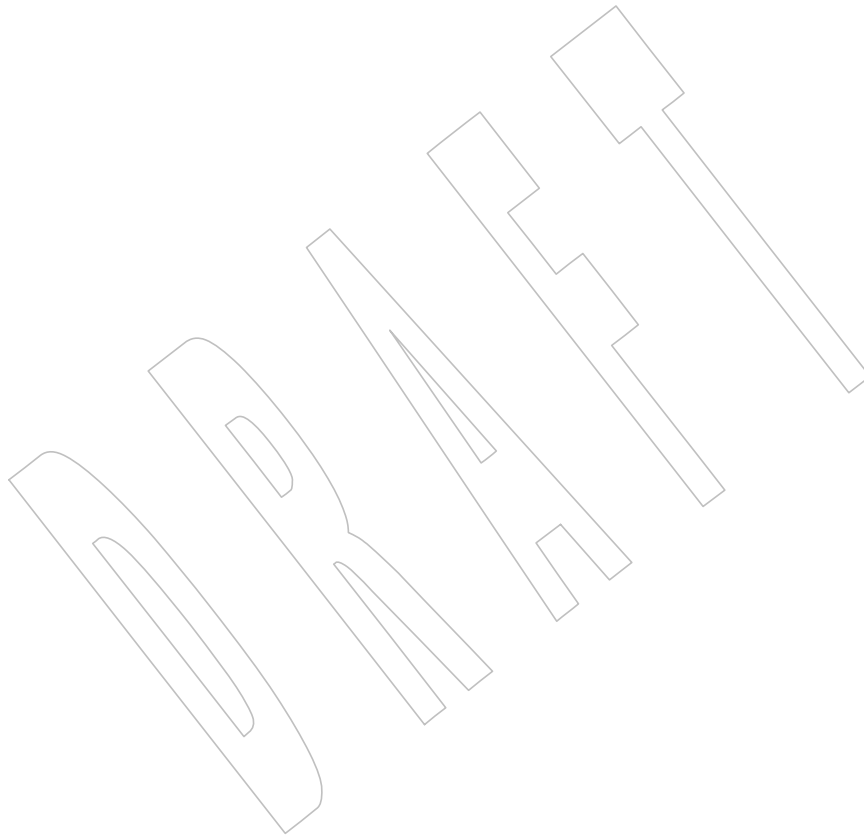
61612 None.

61613 **FUTURE DIRECTIONS**

61614 None.

61615 **SEE ALSO**61616 *fclose()*, *feof()*, *ferror()*, *fileno()*, *fopen()*, *fprintf()*, *fread()*, *fscanf()*, *fseek()*, *getc()*, *gets()*, *popen()*, -
61617 *putc()*, *puts()*, *read*, *setbuf()*, *setvbuf()*, *tmpfile()*, *ungetc()*, *vfprintf()* -61618 XBD **<stdio.h>**, **<unistd.h>** |

61619	CHANGE HISTORY
61620	First released in Issue 1.
61621	Issue 6
61622	Extensions beyond the ISO C standard are marked.
61623	A note that <i>stderr</i> is expected to be open for reading and writing is added to the DESCRIPTION.



stpcpy()

61624 **NAME**
61625 stpcpy — copy a string and return a pointer to the end of the result

SYNOPSIS

61626 CX `#include <string.h>`
61627 `char *stpcpy(char *restrict s1, const char *restrict s2);`

DESCRIPTION

61629 Refer to *strcpy()*.
61630

61631 **NAME**
61632 stpncpy — copy fixed length string, returning a pointer to the array end

61633 **SYNOPSIS**

61634 CX `#include <string.h>`
61635 `char *stpncpy(char *restrict s1, const char *restrict s2, size_t size);`

61636 **DESCRIPTION**

61637 Refer to *strncpy()*.

61638 **NAME**
 61639 `strcasemp, strcasemp_l, strncasemp, strncasemp_l` — case-insensitive string comparisons

61640 **SYNOPSIS**
 61641 `#include <strings.h>`
 61642 `int strcasemp(const char *s1, const char *s2);`
 61643 `int strcasemp_l(const char *s1, const char *s2,`
 61644 `locale_t locale);`
 61645 `int strncasemp(const char *s1, const char *s2, size_t n);`
 61646 `int strncasemp_l(const char *s1, const char *s2,`
 61647 `size_t n, locale_t locale);`

61648 **DESCRIPTION**
 61649 The `strcasemp()` and `strcasemp_l()` functions shall compare, while ignoring differences in case,
 61650 the string pointed to by `s1` to the string pointed to by `s2`. The `strncasemp()` and `strncasemp_l()`
 61651 functions shall compare, while ignoring differences in case, not more than `n` bytes from the
 61652 string pointed to by `s1` to the string pointed to by `s2`.

61653 The `strcasemp()` and `strncasemp()` functions use the current locale of the process to determine
 61654 the case of the characters.

61655 The `strcasemp_l()` and `strncasemp_l()` functions use the locale represented by `locale` to determine
 61656 the case of the characters.

61657 When the `LC_CTYPE` category of the current locale is from the POSIX locale, `strcasemp()` and
 61658 `strncasemp()` shall behave as if the strings had been converted to lowercase and then a byte
 61659 comparison performed. Otherwise, the results are unspecified.

61660 **RETURN VALUE**
 61661 Upon completion, `strcasemp()` and `strcasemp_l()` shall return an integer greater than, equal to,
 61662 or less than 0, if the string pointed to by `s1` is, ignoring case, greater than, equal to, or less than
 61663 the string pointed to by `s2`, respectively.

61664 Upon successful completion, `strncasemp()` and `strncasemp_l()` shall return an integer greater
 61665 than, equal to, or less than 0, if the possibly null-terminated array pointed to by `s1` is, ignoring
 61666 case, greater than, equal to, or less than the possibly null-terminated array pointed to by `s2`,
 61667 respectively.

61668 **ERRORS**
 61669 The `strcasemp_l()` and `strncasemp_l()` functions may fail if:
 61670 [EINVAL] `locale` is not a valid locale object handle.

61671 **EXAMPLES**
 61672 None.

61673 **APPLICATION USAGE**
 61674 None.

61675 **RATIONALE**
 61676 None.

61677 **FUTURE DIRECTIONS**
 61678 None.

61679

SEE ALSO

61680

[wcscasecmp\(\)](#)

61681

XBD [<strings.h>](#)

61682

CHANGE HISTORY

61683

First released in Issue 4, Version 2.

61684

Issue 5

61685

Moved from X/OPEN UNIX extension to BASE.

61686

Issue 7

61687

The *strcasecmp()* and *strncasecmp()* functions are moved from the XSI option to the Base.

61688

The *strcasecmp_l()* and *strncasecmp_l()* functions are added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

61689



61690 **NAME**61691 `strcat` — concatenate two strings61692 **SYNOPSIS**61693 `#include <string.h>`61694 `char *strcat(char *restrict s1, const char *restrict s2);`61695 **DESCRIPTION**61696 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61697 conflict between the requirements described here and the ISO C standard is unintentional. This
61698 volume of POSIX.1-200x defers to the ISO C standard.61699 The `strcat()` function shall append a copy of the string pointed to by `s2` (including the
61700 terminating NUL character) to the end of the string pointed to by `s1`. The initial byte of `s2`
61701 overwrites the NUL character at the end of `s1`. If copying takes place between objects that
61702 overlap, the behavior is undefined.61703 **RETURN VALUE**61704 The `strcat()` function shall return `s1`; no return value is reserved to indicate an error.61705 **ERRORS**

61706 No errors are defined.

61707 **EXAMPLES**

61708 None.

61709 **APPLICATION USAGE**61710 This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 |
61711 applications. Reliable error detection by this function was never guaranteed.61712 **RATIONALE**

61713 None.

61714 **FUTURE DIRECTIONS**

61715 None.

61716 **SEE ALSO**61717 [strncat\(\)](#)61718 XBD [<string.h>](#) |61719 **CHANGE HISTORY**

61720 First released in Issue 1. Derived from Issue 1 of the SVID.

61721 **Issue 6**61722 The `strcat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

61723 **NAME**

61724 `strchr` — string scanning operation

61725 **SYNOPSIS**

61726 `#include <string.h>`

61727 `char *strchr(const char *s, int c);`

61728 **DESCRIPTION**

61729 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61730 conflict between the requirements described here and the ISO C standard is unintentional. This
61731 volume of POSIX.1-200x defers to the ISO C standard.

61732 The `strchr()` function shall locate the first occurrence of `c` (converted to a **char**) in the string
61733 pointed to by `s`. The terminating NUL character is considered to be part of the string.

61734 **RETURN VALUE**

61735 Upon completion, `strchr()` shall return a pointer to the byte, or a null pointer if the byte was not
61736 found.

61737 **ERRORS**

61738 No errors are defined.

61739 **EXAMPLES**

61740 None.

61741 **APPLICATION USAGE**

61742 None.

61743 **RATIONALE**

61744 None.

61745 **FUTURE DIRECTIONS**

61746 None.

61747 **SEE ALSO**

61748 [strrchr\(\)](#)

61749 XBD [<string.h>](#)

61750 **CHANGE HISTORY**

61751 First released in Issue 1. Derived from Issue 1 of the SVID.

61752 **Issue 6**

61753 Extensions beyond the ISO C standard are marked.

61754 **NAME**61755 `strcmp` — compare two strings61756 **SYNOPSIS**61757 `#include <string.h>`61758 `int strcmp(const char *s1, const char *s2);`61759 **DESCRIPTION**61760 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61761 conflict between the requirements described here and the ISO C standard is unintentional. This
61762 volume of POSIX.1-200x defers to the ISO C standard.61763 The `strcmp()` function shall compare the string pointed to by `s1` to the string pointed to by `s2`.61764 The sign of a non-zero return value shall be determined by the sign of the difference between the
61765 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
61766 being compared.61767 **RETURN VALUE**61768 Upon completion, `strcmp()` shall return an integer greater than, equal to, or less than 0, if the
61769 string pointed to by `s1` is greater than, equal to, or less than the string pointed to by `s2`,
61770 respectively.61771 **ERRORS**

61772 No errors are defined.

61773 **EXAMPLES**61774 **Checking a Password Entry**61775 The following example compares the information read from standard input to the value of the
61776 name of the user entry. If the `strcmp()` function returns 0 (indicating a match), a further check
61777 will be made to see if the user entered the proper old password. The `crypt()` function shall
61778 encrypt the old password entered by the user, using the value of the encrypted password in the
61779 **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the
61780 structure, the entered password `oldpasswd` is the correct user's password. Finally, the program
61781 encrypts the new password so that it can store the information in the **passwd** structure.61782 `#include <string.h>`
61783 `#include <unistd.h>`
61784 `#include <stdio.h>`
61785 `...`
61786 `int valid_change;`
61787 `struct passwd *p;`
61788 `char user[100];`
61789 `char oldpasswd[100];`
61790 `char newpasswd[100];`
61791 `char savepasswd[100];`
61792 `...`
61793 `if (strcmp(p->pw_name, user) == 0) {`
61794 `if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {`
61795 `strcpy(savepasswd, crypt(newpasswd, user));`
61796 `p->pw_passwd = savepasswd;`
61797 `valid_change = 1;`
61798 `}`
61799 `else {`

```
61800         fprintf(stderr, "Old password is not valid\n");
61801     }
61802 }
61803 ...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[strncmp\(\)](#)

XBD [<string.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

DRAFT

61817 **NAME**

61818 strcoll, strcoll_l — string comparison using collating information

61819 **SYNOPSIS**

61820 #include <string.h>

61821 int strcoll(const char *s1, const char *s2);

61822 CX int strcoll_l(const char *s1, const char *s2,
61823 locale_t locale);61824 **DESCRIPTION**61825 CX For *strcoll()*: The functionality described on this reference page is aligned with the ISO C
61826 standard. Any conflict between the requirements described here and the ISO C standard is
61827 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.61828 CX The *strcoll()* and *strcoll_l()* functions shall compare the string pointed to by *s1* to the string
61829 pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the current
61830 locale, or of the locale represented by *locale*, respectively.61831 CX The *strcoll()* and *strcoll_l()* functions shall not change the setting of *errno* if successful.61832 Since no return value is reserved to indicate an error, an application wishing to check for error
61833 CX situations should set *errno* to 0, then call *strcoll()*, or *strcoll_l()* then check *errno*.61834 **RETURN VALUE**61835 Upon successful completion, *strcoll()* shall return an integer greater than, equal to, or less than 0,
61836 according to whether the string pointed to by *s1* is greater than, equal to, or less than the string
61837 CX pointed to by *s2* when both are interpreted as appropriate to the current locale. On error,
61838 *strcoll()* may set *errno*, but no return value is reserved to indicate an error.61839 Upon successful completion, *strcoll_l()* shall return an integer greater than, equal to, or less than
61840 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the
61841 string pointed to by *s2* when both are interpreted as appropriate to the locale represented by
61842 *locale*. On error, *strcoll_l()* may set *errno*, but no return value is reserved to indicate an error.61843 **ERRORS**

61844 These functions may fail if:

61845 CX [EINVAL] The *s1* or *s2* arguments contain characters outside the domain of the collating
61846 sequence.61847 The *strcoll_l()* function may fail if:61848 CX [EINVAL] *locale* is not a valid locale object handle.61849 **EXAMPLES**61850 **Comparing Nodes**61851 The following example uses an application-defined function, *node_compare()*, to compare two
61852 nodes based on an alphabetical ordering of the *string* field.

61853 #include <string.h>

61854 ...

61855 struct node { /* These are stored in the table. */

61856 char *string;

61857 int length;

61858 };


```

61859     ...
61860     int node_compare(const void *node1, const void *node2)
61861     {
61862         return strcoll(((const struct node *)node1)->string,
61863                       ((const struct node *)node2)->string);
61864     }
61865     ...

```

APPLICATION USAGE

The *strxfrm()* and *strcmp()* functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alphasort(), *strcmp()*, *strxfrm()*

XBD [<string.h>](#)

CHANGE HISTORY

First released in Issue 3.

Issue 5

The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EINVAL] optional error condition is added.

An example is added.

Issue 7

The *strcoll_1()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

61888 **NAME**

61889 stpcpy, strcpy — copy a string and return a pointer to the end of the result

61890 **SYNOPSIS**

61891 #include <string.h>

61892 CX char *stpcpy(char *restrict s1, const char *restrict s2);

61893 char *strcpy(char *restrict s1, const char *restrict s2);

61894 **DESCRIPTION**61895 CX For *stpcpy()*: The functionality described on this reference page is aligned with the ISO C
61896 standard. Any conflict between the requirements described here and the ISO C standard is
61897 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.61898 CX The *stpcpy()* and *strcpy()* functions shall copy the string pointed to by *s2* (including the
61899 terminating NUL character) into the array pointed to by *s1*.

61900 If copying takes place between objects that overlap, the behavior is undefined.

61901 **RETURN VALUE**61902 CX The *stpcpy()* function shall return a pointer to the terminating NUL character copied into the *s1*
61903 buffer.61904 The *strcpy()* function shall return *s1*.

61905 No return values are reserved to indicate an error.

61906 **ERRORS**

61907 No errors are defined.

61908 **EXAMPLES**61909 **Construction of a Multi-Part Message in a Single Buffer**

61910 #include <string.h>

61911 #include <stdio.h>

61912 int

61913 main (void)

61914 {

61915 char buffer [10];

61916 char *name = buffer;

61917 name = stpcpy (stpcpy (stpcpy (name, "ice"), "-"), "cream");

61918 puts (buffer);

61919 return 0;

61920 }

61921 **Initializing a String**61922 The following example copies the string "-----" into the *permstring* variable.

61923 #include <string.h>

61924 ...

61925 static char permstring[11];

61926 ...

61927 strcpy(permstring, "-----");

61928 ...

61929

Storing a Key and Data

61930

The following example allocates space for a key using *malloc()* then uses *strcpy()* to place the key there. Then it allocates space for data using *malloc()*, and uses *strcpy()* to place data there. (The user-defined function *dbfree()* frees memory previously allocated to an array of type **struct element** *.)

61931

61932

61933

61934

```
#include <string.h>
```

61935

```
#include <stdlib.h>
```

61936

```
#include <stdio.h>
```

61937

```
...
```

61938

```
/* Structure used to read data and store it. */
```

61939

```
struct element {
```

61940

```
    char *key;
```

61941

```
    char *data;
```

61942

```
};
```

61943

```
struct element *tbl, *curtbl;
```

61944

```
char *key, *data;
```

61945

```
int count;
```

61946

```
...
```

61947

```
void dbfree(struct element *, int);
```

61948

```
...
```

61949

```
if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
```

61950

```
    perror("malloc"); dbfree(tbl, count); return NULL;
```

61951

```
}
```

61952

```
strcpy(curtbl->key, key);
```

61953

```
if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
```

61954

```
    perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
```

61955

```
}
```

61956

```
strcpy(curtbl->data, data);
```

61957

```
...
```

61958

APPLICATION USAGE

61959

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

61960

61961

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

61962

61963

RATIONALE

61964

None.

61965

FUTURE DIRECTIONS

61966

None.

61967

SEE ALSO

61968

[*strncpy\(\)*](#), [*wscpy\(\)*](#)

61969

XBD [**<string.h>**](#)

61970

CHANGE HISTORY

61971

First released in Issue 1. Derived from Issue 1 of the SVID.

61972

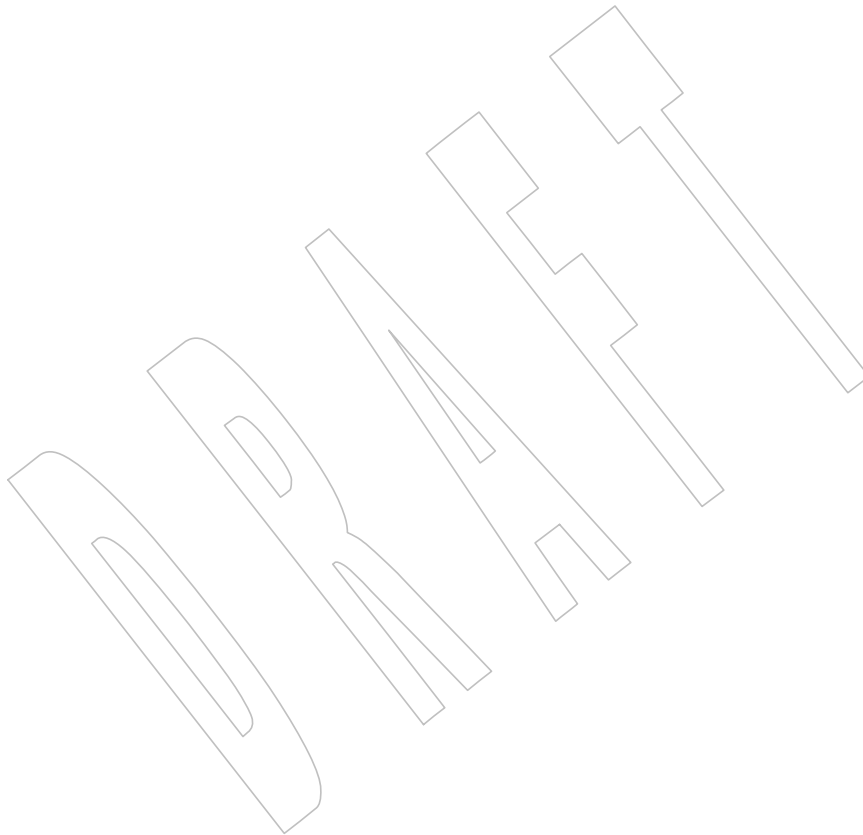
Issue 6

61973

The *strcpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

strcpy()61974
61975
61976**Issue 7**

The *strcpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.



61977 **NAME**

61978 strcspn — get the length of a complementary substring

61979 **SYNOPSIS**

61980 #include <string.h>

61981 size_t strcspn(const char *s1, const char *s2);

61982 **DESCRIPTION**61983 CX The functionality described on this reference page is aligned with the ISO C standard. Any
61984 conflict between the requirements described here and the ISO C standard is unintentional. This
61985 volume of POSIX.1-200x defers to the ISO C standard.61986 The *strcspn()* function shall compute the length (in bytes) of the maximum initial segment of the
61987 string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.61988 **RETURN VALUE**61989 The *strcspn()* function shall return the length of the computed segment of the string pointed to
61990 by *s1*; no return value is reserved to indicate an error.61991 **ERRORS**

61992 No errors are defined.

61993 **EXAMPLES**

61994 None.

61995 **APPLICATION USAGE**

61996 None.

61997 **RATIONALE**

61998 None.

61999 **FUTURE DIRECTIONS**

62000 None.

62001 **SEE ALSO**62002 [strspn\(\)](#)62003 XBD [<string.h>](#)62004 **CHANGE HISTORY**

62005 First released in Issue 1. Derived from Issue 1 of the SVID.

62006 **Issue 5**62007 The RETURN VALUE section is updated to indicate that *strcspn()* returns the length of *s1*, and
62008 not *s1* itself as was previously stated.62009 **Issue 6**62010 The Open Group Corrigendum U030/1 is applied. The text of the RETURN VALUE section is
62011 updated to indicate that the computed segment length is returned, not the *s1* length.

62012 **NAME**62013 `strdup`, `strndup` — duplicate a specific number of bytes from a string62014 **SYNOPSIS**

```
62015 CX      #include <string.h>
62016          char *strdup(const char *s);
62017          char *strndup(const char *s, size_t size);
```

62018 **DESCRIPTION**

62019 The `strdup()` function shall return a pointer to a new string, which is a duplicate of the string
 62020 pointed to by `s`. The returned pointer can be passed to `free()`. A null pointer is returned if the
 62021 new string cannot be created.

62022 The `strndup()` function shall be equivalent to the `strdup()` function, duplicating the provided `s` in
 62023 a new block of memory allocated as if by using `malloc()`, with the exception being that `strndup()`
 62024 copies at most `size` plus one bytes into the newly allocated memory, terminating the new string
 62025 with a NUL character. If the length of `s` is larger than `size`, only `size` bytes shall be duplicated. If
 62026 `size` is larger than the length of `s`, all bytes in `s` shall be copied into the new memory buffer,
 62027 including the terminating NUL character. The newly created string shall always be properly
 62028 terminated.

62029 **RETURN VALUE**

62030 The `strdup()` function shall return a pointer to a new string on success. Otherwise, it shall return
 62031 a null pointer and set `errno` to indicate the error.

62032 Upon successful completion, the `strndup()` function shall return a pointer to the newly allocated
 62033 memory containing the duplicated string. Otherwise, it shall return a null pointer and set `errno`
 62034 to indicate the error.

62035 **ERRORS**

62036 These functions shall fail if:

62037 [ENOMEM] Storage space available is insufficient.

62038 **EXAMPLES**

62039 None.

62040 **APPLICATION USAGE**

62041 None.

62042 **RATIONALE**

62043 None.

62044 **FUTURE DIRECTIONS**

62045 None.

62046 **SEE ALSO**62047 *free()*, *malloc()*, *wcsdup()*62048 XBD `<string.h>`62049 **CHANGE HISTORY**

62050 First released in Issue 4, Version 2.

62051

Issue 5

62052

Moved from X/OPEN UNIX extension to BASE.

62053

Issue 7

62054

Austin Group Interpretation 1003.1-2001 #044 is applied, changing the “may fail” [ENOMEM] error to become a “shall fail” error.

62055

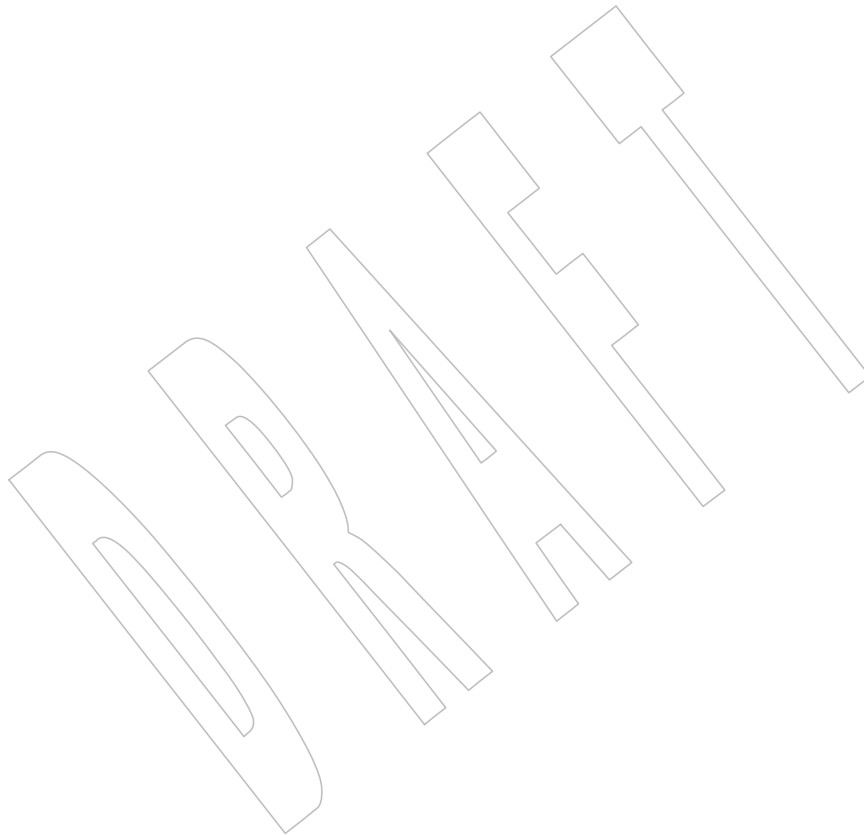
62056

The *strdup()* function is moved from the XSI option to the Base.

62057

The *strndup()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

62058



62059 **NAME**
 62060 `strerror, strerror_l, strerror_r` — get error message string

62061 **SYNOPSIS**
 62062 `#include <string.h>`
 62063 `char *strerror(int errnum);`
 62064 CX `char *strerror_l(int errnum, locale_t locale);`
 62065 `int strerror_r(int errnum, char *strerrbuf, size_t buflen);`

62066 DESCRIPTION

62067 CX For `strerror()`: The functionality described on this reference page is aligned with the ISO C
 62068 standard. Any conflict between the requirements described here and the ISO C standard is
 62069 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

62070 The `strerror()` function shall map the error number in `errnum` to a locale-dependent error
 62071 message string and shall return a pointer to it. Typically, the values for `errnum` come from `errno`,
 62072 but `strerror()` shall map any value of type `int` to a message.

62073 The string pointed to shall not be modified by the application. The string may be overwritten by
 62074 a subsequent call to `strerror()`.

62075 CX The string may be overwritten by a subsequent call to `strerror_l()` in the same thread.

62076 The contents of the error message strings returned by `strerror()` should be determined by the
 62077 setting of the `LC_MESSAGES` category in the current locale.

62078 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 62079 `strerror()`.

62080 CX The `strerror()` and `strerror_l()` functions shall not change the setting of `errno` if successful.

62081 Since no return value is reserved to indicate an error, an application wishing to check for error
 62082 situations should set `errno` to 0, then call `strerror()`, then check `errno`.

62083 The `strerror()` function need not be thread-safe. A function that is not required to be thread-safe
 62084 is not required to be reentrant.

62085 The `strerror_l()` function shall map the error number in `errnum` to a locale-dependent error
 62086 message string in the locale represented by `locale` and shall return a pointer to it.

62087 The `strerror_r()` function shall map the error number in `errnum` to a locale-dependent error
 62088 message string and shall return the string in the buffer pointed to by `strerrbuf`, with length
 62089 `buflen`.

62090 RETURN VALUE

62091 Upon completion, whether successful or not, `strerror()` shall return a pointer to the generated
 62092 CX message string. On error `errno` may be set, but no return value is reserved to indicate an error.

62093 Upon successful completion, `strerror_l()` shall return a pointer to the generated message string. If
 62094 `errnum` is not a valid error number, `errno` may be set to [EINVAL], but a pointer to a message
 62095 string shall still be returned. If any other error occurs, `errno` shall be set to indicate the error and
 62096 a null pointer shall be returned.

62097 Upon successful completion, `strerror_r()` shall return 0. Otherwise, an error number shall be
 62098 returned to indicate the error.

62099 **ERRORS**

62100 These functions may fail if:

62101 CX [EINVAL] The value of *errnum* is not a valid error number.62102 The *strerror_l()* function may fail if:62103 CX [EINVAL] The *locale* argument is not a valid locale object handle.62104 The *strerror_r()* function may fail if:62105 CX [ERANGE] Insufficient storage was supplied via *strrdbuf* and *buflen* to contain the
62106 generated message string.62107 **EXAMPLES**

62108 None.

62109 **APPLICATION USAGE**62110 Historically in some implementations calls to *perror()* would overwrite the string that the
62111 pointer returned by *strerror()* points to. Such implementations did not conform to the ISO C
62112 standard; however, application developers should be aware of this behavior if they wish their
62113 applications to be portable to such implementations.62114 **RATIONALE**62115 The *strerror_l()* function is required to be thread-safe, thereby eliminating the need for an
62116 equivalent to the *strerror_r()* function.62117 **FUTURE DIRECTIONS**

62118 None.

62119 **SEE ALSO**62120 *perror()*

62121 XBD <string.h>

62122 **CHANGE HISTORY**

62123 First released in Issue 3.

62124 **Issue 5**62125 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

62126 A note indicating that this function need not be reentrant is added to the DESCRIPTION.

62127 **Issue 6**

62128 Extensions beyond the ISO C standard are marked.

62129 The following new requirements on POSIX implementations derive from alignment with the
62130 Single UNIX Specification:62131 • In the RETURN VALUE section, the fact that *errno* may be set is added.

62132 • The [EINVAL] optional error condition is added.

62133 The normative text is updated to avoid use of the term “must” for application requirements.

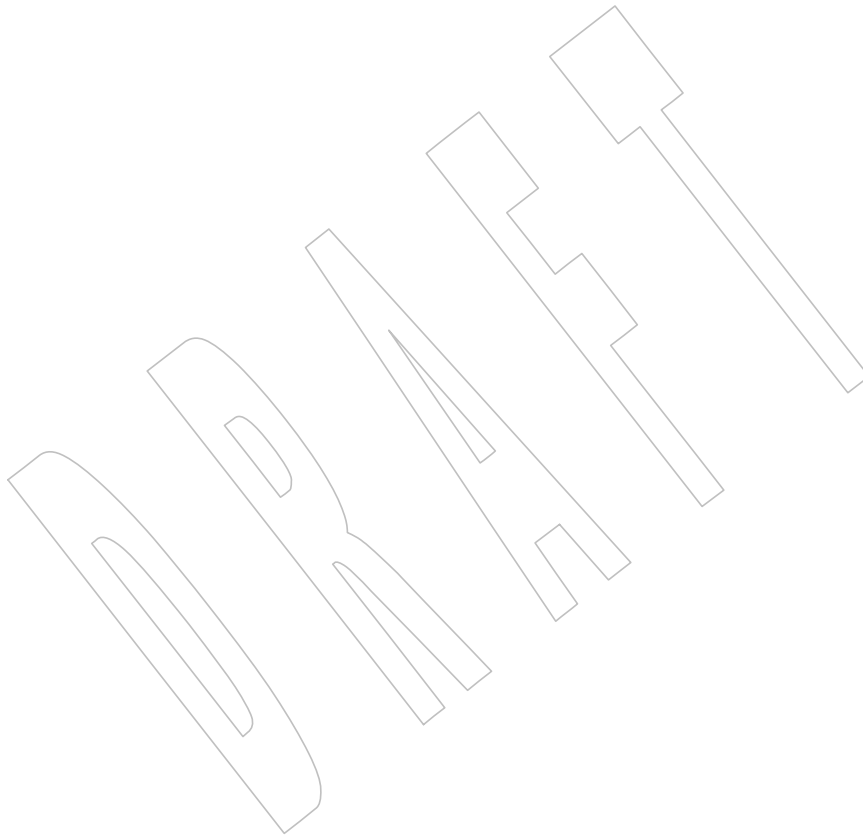
62134 The *strerror_r()* function is added in response to IEEE PASC Interpretation 1003.1c #39.62135 The *strerror_r()* function is marked as part of the Thread-Safe Functions option.62136 **Issue 7**

62137 Austin Group Interpretation 1003.1-2001 #072 is applied, updating the ERRORS section.

62138 SD5-XSH-ERN-191 is applied, updating the APPLICATION USAGE section. +

62139 The *strerror_l()* function is added from The Open Group Technical Standard, 2006, Extended API
62140 Set Part 4.

The *strerror_r()* function is moved from the Thread-Safe Functions option to the Base.



62142 **NAME**
 62143 strfmon, strfmon_l — convert monetary value to a string

62144 **SYNOPSIS**
 62145 #include <monetary.h>
 62146 ssize_t strfmon(char *restrict s, size_t maxsize,
 62147 const char *restrict format, ...);
 62148 ssize_t strfmon_l(char *restrict s, size_t maxsize,
 62149 locale_t locale, const char *restrict format, ...);

62150 **DESCRIPTION**
 62151 The *strfmon()* function shall place characters into the array pointed to by *s* as controlled by the
 62152 string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

62153 The format is a character string, beginning and ending in its initial state, if any, that contains two
 62154 types of objects: *plain characters*, which are simply copied to the output stream, and *conversion*
 62155 *specifications*, each of which shall result in the fetching of zero or more arguments which are
 62156 converted and formatted. The results are undefined if there are insufficient arguments for the
 62157 format. If the format is exhausted while arguments remain, the excess arguments are simply
 62158 ignored.

62159 The application shall ensure that a conversion specification consists of the following sequence:

- 62160 • A '%' character
- 62161 • Optional flags
- 62162 • Optional field width
- 62163 • Optional left precision
- 62164 • Optional right precision
- 62165 • A required conversion specifier character that determines the conversion to be performed

62166 The *strfmon_l()* function shall be equivalent to the *strfmon()* function, except that the locale data
 62167 used is from the locale represented by *locale*.

62168 **Flags**

62169 One or more of the following optional flags can be specified to control the conversion:

- 62170 =*f* An '=' followed by a single character *f* which is used as the numeric fill character. In
 62171 order to work with precision or width counts, the fill character shall be a single byte
 62172 character; if not, the behavior is undefined. The default numeric fill character is the
 62173 <space>. This flag does not affect field width filling which always uses the <space>.
 62174 This flag is ignored unless a left precision (see below) is specified.
- 62175 ^ Do not format the currency amount with grouping characters. The default is to insert
 62176 the grouping characters if defined for the current locale.
- 62177 + or (Specify the style of representing positive and negative currency amounts. Only one of
 62178 '+' or '(' may be specified. If '+' is specified, the locale's equivalent of '+' and '-'
 62179 are used (for example, in the U.S., the empty string if positive and '-' if negative). If
 62180 '(' is specified, negative amounts are enclosed within parentheses. If neither flag is
 62181 specified, the '+' style is used.

- 62182 ! Suppress the currency symbol from the output conversion.
- 62183 – Specify the alignment. If this flag is present the result of the conversion is left-justified
62184 (padded to the right) rather than right-justified. This flag shall be ignored unless a field
62185 width (see below) is specified.

62186 Field Width

- 62187 *w* A decimal digit string *w* specifying a minimum field width in bytes in which the result
62188 of the conversion is right-justified (or left-justified if the flag ‘-’ is specified). The
62189 default is 0.

62190 Left Precision

- 62191 #*n* A ‘#’ followed by a decimal digit string *n* specifying a maximum number of digits
62192 expected to be formatted to the left of the radix character. This option can be used to
62193 keep the formatted output from multiple calls to the *strfmon()* function aligned in the
62194 same columns. It can also be used to fill unused positions with a special character as in
62195 "\$***123.45". This option causes an amount to be formatted as if it has the number
62196 of digits specified by *n*. If more than *n* digit positions are required, this conversion
62197 specification is ignored. Digit positions in excess of those actually required are filled
62198 with the numeric fill character (see the =*f* flag above).

62199 If grouping has not been suppressed with the ‘^’ flag, and it is defined for the current
62200 locale, grouping separators are inserted before the fill characters (if any) are added.
62201 Grouping separators are not applied to fill characters even if the fill character is a digit.

62202 To ensure alignment, any characters appearing before or after the number in the
62203 formatted output such as currency or sign symbols are padded as necessary with
62204 <space>s to make their positive and negative formats an equal length.

62205 Right Precision

- 62206 .*p* A period followed by a decimal digit string *p* specifying the number of digits after the
62207 radix character. If the value of the right precision *p* is 0, no radix character appears. If a
62208 right precision is not included, a default specified by the current locale is used. The
62209 amount being formatted is rounded to the specified number of digits prior to
62210 formatting.

62211 Conversion Specifier Characters

62212 The conversion specifier characters and their meanings are:

- 62213 *i* The **double** argument is formatted according to the locale’s international currency
62214 format (for example, in the U.S.: USD 1,234.56). If the argument is ±Inf or NaN, the
62215 result of the conversion is unspecified.
- 62216 *n* The **double** argument is formatted according to the locale’s national currency format
62217 (for example, in the U.S.: \$1,234.56). If the argument is ±Inf or NaN, the result of the
62218 conversion is unspecified.
- 62219 % Convert to a ‘%’; no argument is converted. The entire conversion specification shall
62220 be %%.

62221

Locale Information

62222

62223

62224

62225

62226

The *LC_MONETARY* category of the locale of the process affects the behavior of this function including the monetary radix character (which may be different from the numeric radix character affected by the *LC_NUMERIC* category), the grouping separator, the currency symbols, and formats. The international currency symbol should be conformant with the ISO 4217:2001 standard.

62227

If the value of *maxsize* is greater than `{SSIZE_MAX}`, the result is implementation-defined.

62228

RETURN VALUE

62229

62230

62231

62232

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise, `-1` shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate the error.

62233

ERRORS

62234

These functions shall fail if:

62235

[E2BIG] Conversion stopped due to lack of space in the buffer.

62236

The *strfmon_l()* function may fail if:

62237

[EINVAL] *locale* is not a valid locale object.

62238

EXAMPLES

62239

62240

Given a locale for the U.S. and the values 123.45, `-123.45`, and 3456.781, the following output might be produced. Square brackets ("`[]`") are used in this example to delimit the output.

62241

`%n` [\$123.45] Default formatting

62242

[-\$123.45]

62243

[\$3,456.78]

62244

`%11n` [\$123.45] Right align within an 11-character field

62245

[-\$123.45]

62246

[\$3,456.78]

62247

`%#5n` [\$ 123.45] Aligned columns for values up to 99999

62248

[-\$ 123.45]

62249

[\$ 3,456.78]

62250

`%=*#5n` [\$***123.45] Specify a fill character

62251

[-\$***123.45]

62252

[\$*3,456.78]

62253

`%=0#5n` [\$000123.45] Fill characters do not use grouping

62254

[-\$000123.45] even if the fill character is a digit

62255

[\$03,456.78]

62256

`%^#5n` [\$ 123.45] Disable the grouping separator

62257

[-\$ 123.45]

62258

[\$ 3456.78]

62259

`%^#5.0n` [\$ 123] Round off to whole units

62260

[-\$ 123]

62261

[\$ 3457]

62262

`%^#5.4n` [\$ 123.4500] Increase the precision

62263

[-\$ 123.4500]

62264

[\$ 3456.7810]

62265

`%(#5n` [\$ 123.45] Use an alternative pos/neg style

62266

[(\$ 123.45)]

62267 [\$ 3,456.78]

62268 %!(#5n [123.45] Disable the currency symbol

62269 [(123.45)

62270 [3,456.78]

62271 %-14#5.4n [\$ 123.4500] Left-justify the output

62272 [-\$ 123.4500]

62273 [\$ 3,456.7810]

62274 %14#5.4n [\$ 123.4500] Corresponding right-justified output

62275 [-\$ 123.4500]

62276 [\$ 3,456.7810]

62277 See also the EXAMPLES section in *fprintf()*.

62278 APPLICATION USAGE

62279 None.

62280 RATIONALE

62281 None.

62282 FUTURE DIRECTIONS

62283 Lowercase conversion characters are reserved for future standards use and uppercase for
62284 implementation-defined use.

62285 SEE ALSO

62286 *fprintf()*, *localeconv()*

62287 XBD <monetary.h>

62288 CHANGE HISTORY

62289 First released in Issue 4.

62290 Issue 5

62291 Moved from ENHANCED I18N to BASE.

62292 The [ENOSYS] error is removed.

62293 A sentence is added to the DESCRIPTION warning about values of *maxsize* that are greater than
62294 {SSIZE_MAX}.

62295 Issue 6

62296 The normative text is updated to avoid use of the term “must” for application requirements.

62297 The **restrict** keyword is added to the *strfmon()* prototype for alignment with the
62298 ISO/IEC 9899:1999 standard.

62299 The EXAMPLES section is reworked, clarifying the output format.

62300 Issue 7

62301 SD5-XSH-ERN-29 is applied, updating the examples for %(#5n and %!(#5n.

62302 The *strfmon()* function is moved from the XSI option to the Base.

62303 The *strfmon_l()* function is added from The Open Group Technical Standard, 2006, Extended API
62304 Set Part 4.

62305 **NAME**
 62306 `strptime, strptime_l` — convert date and time to a string

62307 **SYNOPSIS**

```
62308 #include <time.h>
62309
62310 size_t strptime(char *restrict s, size_t maxsize,
62311               const char *restrict format, const struct tm *restrict timeptr);
CX 62311 size_t strptime_l(char *restrict s, size_t maxsize,
62312                 const char *restrict format, const struct tm *restrict timeptr,
62313                 locale_t locale);
```

62314 **DESCRIPTION**

62315 CX For `strptime()`: The functionality described on this reference page is aligned with the ISO C
 62316 standard. Any conflict between the requirements described here and the ISO C standard is
 62317 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

62318 The `strptime()` function shall place bytes into the array pointed to by `s` as controlled by the string
 62319 pointed to by `format`. The format is a character string, beginning and ending in its initial shift
 62320 state, if any. The `format` string consists of zero or more conversion specifications and ordinary
 62321 characters. A conversion specification consists of a '%' character, possibly followed by an `E` or `O`
 62322 modifier, and a terminating conversion specifier character that determines the conversion
 62323 specification's behavior. All ordinary characters (including the terminating NUL character) are
 62324 copied unchanged into the array. If copying takes place between objects that overlap, the
 62325 behavior is undefined. No more than `maxsize` bytes are placed into the array. Each conversion
 62326 specifier is replaced by appropriate characters as described in the following list. The appropriate
 62327 characters are determined using the `LC_TIME` category of the current locale and by the values of
 62328 zero or more members of the broken-down time structure pointed to by `timeptr`, as specified in
 62329 brackets in the description. If any of the specified values are outside the normal range, the
 62330 characters stored are unspecified.

62331 CX The `strptime_l()` function shall be equivalent to the `strptime()` function, except that the locale data
 62332 used is from the locale represented by `locale`.

62333 Local timezone information is used as though `strptime()` called `tzset()`.

62334 The following conversion specifications are supported:

62335	%a	Replaced by the locale's abbreviated weekday name. [<code>tm_wday</code>]
62336	%A	Replaced by the locale's full weekday name. [<code>tm_wday</code>]
62337	%b	Replaced by the locale's abbreviated month name. [<code>tm_mon</code>]
62338	%B	Replaced by the locale's full month name. [<code>tm_mon</code>]
62339	%c	Replaced by the locale's appropriate date and time representation. (See the Base 62340 Definitions volume of POSIX.1-200x, <time.h> .)
62341	%C	Replaced by the year divided by 100 and truncated to an integer, as a decimal number 62342 [00,99]. [<code>tm_year</code>]
62343	%d	Replaced by the day of the month as a decimal number [01,31]. [<code>tm_mday</code>]
62344	%D	Equivalent to %m/%d/%y. [<code>tm_mon, tm_mday, tm_year</code>]
62345	%e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded 62346 by a space. [<code>tm_mday</code>]

strftime()*System Interfaces*

62347		%F	Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format). [<i>tm_year</i> , <i>tm_mon</i> , <i>tm_mday</i>]
62348			
62349		%g	Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
62350			
62351		%G	Replaced by the week-based year (see below) as a decimal number (for example, 1977). [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
62352			
62353		%h	Equivalent to %b. [<i>tm_mon</i>]
62354		%H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [<i>tm_hour</i>]
62355		%I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [<i>tm_hour</i>]
62356		%j	Replaced by the day of the year as a decimal number [001,366]. [<i>tm_yday</i>]
62357		%m	Replaced by the month as a decimal number [01,12]. [<i>tm_mon</i>]
62358		%M	Replaced by the minute as a decimal number [00,59]. [<i>tm_min</i>]
62359		%n	Replaced by a <newline>.
62360		%p	Replaced by the locale's equivalent of either a.m. or p.m. [<i>tm_hour</i>]
62361	CX	%r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be equivalent to %I:%M:%S %p. [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
62362			
62363		%R	Replaced by the time in 24-hour notation (%H:%M). [<i>tm_hour</i> , <i>tm_min</i>]
62364		%S	Replaced by the second as a decimal number [00,60]. [<i>tm_sec</i>]
62365		%t	Replaced by a <tab>.
62366		%T	Replaced by the time (%H:%M:%S). [<i>tm_hour</i> , <i>tm_min</i> , <i>tm_sec</i>]
62367		%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [<i>tm_wday</i>]
62368			
62369		%U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
62370			
62371			
62372		%V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
62373			
62374			
62375			
62376			
62377		%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [<i>tm_wday</i>]
62378			
62379		%W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [<i>tm_year</i> , <i>tm_wday</i> , <i>tm_yday</i>]
62380			
62381			
62382		%x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-200x, <time.h>.)
62383			
62384		%X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-200x, <time.h>.)
62385			
62386		%y	Replaced by the last two digits of the year as a decimal number [00,99]. [<i>tm_year</i>]

62387	%Y	Replaced by the year as a decimal number (for example, 1997). [<i>tm_year</i>]
62388	%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430"
62389		means 4 hours 30 minutes behind UTC (west of Greenwich). If <i>tm_isdst</i> is zero, the
62390	CX	standard time offset is used. If <i>tm_isdst</i> is greater than zero, the daylight savings time
62391		offset is used. If <i>tm_isdst</i> is negative, no characters are returned. [<i>tm_isdst</i>]
62392		
62393	%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone
62394		information exists. [<i>tm_isdst</i>]
62395	%%	Replaced by %.
62396		If a conversion specification does not correspond to any of the above, the behavior is undefined.
62397	CX	If a struct tm broken-down time structure is created by <i>localtime()</i> or <i>localtime_r()</i> , or modified
62398		by <i>mktime()</i> , and the value of <i>TZ</i> is subsequently modified, the results of the %Z and %z
62399		<i>strptime()</i> conversion specifiers are undefined, when <i>strptime()</i> is called with such a broken-down
62400		time structure.
62401		If a struct tm broken-down time structure is created or modified by <i>gmtime()</i> or <i>gmtime_r()</i> , it is
62402		unspecified whether the result of the %Z and %z conversion specifiers shall refer to UTC or the
62403		current local timezone, when <i>strptime()</i> is called with such a broken-down time structure.

62404 Modified Conversion Specifiers

62405 Some conversion specifiers can be modified by the E or O modifier characters to indicate that an
 62406 alternative format or specification should be used rather than the one normally used by the
 62407 unmodified conversion specifier. If the alternative format or specification does not exist for the
 62408 current locale (see ERA in XBD Section 7.3.5, on page 144), the behavior shall be as if the
 62409 unmodified conversion specification were used.

62410	%EC	Replaced by the locale's alternative appropriate date and time representation.
62411	%EC	Replaced by the name of the base year (period) in the locale's alternative
62412		representation.
62413	%Ex	Replaced by the locale's alternative date representation.
62414	%EX	Replaced by the locale's alternative time representation.
62415	%EY	Replaced by the offset from %EC (year only) in the locale's alternative representation.
62416	%EY	Replaced by the full alternative year representation.
62417	%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled
62418		as needed with leading zeros if there is any alternative symbol for zero; otherwise, with
62419		leading spaces.
62420	%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled
62421		as needed with leading spaces.
62422	%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
62423	%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
62424	%Om	Replaced by the month using the locale's alternative numeric symbols.
62425	%OM	Replaced by the minutes using the locale's alternative numeric symbols.
62426	%OS	Replaced by the seconds using the locale's alternative numeric symbols.
62427	%Ou	Replaced by the weekday as a number in the locale's alternative representation
62428		(Monday=1).

strftime()

62429 %OU Replaced by the week number of the year (Sunday as the first day of the week, rules
62430 corresponding to %U) using the locale's alternative numeric symbols.

62431 %OV Replaced by the week number of the year (Monday as the first day of the week, rules
62432 corresponding to %V) using the locale's alternative numeric symbols.

62433 %Ow Replaced by the number of the weekday (Sunday=0) using the locale's alternative
62434 numeric symbols.

62435 %OW Replaced by the week number of the year (Monday as the first day of the week) using
62436 the locale's alternative numeric symbols.

62437 %Oy Replaced by the year (offset from %C) using the locale's alternative numeric symbols.

62438 %g, %G, and %V give values according to the ISO 8601:2000 standard week-based year. In this
62439 system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th,
62440 which is also the week that includes the first Thursday of the year, and is also the first week that
62441 contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the
62442 preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January
62443 1999, %G is replaced by 1998 and %V is replaced by 53. If December 29th, 30th, or 31st is a
62444 Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday
62445 30th December 1997, %G is replaced by 1998 and %V is replaced by 01.

62446 If a conversion specifier is not one of the above, the behavior is undefined.

RETURN VALUE

62447 If the total number of resulting bytes including the terminating null byte is not more than
62448 *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*,
62449 not including the terminating NUL character. Otherwise, 0 shall be returned and the contents of
62450 the array are unspecified.

ERRORS

62451 The *strftime_l()* function may fail if:

62452 CX [EINVAL] *locale* is not a valid locale object handle.

EXAMPLES**Getting a Localized Date String**

62454 The following example first sets the locale to the user's default. The locale information will be
62455 used in the *nl_langinfo()* and *strftime()* functions. The *nl_langinfo()* function returns the localized
62456 date string which specifies how the date is laid out. The *strftime()* function takes this
62457 information and, using the **tm** structure for values, places the date and time information into
62458 *datestring*.

```
62462            #include <time.h>
62463            #include <locale.h>
62464            #include <langinfo.h>
62465            ...
62466            struct tm *tm;
62467            char datestring[256];
62468            ...
62469            setlocale (LC_ALL, "");
62470            ...
62471            strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
62472            ...
```

APPLICATION USAGE

The range of values for %S is [00,60] rather than [00,59] to allow for the occasional leap second.

Some of the conversion specifications are duplicates of others. They are included for compatibility with *nl_cxtime()* and *nl_ascxtime()*, which were published in Issue 2.

Applications should use %Y (4-digit years) in preference to %y (2-digit years).

In the C locale, the E and O modifiers are ignored and the replacement strings for the following specifiers are:

%a	The first three characters of %A.
%A	One of Sunday, Monday, . . . , Saturday.
%b	The first three characters of %B.
%B	One of January, February, . . . , December.
%c	Equivalent to %a %b %e %T %Y.
%p	One of AM or PM.
%r	Equivalent to %I:%M:%S %p.
%x	Equivalent to %m/%d/%y.
%X	Equivalent to %T.
%Z	Implementation-defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *strptime()*, *time*, *tzset()*, *uselocale()*, *utime()*

XBD Section 7.3.5 (on page 144), [<time.h>](#)

CHANGE HISTORY

First released in Issue 3.

Issue 5

The description of %OV is changed to be consistent with %V and defines Monday as the first day of the week.

The description of %Oy is clarified.

Issue 6

Extensions beyond the ISO C standard are marked.

The Open Group Corrigendum U033/8 is applied. The %V conversion specifier is changed from “Otherwise, it is week 53 of the previous year, and the next week is week 1” to “Otherwise, it is the last week of the previous year, and the next week is week 1”.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The %C, %D, %e, %h, %n, %r, %R, %t, and %T conversion specifiers are added.
- The modified conversion specifiers are added for consistency with the ISO POSIX-2 standard *date* utility.

62514

The following changes are made for alignment with the ISO/IEC 9899: 1999 standard:

62515

- The *strftime()* prototype is updated.

62516

- The DESCRIPTION is extensively revised.

62517

- The %z conversion specifier is added.

62518

A new example is added.

62519

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/60 is applied.

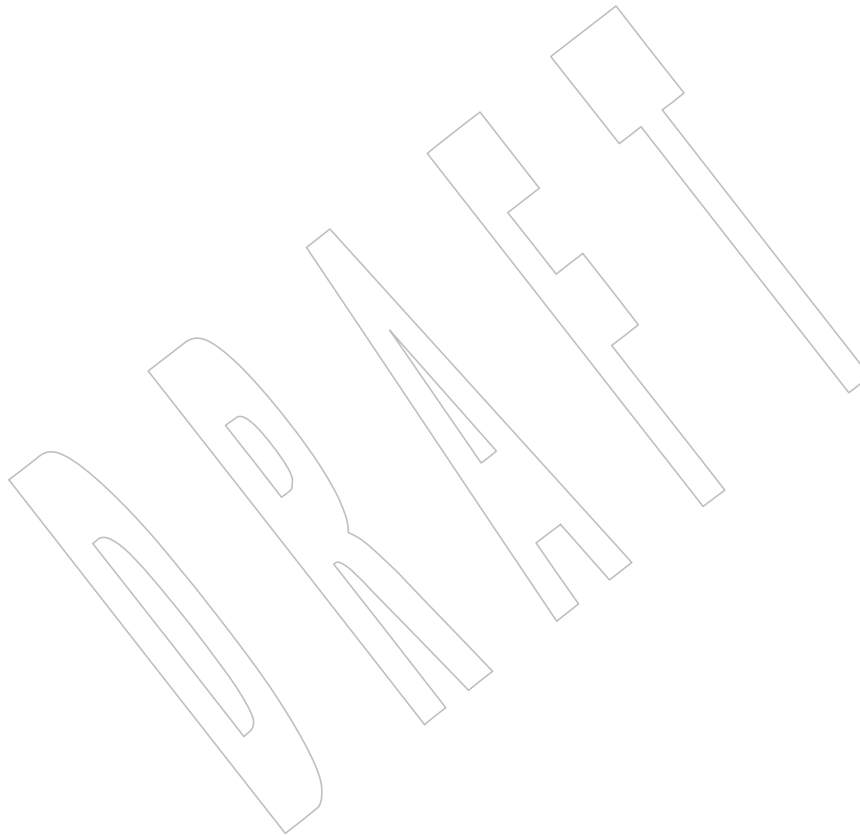
62520

Issue 7

62521

The *strftime_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

62522



62523 **NAME**
 62524 `strlen`, `strnlen` — get length of fixed size string

62525 **SYNOPSIS**

62526 `#include <string.h>`
 62527 `size_t strlen(const char *s);`
 62528 CX `size_t strnlen(const char *s, size_t maxlen);`

62529 **DESCRIPTION**

62530 CX For `strlen()`: The functionality described on this reference page is aligned with the ISO C
 62531 standard. Any conflict between the requirements described here and the ISO C standard is
 62532 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

62533 The `strlen()` function shall compute the number of bytes in the string to which `s` points, not
 62534 including the terminating NUL character.

62535 CX The `strnlen()` function shall compute the smaller of the number of bytes in the array to which `s`
 62536 points, not including the terminating NUL character, or the value of the `maxlen` argument. The
 62537 `strnlen()` function shall never examine more than `maxlen` bytes of the array pointed to by `s`.

62538 **RETURN VALUE**

62539 The `strlen()` function shall return the length of `s`; no return value shall be reserved to indicate an
 62540 error.

62541 CX The `strnlen()` function shall return an integer containing the smaller of either the length of the
 62542 string pointed to by `s` or `maxlen`.

62543 **ERRORS**

62544 No errors are defined.

62545 **EXAMPLES**

62546 **Getting String Lengths**

62547 The following example sets the maximum length of `key` and `data` by using `strlen()` to get the
 62548 lengths of those strings.

```
62549 #include <string.h>
62550 ...
62551 struct element {
62552     char *key;
62553     char *data;
62554 };
62555 ...
62556 char *key, *data;
62557 int len;

62558 *keylength = *datalength = 0;
62559 ...
62560 if ((len = strlen(key)) > *keylength)
62561     *keylength = len;
62562 if ((len = strlen(data)) > *datalength)
62563     *datalength = len;
62564 ...
```

strlen()62565 **APPLICATION USAGE**

62566 None.

62567 **RATIONALE**

62568 None.

62569 **FUTURE DIRECTIONS**

62570 None.

62571 **SEE ALSO**62572 *wcslen()*62573 XBD <**string.h**>62574 **CHANGE HISTORY**

62575 First released in Issue 1. Derived from Issue 1 of the SVID.

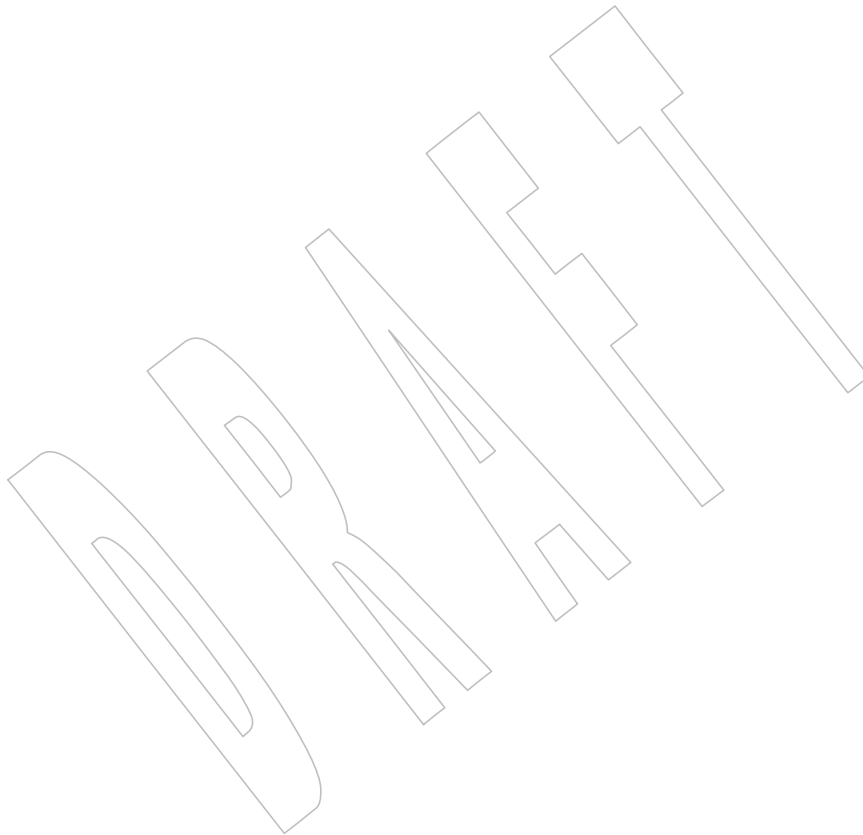
62576 **Issue 5**62577 The RETURN VALUE section is updated to indicate that *strlen()* returns the length of *s*, and not
62578 *s* itself as was previously stated.62579 **Issue 7**62580 The *strlen()* function is added from The Open Group Technical Standard, 2006, Extended API
62581 Set Part 1.

DRAFT

62582 **NAME**
62583 `strncasecmp, strncasecmp_l` — case-insensitive string comparisons

62584 **SYNOPSIS**
62585 `#include <strings.h>`
62586 `int strncasecmp(const char *s1, const char *s2, size_t n);`
62587 `int strncasecmp_l(const char *s1, const char *s2,`
62588 `size_t n, locale_t locale);`

62589 **DESCRIPTION**
62590 Refer to *strcasecmp()*.



62591 **NAME**
 62592 `strncat` — concatenate a string with part of another

62593 **SYNOPSIS**
 62594 `#include <string.h>`

62595 `char *strncat(char *restrict s1, const char *restrict s2, size_t n);`

62596 **DESCRIPTION**

62597 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62598 conflict between the requirements described here and the ISO C standard is unintentional. This
 62599 volume of POSIX.1-200x defers to the ISO C standard.

62600 The `strncat()` function shall append not more than *n* bytes (a NUL character and bytes that
 62601 follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by
 62602 *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL
 62603 character is always appended to the result. If copying takes place between objects that overlap,
 62604 the behavior is undefined.

62605 **RETURN VALUE**

62606 The `strncat()` function shall return *s1*; no return value shall be reserved to indicate an error.

62607 **ERRORS**

62608 No errors are defined.

62609 **EXAMPLES**

62610 None.

62611 **APPLICATION USAGE**

62612 None.

62613 **RATIONALE**

62614 None.

62615 **FUTURE DIRECTIONS**

62616 None.

62617 **SEE ALSO**

62618 [strcat\(\)](#)

62619 XBD [<string.h>](#)

62620 **CHANGE HISTORY**

62621 First released in Issue 1. Derived from Issue 1 of the SVID.

62622 **Issue 6**

62623 The `strncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

62624 **NAME**
 62625 `strncmp` — compare part of two strings

62626 **SYNOPSIS**
 62627 `#include <string.h>`
 62628 `int strncmp(const char *s1, const char *s2, size_t n);`

62629 **DESCRIPTION**
 62630 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62631 conflict between the requirements described here and the ISO C standard is unintentional. This
 62632 volume of POSIX.1-200x defers to the ISO C standard.

62633 The `strncmp()` function shall compare not more than *n* bytes (bytes that follow a NUL character
 62634 are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

62635 The sign of a non-zero return value is determined by the sign of the difference between the
 62636 values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings
 62637 being compared.

62638 **RETURN VALUE**
 62639 Upon successful completion, `strncmp()` shall return an integer greater than, equal to, or less than
 62640 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the
 62641 possibly null-terminated array pointed to by *s2* respectively.

62642 **ERRORS**
 62643 No errors are defined.

62644 **EXAMPLES**
 62645 None.

62646 **APPLICATION USAGE**
 62647 None.

62648 **RATIONALE**
 62649 None.

62650 **FUTURE DIRECTIONS**
 62651 None.

62652 **SEE ALSO**
 62653 [strcmp\(\)](#)
 62654 XBD [<string.h>](#)

62655 **CHANGE HISTORY**
 62656 First released in Issue 1. Derived from Issue 1 of the SVID.

62657 **Issue 6**
 62658 Extensions beyond the ISO C standard are marked.

62659 **NAME**

62660 stpncpy, strncpy — copy fixed length string, returning a pointer to the array end

62661 **SYNOPSIS**

62662 #include <string.h>

62663 CX `char *stpncpy(char *restrict s1, const char *restrict s2, size_t n);`62664 `char *strncpy(char *restrict s1, const char *restrict s2, size_t n);`62665 **DESCRIPTION**62666 CX For *strncpy()*: The functionality described on this reference page is aligned with the ISO C
62667 standard. Any conflict between the requirements described here and the ISO C standard is
62668 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.62669 CX The *stpncpy()* and *strncpy()* functions shall copy not more than *n* bytes (bytes that follow a NUL
62670 character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.62671 If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be
62672 appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

62673 If copying takes place between objects that overlap, the behavior is undefined.

62674 **RETURN VALUE**62675 CX If a NUL character is written to the destination, the *stpncpy()* function shall return the address of
62676 the first such NUL character. Otherwise, it shall return $\&s2[n]$.62677 The *strncpy()* function shall return *s1*.

62678 No return values are reserved to indicate an error.

62679 **ERRORS**

62680 No errors are defined.

62681 **EXAMPLES**

62682 None.

62683 **APPLICATION USAGE**62684 Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that
62685 the *s2* and *s1* arrays do not overlap.62686 Character movement is performed differently in different implementations. Thus, overlapping
62687 moves may yield surprises.62688 If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not
62689 null-terminated.62690 **RATIONALE**

62691 None.

62692 **FUTURE DIRECTIONS**

62693 None.

62694 **SEE ALSO**62695 *strcpy()*, *wcsncpy()*

62696 XBD <string.h>

62697

CHANGE HISTORY

62698

First released in Issue 1. Derived from Issue 1 of the SVID.

62699

Issue 6

62700

The *strncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

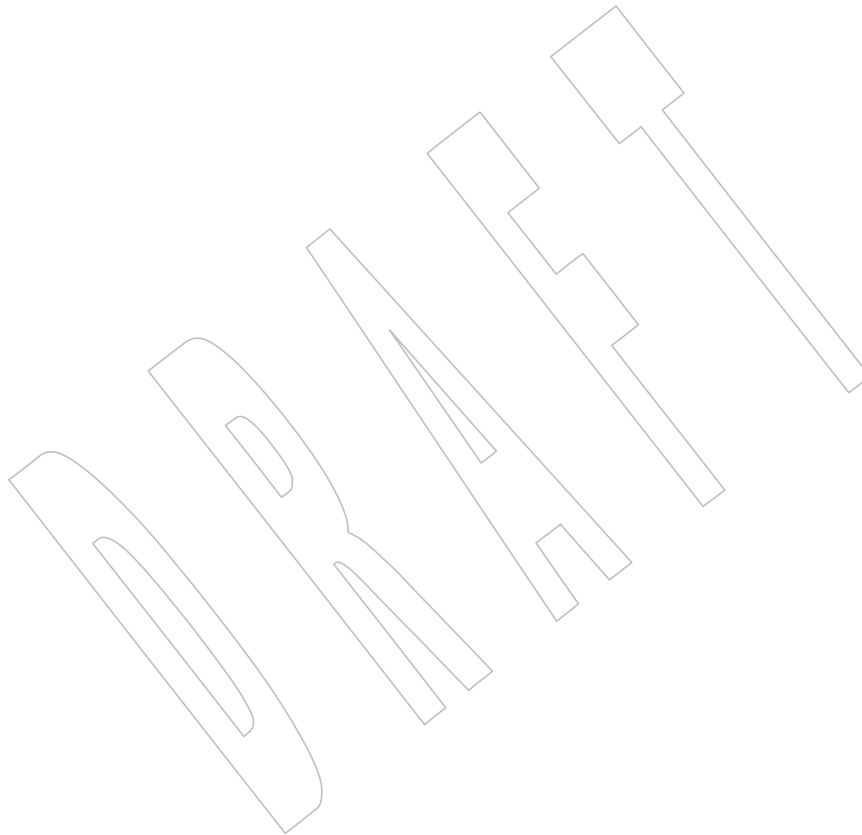
62701

Issue 7

62702

The *stpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

62703



strndup()

62704 **NAME**
62705 `strndup` — duplicate a specific number of bytes from a string

SYNOPSIS

62706 CX `#include <string.h>`
62707 `char *strndup(const char *s, size_t size);`

DESCRIPTION

62709 Refer to *strdup()*.
62710

62711 **NAME**
62712 `strnlen` — get length of fixed size string

62713 **SYNOPSIS**

62714 CX `#include <string.h>`
62715 `size_t strnlen(const char *s, size_t maxlen);`

62716 **DESCRIPTION**

62717 Refer to *strlen()*.

62718 **NAME**
 62719 `strpbrk` — scan a string for a byte

62720 **SYNOPSIS**
 62721 `#include <string.h>`
 62722 `char *strpbrk(const char *s1, const char *s2);`

62723 **DESCRIPTION**
 62724 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62725 conflict between the requirements described here and the ISO C standard is unintentional. This
 62726 volume of POSIX.1-200x defers to the ISO C standard.

62727 The `strpbrk()` function shall locate the first occurrence in the string pointed to by `s1` of any byte
 62728 from the string pointed to by `s2`.

62729 **RETURN VALUE**
 62730 Upon successful completion, `strpbrk()` shall return a pointer to the byte or a null pointer if no
 62731 byte from `s2` occurs in `s1`.

62732 **ERRORS**
 62733 No errors are defined.

62734 **EXAMPLES**
 62735 None.

62736 **APPLICATION USAGE**
 62737 None.

62738 **RATIONALE**
 62739 None.

62740 **FUTURE DIRECTIONS**
 62741 None.

62742 **SEE ALSO**
 62743 [*strchr\(\)*](#), [*strrchr\(\)*](#)

62744 XBD [*<string.h>*](#)

62745 **CHANGE HISTORY**
 62746 First released in Issue 1. Derived from Issue 1 of the SVID.

62747 **NAME**

62748 strptime — date and time conversion

62749 **SYNOPSIS**

```
62750 XSI #include <time.h>
62751 char *strptime(const char *restrict buf, const char *restrict format,
62752 struct tm *restrict tm);
```

62753 **DESCRIPTION**

62754 The *strptime()* function shall convert the character string pointed to by *buf* to values which are
62755 stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

62756 The *format* is composed of zero or more directives. Each directive is composed of one of the
62757 following: one or more white-space characters (as specified by *isspace()*); an ordinary character
62758 (neither '%' nor a white-space character); or a conversion specification. Each conversion
62759 specification is composed of a '%' character followed by a conversion character which specifies
62760 the replacement required. The conversions are determined using the *LC_TIME* category of the
62761 current locale. The application shall ensure that there is white-space or other non-alphanumeric
62762 characters between any two conversion specifications. In the following list, where numeric
62763 ranges of values are given (represented by the pattern [*x*,*y*]), the value shall fall within the
62764 range given (both bounds being inclusive), and the number of characters scanned (excluding the
62765 one matching the next directive) shall be no more than the maximum number required to
62766 represent any value in the range without leading zeros. The following conversion specifications
62767 are supported:

62768	%a	The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.
62769		
62770	%A	Equivalent to %a.
62771	%b	The month, using the locale's month names; either the abbreviated or full name may be specified.
62772		
62773	%B	Equivalent to %b.
62774	%c	Replaced by the locale's appropriate date and time representation.
62775	%C	The century number [00,99]; leading zeros shall be permitted but shall not be required.
62776	%d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
62777	%D	The date as %m/%d/%y.
62778	%e	Equivalent to %d.
62779	%h	Equivalent to %b.
62780	%H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required.
62781		
62782	%I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required.
62783		
62784	%j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required.
62785		
62786	%m	The month number [01,12]; leading zeros shall be permitted but shall not be required.

62787	%M	The minute [00,59]; leading zeros shall be permitted but shall not be required.
62788	%n	Any white space.
62789	%p	The locale's equivalent of a.m. or p.m.
62790	%r	12-hour clock time using the AM/PM notation if t_fmt_ampm is not an empty string in the <i>LC_TIME</i> portion of the current locale; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
62791		
62792		
62793	%R	The time as %H:%M.
62794	%S	The seconds [00,60]; leading zeros shall be permitted but shall not be required.
62795	%t	Any white space.
62796	%T	The time as %H:%M:%S.
62797	%U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
62798		
62799	%w	The weekday as a decimal number [0,6], with 0 representing Sunday.
62800	%W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
62801		
62802	%x	The date, using the locale's date format.
62803	%X	The time, using the locale's time format.
62804	%y	The year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required.
62805		
62806		
62807		
62808	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
62809		
62810		
62811	%Y	The year, including the century (for example, 1988); leading zeros shall be permitted but shall not be required.
62812		
62813	%%	Replaced by %.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.

62819	%Ec	The locale's alternative appropriate date and time representation.
62820	%EC	The name of the base year (period) in the locale's alternative representation.
62821	%Ex	The locale's alternative date representation.
62822	%EX	The locale's alternative time representation.
62823	%Ey	The offset from %EC (year only) in the locale's alternative representation.
62824	%EY	The full alternative year representation.
62825	%Od	The day of the month using the locale's alternative numeric symbols; leading zeros shall be permitted but shall not be required.
62826		

62827	%Oe	Equivalent to %Od.
62828	%OH	The hour (24-hour clock) using the locale's alternative numeric symbols.
62829	%OI	The hour (12-hour clock) using the locale's alternative numeric symbols.
62830	%Om	The month using the locale's alternative numeric symbols.
62831	%OM	The minutes using the locale's alternative numeric symbols.
62832	%OS	The seconds using the locale's alternative numeric symbols.
62833	%OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
62834		
62835	%Ow	The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
62836		
62837	%OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
62838		
62839	%Oy	The year (offset from %C) using the locale's alternative numeric symbols.
62840		A conversion specification composed of white-space characters is executed by scanning input up to the first character that is not white-space (which remains unscanned), or until no more characters can be scanned.
62841		
62842		
62843		A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.
62844		
62845		
62846		
62847		A series of conversion specifications composed of %n, %t, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.
62848		
62849		
62850		Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate tm structure members are set to values corresponding to the locale information. Case is ignored when matching items in <i>buf</i> such as month or weekday names. If no match is found, <i>strptime()</i> fails and no more characters are scanned.
62851		
62852		
62853		
62854		
62855		
62856		

RETURN VALUE

62857
62858 Upon successful completion, *strptime()* shall return a pointer to the character following the last
62859 character parsed. Otherwise, a null pointer shall be returned.

ERRORS

62860
62861 No errors are defined.

EXAMPLES**Convert a Data-Plus-Time String to Broken-Down Time and Then into Seconds**

62863
62864 The following example demonstrates the use of *strptime()* to convert a string into broken-down
62865 time. The broken-down time is then converted into seconds since the Epoch using *mktime()*.

```
62866 #include <time.h>
62867 ...
62868 struct tm tm;
62869 time_t t;
```

```

62870     if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
62871         /* Handle error */;

62872     printf("year: %d; month: %d; day: %d;\n",
62873           tm.tm_year, tm.tm_mon, tm.tm_mday);
62874     printf("hour: %d; minute: %d; second: %d\n",
62875           tm.tm_hour, tm.tm_min, tm.tm_sec);
62876     printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);

62877     tm.tm_isdst = -1;      /* Not set by strptime(); tells mktime()
62878                           to determine whether daylight saving time
62879                           is in effect */

62880     t = mktime(&tm);
62881     if (t == -1)
62882         /* Handle error */;
62883     printf("seconds since the Epoch: %ld\n", (long) t);"

```

APPLICATION USAGE

Several “equivalent to” formats and the special processing of white-space characters are provided in order to ease the use of identical *format* strings for *strptime()* and *strptime()*.

Applications should use %Y (4-digit years including century) in preference to %y (2-digit years).

It is unspecified whether multiple calls to *strptime()* using the same **tm** structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to *strptime()* with a format and all data needed to completely specify the date and time being converted.

RATIONALE

None.

FUTURE DIRECTIONS

The *strptime()* function is expected to be mandatory in the next version of this volume of POSIX.1-200x.

SEE ALSO

fscanf(), *strptime()*, *time*

XBD <[time.h](#)>

CHANGE HISTORY

First released in Issue 4.

Issue 5

Moved from ENHANCED I18N to BASE.

The [ENOSYS] error is removed.

The exact meaning of the %y and %Oy specifiers is clarified in the DESCRIPTION.

Issue 6

The Open Group Corrigendum U033/5 is applied. The %r specifier description is reworded.

The normative text is updated to avoid use of the term “must” for application requirements.

The **restrict** keyword is added to the *strptime()* prototype for alignment with the ISO/IEC 9899:1999 standard.

The Open Group Corrigendum U047/2 is applied.

The DESCRIPTION is updated to use the terms “conversion specifier” and “conversion specification” for consistency with *strptime()*.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/133 is applied, adding the example to the

62915

EXAMPLES section.

62916

Issue 7

62917

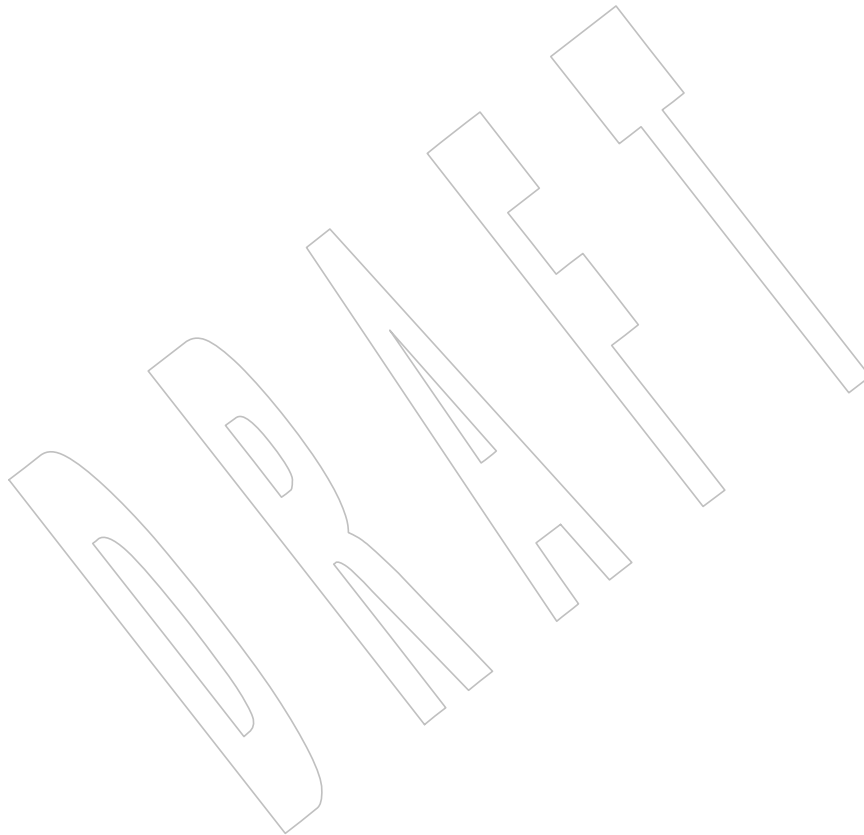
SD5-XSH-ERN-67 is applied, correcting the APPLICATION USAGE to remove the impression that %Y is 4-digit years.

62918

62919

Austin Group Interpretation 1003.1-2001 #041 is applied, updating the DESCRIPTION and APPLICATION USAGE sections.

62920



62921 **NAME**
 62922 `strrchr` — string scanning operation

62923 **SYNOPSIS**
 62924 `#include <string.h>`
 62925 `char *strrchr(const char *s, int c);`

62926 **DESCRIPTION**
 62927 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 62928 conflict between the requirements described here and the ISO C standard is unintentional. This
 62929 volume of POSIX.1-200x defers to the ISO C standard.

62930 The `strrchr()` function shall locate the last occurrence of `c` (converted to a **char**) in the string
 62931 pointed to by `s`. The terminating NUL character is considered to be part of the string.

62932 **RETURN VALUE**
 62933 Upon successful completion, `strrchr()` shall return a pointer to the byte or a null pointer if `c` does
 62934 not occur in the string.

62935 **ERRORS**
 62936 No errors are defined.

62937 **EXAMPLES**

62938 **Finding the Base Name of a File**

62939 The following example uses `strrchr()` to get a pointer to the base name of a file. The `strrchr()`
 62940 function searches backwards through the name of the file to find the last `'/'` character in `name`.
 62941 This pointer (plus one) will point to the base name of the file.

```
62942 #include <string.h>
62943 ...
62944 const char *name;
62945 char *basename;
62946 ...
62947 basename = strrchr(name, '/') + 1;
62948 ...
```

62949 **APPLICATION USAGE**
 62950 None.

62951 **RATIONALE**
 62952 None.

62953 **FUTURE DIRECTIONS**
 62954 None.

62955 **SEE ALSO**
 62956 [strchr\(\)](#)
 62957 XBD [<string.h>](#)

62958 **CHANGE HISTORY**
 62959 First released in Issue 1. Derived from Issue 1 of the SVID.

62960 **NAME**

62961 strsignal — get name of signal

62962 **SYNOPSIS**

```
62963 CX #include <string.h>
62964 char *strsignal(int signum);
```

62965 **DESCRIPTION**

62966 The *strsignal()* function shall map the signal number in *signum* to an implementation-defined
 62967 string and shall return a pointer to it. It shall use the same set of messages as the *psignal()*
 62968 function.

62969 The string pointed to shall not be modified by the application, but may be overwritten by a
 62970 subsequent call to *strsignal()* or *setlocale()*.

62971 The contents of the message strings returned by *strsignal()* should be determined by the setting
 62972 of the *LC_MESSAGES* category in the current locale.

62973 The implementation shall behave as if no function defined in this standard calls *strsignal()*.

62974 Since no return value is reserved to indicate an error, an application wishing to check for error
 62975 situations should set *errno* to 0, then call *strsignal()*, then check *errno*.

62976 The *strsignal()* function need not be reentrant. A function that is not required to be reentrant is
 62977 not required to be thread-safe.

62978 **RETURN VALUE**

62979 Upon successful completion, *strsignal()* shall return a pointer to a string. Otherwise, if *signum* is
 62980 not a valid signal number, the return value is unspecified.

62981 **ERRORS**

62982 No errors are defined.

62983 **EXAMPLES**

62984 None.

62985 **APPLICATION USAGE**

62986 None.

62987 **RATIONALE**

62988 If *signum* is not a valid signal number, some implementations return NULL, while for others the
 62989 *strsignal()* function returns a pointer to a string containing an unspecified message denoting an
 62990 unknown signal. POSIX.1-200x leaves this return value unspecified.

62991 **FUTURE DIRECTIONS**

62992 None.

62993 **SEE ALSO**

62994 *psiginfo()*, *setlocale()*

62995 XBD [<string.h>](#)

62996 **CHANGE HISTORY**

62997 First released in Issue 7.

62998 **NAME**62999 `strspn` — get length of a substring63000 **SYNOPSIS**63001 `#include <string.h>`63002 `size_t strspn(const char *s1, const char *s2);`63003 **DESCRIPTION**63004 CX The functionality described on this reference page is aligned with the ISO C standard. Any
63005 conflict between the requirements described here and the ISO C standard is unintentional. This
63006 volume of POSIX.1-200x defers to the ISO C standard.63007 The `strspn()` function shall compute the length (in bytes) of the maximum initial segment of the
63008 string pointed to by `s1` which consists entirely of bytes from the string pointed to by `s2`.63009 **RETURN VALUE**63010 The `strspn()` function shall return the computed length; no return value is reserved to indicate an
63011 error.63012 **ERRORS**

63013 No errors are defined.

63014 **EXAMPLES**

63015 None.

63016 **APPLICATION USAGE**

63017 None.

63018 **RATIONALE**

63019 None.

63020 **FUTURE DIRECTIONS**

63021 None.

63022 **SEE ALSO**63023 [*strcspn\(\)*](#)63024 XBD [*<string.h>*](#)63025 **CHANGE HISTORY**

63026 First released in Issue 1. Derived from Issue 1 of the SVID.

63027 **Issue 5**63028 The RETURN VALUE section is updated to indicate that `strspn()` returns the length of `s`, and not
63029 `s` itself as was previously stated.63030 **Issue 7**

63031 SD5-XSH-ERN-182 is applied. +

63032 **NAME**63033 *strstr* — find a substring63034 **SYNOPSIS**

63035 #include <string.h>

63036 char *strstr(const char *s1, const char *s2);

63037 **DESCRIPTION**63038 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
63039 conflict between the requirements described here and the ISO C standard is unintentional. This
63040 volume of POSIX.1-200x defers to the ISO C standard.63041 The *strstr()* function shall locate the first occurrence in the string pointed to by *s1* of the
63042 sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.63043 **RETURN VALUE**63044 Upon successful completion, *strstr()* shall return a pointer to the located string or a null pointer
63045 if the string is not found.63046 If *s2* points to a string with zero length, the function shall return *s1*.63047 **ERRORS**

63048 No errors are defined.

63049 **EXAMPLES**

63050 None.

63051 **APPLICATION USAGE**

63052 None.

63053 **RATIONALE**

63054 None.

63055 **FUTURE DIRECTIONS**

63056 None.

63057 **SEE ALSO**63058 [strchr\(\)](#)63059 XBD [<string.h>](#)63060 **CHANGE HISTORY**

63061 First released in Issue 3. Included for alignment with the ANSI C standard.

63062 NAME

63063 strtod, strtodf, strtold — convert a string to a double-precision number

63064 SYNOPSIS

63065 #include <stdlib.h>

63066 double strtod(const char *restrict *nptr*, char **restrict *endptr*);63067 float strtodf(const char *restrict *nptr*, char **restrict *endptr*);63068 long double strtold(const char *restrict *nptr*, char **restrict *endptr*);

63069 DESCRIPTION

63070 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63071 conflict between the requirements described here and the ISO C standard is unintentional. This
 63072 volume of POSIX.1-200x defers to the ISO C standard.

63073 These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**,
 63074 and **long double** representation, respectively. First, they decompose the input string into three
 63075 parts:

- 63076 1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace()*)
- 63077 2. A subject sequence interpreted as a floating-point constant or representing infinity or
 63078 NaN
- 63079 3. A final string of one or more unrecognized characters, including the terminating NUL
 63080 character of the input string

63081 Then they shall attempt to convert the subject sequence to a floating-point number, and return
 63082 the result.

63083 The expected form of the subject sequence is an optional plus or minus sign, then one of the
 63084 following:

- 63085 • A non-empty sequence of decimal digits optionally containing a radix character; then an
 63086 optional exponent part consisting of the character 'e' or the character 'E', optionally
 63087 followed by a '+' or '-' character, and then followed by one or more decimal digits
- 63088 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
 63089 character; then an optional binary exponent part consisting of the character 'p' or the
 63090 character 'P', optionally followed by a '+' or '-' character, and then followed by one or
 63091 more decimal digits
- 63092 • One of INF or INFINITY, ignoring case
- 63093 • One of NAN or NAN(*n-char-sequence_{opt}*), ignoring case in the NAN part, where:

```
63094 n-char-sequence :
63095     digit
63096     nondigit
63097     n-char-sequence digit
63098     n-char-sequence nondigit
```

63099 The subject sequence is defined as the longest initial subsequence of the input string, starting
 63100 with the first non-white-space character, that is of the expected form. The subject sequence
 63101 contains no characters if the input string is not of the expected form.

63102 If the subject sequence has the expected form for a floating-point number, the sequence of
 63103 characters starting with the first digit or the decimal-point character (whichever occurs first)
 63104 shall be interpreted as a floating constant of the C language, except that the radix character shall
 63105 be used in place of a period, and that if neither an exponent part nor a radix character appears in

63106 a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal
 63107 floating-point number, an exponent part of the appropriate type with value zero is assumed to
 63108 follow the last digit in the string. If the subject sequence begins with a minus sign, the sequence
 63109 shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an
 63110 infinity, if representable in the return type, else as if it were a floating constant that is too large
 63111 for the range of the return type. A character sequence NAN or NAN(*n-char-sequence_{opt}*) shall be
 63112 interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence
 63113 part that does not have the expected form; the meaning of the *n-char* sequences is
 63114 implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*,
 63115 provided that *endptr* is not a null pointer.

63116 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value
 63117 resulting from the conversion is correctly rounded.

63118 CX The radix character is defined in the locale of the process (category *LC_NUMERIC*). In the
 63119 POSIX locale, or in a locale where the radix character is not defined, the radix character shall
 63120 default to a period (`'.'`).

63121 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 63122 accepted.

63123 If the subject sequence is empty or does not have the expected form, no conversion shall be
 63124 performed; the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not
 63125 a null pointer.

63126 CX The *strtod()* function shall not change the setting of *errno* if successful.

63127 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 63128 for error situations should set *errno* to 0, then call *strtod()*, *strtof()*, or *strtold()*, then check *errno*.

63129 RETURN VALUE

63130 Upon successful completion, these functions shall return the converted value. If no conversion
 63131 could be performed, 0 shall be returned, and *errno* may be set to [EINVAL].

63132 If the correct value is outside the range of representable values, \pm HUGE_VAL, \pm HUGE_VALF, or
 63133 \pm HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to
 63134 [ERANGE].

63135 If the correct value would cause an underflow, a value whose magnitude is no greater than the
 63136 smallest normalized positive number in the return type shall be returned and *errno* set to
 63137 [ERANGE].

63138 ERRORS

63139 These functions shall fail if:

63140 CX [ERANGE] The value to be returned would cause overflow or underflow.

63141 These functions may fail if:

63142 CX [EINVAL] No conversion could be performed.

63143 EXAMPLES

63144 None.

63145 APPLICATION USAGE

63146 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
 63147 result is not exactly representable, the result should be one of the two numbers in the
 63148 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
 63149 extra stipulation that the error should have a correct sign for the current rounding direction.

63150 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in `<float.h>`)
 63151 significant digits, the result should be correctly rounded. If the subject sequence *D* has the

63152 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
 63153 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the
 63154 values of *L*, *D*, and *U* satisfy $L \leq D \leq U$. The result should be one of the (equal or adjacent)
 63155 values that would be obtained by correctly rounding *L* and *U* according to the current rounding
 63156 direction, with the extra stipulation that the error with respect to *D* should have a correct sign
 63157 for the current rounding direction.

63158 The changes to *strtod()* introduced by the ISO/IEC 9899:1999 standard can alter the behavior of
 63159 well-formed applications complying with the ISO/IEC 9899:1990 standard and thus earlier
 63160 versions of this standard. One such example would be:

```

63161 int
63162 what_kind_of_number (char *s)
63163 {
63164     char *endp;
63165     double d;
63166     long l;
63167
63168     d = strtod(s, &endp);
63169     if (s != endp && *endp == '\0')
63170         printf("It's a float with value %g\n", d);
63171     else
63172     {
63173         l = strtol(s, &endp, 0);
63174         if (s != endp && *endp == '\0')
63175             printf("It's an integer with value %ld\n", l);
63176         else
63177             return 1;
63178     }
63179     return 0;
  
```

63180 If the function is called with:

```
63181 what_kind_of_number ("0x10")
```

63182 an ISO/IEC 9899:1990 standard-compliant library will result in the function printing:

```
63183 It's an integer with value 16
```

63184 With the ISO/IEC 9899:1999 standard, the result is:

```
63185 It's a float with value 16
```

63186 The change in behavior is due to the inclusion of floating-point numbers in hexadecimal
 63187 notation without requiring that either a decimal point or the binary exponent be present.

63188 RATIONALE

63189 None.

63190 FUTURE DIRECTIONS

63191 None.

63192 SEE ALSO

63193 [fscanf\(\)](#), [isspace\(\)](#), [localeconv\(\)](#), [setlocale\(\)](#), [strtol\(\)](#)

63194 XBD Chapter 7 (on page 121), [<float.h>](#), [<stdlib.h>](#)

63195 **CHANGE HISTORY**

63196 First released in Issue 1. Derived from Issue 1 of the SVID.

63197 **Issue 5**63198 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.63199 **Issue 6**

63200 Extensions beyond the ISO C standard are marked.

63201 The following new requirements on POSIX implementations derive from alignment with the
63202 Single UNIX Specification:

- 63203
- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
63204 added if no conversion could be performed.

63205 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- 63206
- The *strtod()* function is updated.
 - The *strtof()* and *strtold()* functions are added.
 - The DESCRIPTION is extensively revised.

63209 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

63210 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/61 is applied, correcting the second
63211 paragraph in the RETURN VALUE section. This change clarifies the sign of the return value.63212 **Issue 7**

63213 Austin Group Interpretation 1003.1-2001 #015 is applied.

63214 **NAME**
 63215 `strtoimax, strtoumax` — convert string to integer type

63216 **SYNOPSIS**
 63217 `#include <inttypes.h>`
 63218 `intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,`
 63219 `int base);`
 63220 `uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,`
 63221 `int base);`

63222 **DESCRIPTION**
 63223 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63224 conflict between the requirements described here and the ISO C standard is unintentional. This
 63225 volume of POSIX.1-200x defers to the ISO C standard.
 63226 These functions shall be equivalent to the `strtol()`, `strtoll()`, `strtoul()`, and `strtoull()` functions,
 63227 except that the initial portion of the string shall be converted to `intmax_t` and `uintmax_t`
 63228 representation, respectively.

63229 **RETURN VALUE**
 63230 These functions shall return the converted value, if any.
 63231 If no conversion could be performed, zero shall be returned.
 63232 If the correct value is outside the range of representable values, `{INTMAX_MAX}`,
 63233 `{INTMAX_MIN}`, or `{UINTMAX_MAX}` shall be returned (according to the return type and sign
 63234 of the value, if any), and `errno` shall be set to `[ERANGE]`.

63235 **ERRORS**
 63236 These functions shall fail if:
 63237 `[ERANGE]` The value to be returned is not representable.
 63238 These functions may fail if:
 63239 `[EINVAL]` The value of `base` is not supported.

63240 **EXAMPLES**
 63241 None.

63242 **APPLICATION USAGE**
 63243 None.

63244 **RATIONALE**
 63245 None.

63246 **FUTURE DIRECTIONS**
 63247 None.

63248 **SEE ALSO**
 63249 [`strtol\(\)`, `strtoul\(\)`](#)
 63250 XBD [<inttypes.h>](#)

63251 **CHANGE HISTORY**
 63252 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

63253 **NAME**
 63254 `strtok, strtok_r` — split string into tokens

63255 **SYNOPSIS**

63256 `#include <string.h>`
 63257 `char *strtok(char *restrict s1, const char *restrict s2);`
 63258 CX `char *strtok_r(char *restrict s, const char *restrict sep,`
 63259 `char **restrict lasts);`

63260 **DESCRIPTION**

63261 CX For `strtok()`: The functionality described on this reference page is aligned with the ISO C
 63262 standard. Any conflict between the requirements described here and the ISO C standard is
 63263 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

63264 A sequence of calls to `strtok()` breaks the string pointed to by `s1` into a sequence of tokens, each
 63265 of which is delimited by a byte from the string pointed to by `s2`. The first call in the sequence
 63266 has `s1` as its first argument, and is followed by calls with a null pointer as their first argument.
 63267 The separator string pointed to by `s2` may be different from call to call.

63268 The first call in the sequence searches the string pointed to by `s1` for the first byte that is *not*
 63269 contained in the current separator string pointed to by `s2`. If no such byte is found, then there
 63270 are no tokens in the string pointed to by `s1` and `strtok()` shall return a null pointer. If such a byte
 63271 is found, it is the start of the first token.

63272 The `strtok()` function then searches from there for a byte that *is* contained in the current separator
 63273 string. If no such byte is found, the current token extends to the end of the string pointed to by
 63274 `s1`, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is
 63275 overwritten by a NUL character, which terminates the current token. The `strtok()` function saves
 63276 a pointer to the following byte, from which the next search for a token shall start.

63277 Each subsequent call, with a null pointer as the value of the first argument, starts searching from
 63278 the saved pointer and behaves as described above.

63279 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 63280 `strtok()`.

63281 CX The `strtok()` function need not be thread-safe. A function that is not required to be thread-safe is
 63282 not required to be reentrant.

63283 The `strtok_r()` function considers the null-terminated string `s` as a sequence of zero or more text
 63284 tokens separated by spans of one or more characters from the separator string `sep`. The
 63285 argument `lasts` points to a user-provided pointer which points to stored information necessary
 63286 for `strtok_r()` to continue scanning the same string.

63287 In the first call to `strtok_r()`, `s` points to a null-terminated string, `sep` to a null-terminated string of
 63288 separator characters, and the value pointed to by `lasts` is ignored. The `strtok_r()` function shall
 63289 return a pointer to the first character of the first token, write a null character into `s` immediately
 63290 following the returned token, and update the pointer to which `lasts` points.

63291 In subsequent calls, `s` is a null pointer and `lasts` shall be unchanged from the previous call so that
 63292 subsequent calls shall move through the string `s`, returning successive tokens until no tokens
 63293 remain. The separator string `sep` may be different from call to call. When no token remains in `s`, a
 63294 null pointer shall be returned.

strtok()63295 **RETURN VALUE**

63296 Upon successful completion, *strtok()* shall return a pointer to the first byte of a token. Otherwise,
63297 if there is no token, *strtok()* shall return a null pointer.

63298 CX The *strtok_r()* function shall return a pointer to the token found, or a null pointer when no token
63299 is found.

63300 **ERRORS**

63301 No errors are defined.

63302 **EXAMPLES**63303 **Searching for Word Separators**

63304 The following example searches for tokens separated by <space>s.

```
63305 #include <string.h>
63306 ...
63307 char *token;
63308 char *line = "LINE TO BE SEPARATED";
63309 char *search = " ";

63310 /* Token will point to "LINE". */
63311 token = strtok(line, search);

63312 /* Token will point to "TO". */
63313 token = strtok(NULL, search);
```

63314 **Breaking a Line**

63315 The following example uses *strtok()* to break a line into two character strings separated by any
63316 combination of <space>s, <tab>s, or <newline>s.

```
63317 #include <string.h>
63318 ...
63319 struct element {
63320     char *key;
63321     char *data;
63322 };
63323 ...
63324 char line[LINE_MAX];
63325 char *key, *data;
63326 ...
63327 key = strtok(line, " \n");
63328 data = strtok(NULL, " \n");
63329 ...
```

63330 **APPLICATION USAGE**

63331 The *strtok_r()* function is thread-safe and stores its state in a user-supplied buffer instead of
63332 possibly using a static data area that may be overwritten by an unrelated call from another
63333 thread.

63334 **RATIONALE**

63335 The *strtok()* function searches for a separator string within a larger string. It returns a pointer to
63336 the last substring between separator strings. This function uses static storage to keep track of
63337 the current string position between calls. The new function, *strtok_r()*, takes an additional
63338 argument, *lasts*, to keep track of the current position in the string.

63339

FUTURE DIRECTIONS

63340

None.

63341

SEE ALSO

63342

XBD <string.h>

63343

CHANGE HISTORY

63344

First released in Issue 1. Derived from Issue 1 of the SVID.

63345

Issue 5

63346

The *strtok_r()* function is included for alignment with the POSIX Threads Extension.

63347

A note indicating that the *strtok()* function need not be reentrant is added to the DESCRIPTION.

63348

Issue 6

63349

Extensions beyond the ISO C standard are marked.

63350

The *strtok_r()* function is marked as part of the Thread-Safe Functions option.

63351

In the DESCRIPTION, the note about reentrancy is expanded to cover thread-safety.

63352

The APPLICATION USAGE section is updated to include a note on the thread-safe function and its avoidance of possibly using a static data area.

63353

63354

The **restrict** keyword is added to the *strtok()* and *strtok_r()* prototypes for alignment with the ISO/IEC 9899:1999 standard.

63355

63356

Issue 7

63357

The *strtok_r()* function is moved from the Thread-Safe Functions option to the Base.

DRAFT

63358 **NAME**
 63359 `strtol, strtoll` — convert a string to a long integer

63360 **SYNOPSIS**
 63361 `#include <stdlib.h>`
 63362 `long strtol(const char *restrict str, char **restrict endptr, int base);`
 63363 `long long strtoll(const char *restrict str, char **restrict endptr,`
 63364 `int base)`

63365 **DESCRIPTION**

63366 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63367 conflict between the requirements described here and the ISO C standard is unintentional. This
 63368 volume of POSIX.1-200x defers to the ISO C standard.

63369 These functions shall convert the initial portion of the string pointed to by *str* to a type **long** and
 63370 **long long** representation, respectively. First, they decompose the input string into three parts:

- 63371 1. An initial, possibly empty, sequence of white-space characters (as specified by `isspace()`)
- 63372 2. A subject sequence interpreted as an integer represented in some radix determined by the
 63373 value of *base*
- 63374 3. A final string of one or more unrecognized characters, including the terminating NUL
 63375 character of the input string.

63376 Then they shall attempt to convert the subject sequence to an integer, and return the result.

63377 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
 63378 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
 63379 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
 63380 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
 63381 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 63382 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

63383 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 63384 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 63385 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
 63386 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
 63387 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
 63388 digits, following the sign if present.

63389 The subject sequence is defined as the longest initial subsequence of the input string, starting
 63390 with the first non-white-space character that is of the expected form. The subject sequence shall
 63391 contain no characters if the input string is empty or consists entirely of white-space characters,
 63392 or if the first non-white-space character is other than a sign or a permissible letter or digit.

63393 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
 63394 characters starting with the first digit shall be interpreted as an integer constant. If the subject
 63395 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
 63396 base for conversion, ascribing to each letter its value as given above. If the subject sequence
 63397 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to
 63398 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 63399 pointer.

63400 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 63401 accepted.

63402 If the subject sequence is empty or does not have the expected form, no conversion is performed;

63403 the value of *str* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 63404 pointer.

63405 CX The *strtol()* function shall not change the setting of *errno* if successful.

63406 Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned
 63407 on error and are also valid returns on success, an application wishing to check for error
 63408 situations should set *errno* to 0, then call *strtol()* or *strtoll()*, then check *errno*.

63409 RETURN VALUE

63410 Upon successful completion, these functions shall return the converted value, if any. If no
 63411 conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

63412 If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
 63413 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
 63414 *errno* set to [ERANGE].

63415 ERRORS

63416 These functions shall fail if:

63417 [ERANGE] The value to be returned is not representable.

63418 These functions may fail if:

63419 CX [EINVAL] The value of *base* is not supported.

63420 EXAMPLES

63421 None.

63422 APPLICATION USAGE

63423 None.

63424 RATIONALE

63425 None.

63426 FUTURE DIRECTIONS

63427 None.

63428 SEE ALSO

63429 *fscanf()*, *isalpha()*, *strtod()*

63430 XBD <stdlib.h>

63431 CHANGE HISTORY

63432 First released in Issue 1. Derived from Issue 1 of the SVID.

63433 Issue 5

63434 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

63435 Issue 6

63436 Extensions beyond the ISO C standard are marked.

63437 The following new requirements on POSIX implementations derive from alignment with the
 63438 Single UNIX Specification:

- 63439 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
- 63440 added if no conversion could be performed.

63441 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

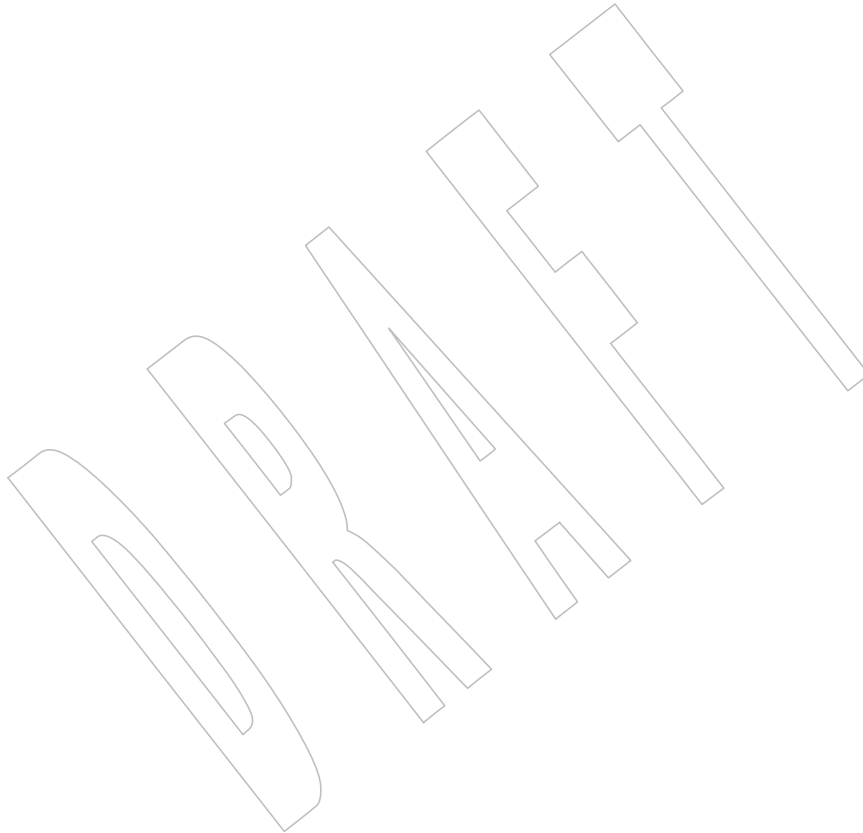
- 63442 • The *strtol()* prototype is updated.
- 63443 • The *strtoll()* function is added.

strtold()

63444 **NAME**
63445 **strtold** — convert a string to a double-precision number

63446 **SYNOPSIS**
63447 #include <stdlib.h>
63448 long double strtold(const char *restrict *nptr*, char **restrict *endptr*);

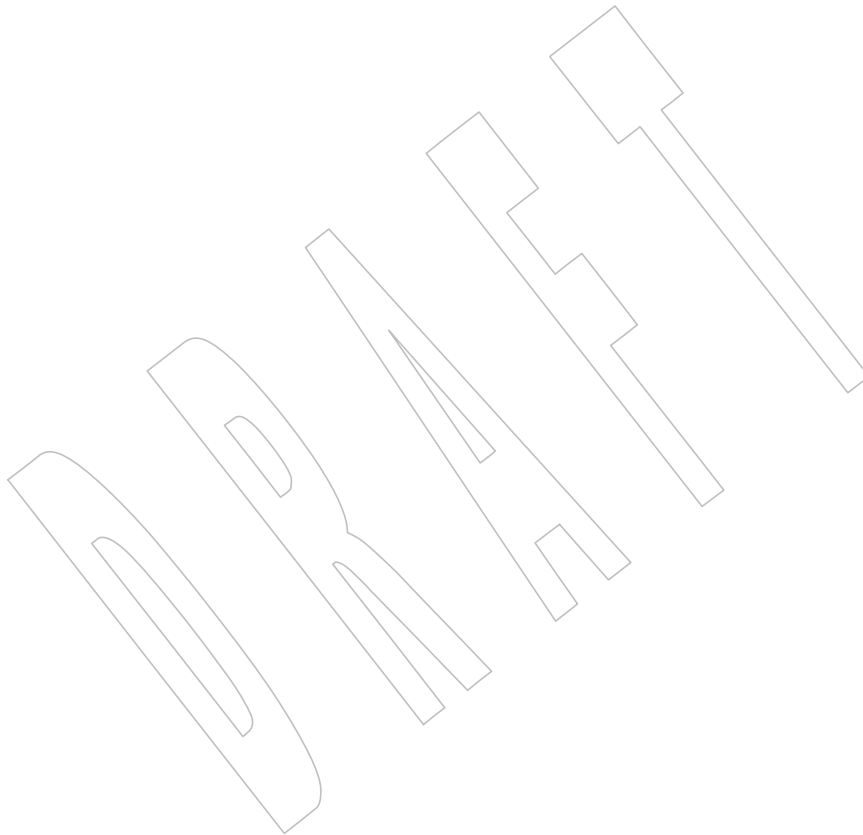
63449 **DESCRIPTION**
63450 Refer to *strtod()*.



63451 **NAME**
63452 strtoll — convert a string to a long integer

63453 **SYNOPSIS**
63454 #include <stdlib.h>
63455 long long strtoll(const char *restrict str, char **restrict endptr,
63456 int base);

63457 **DESCRIPTION**
63458 Refer to *strtol()*.



63459 **NAME**
 63460 `strtol, strtoull` — convert a string to an unsigned long

63461 **SYNOPSIS**
 63462 `#include <stdlib.h>`
 63463 `unsigned long strtol(const char *restrict str,`
 63464 `char **restrict endptr, int base);`
 63465 `unsigned long long strtoull(const char *restrict str,`
 63466 `char **restrict endptr, int base);`

63467 **DESCRIPTION**

63468 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 63469 conflict between the requirements described here and the ISO C standard is unintentional. This
 63470 volume of POSIX.1-200x defers to the ISO C standard.

63471 These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned**
 63472 **long** and **unsigned long long** representation, respectively. First, they decompose the input string
 63473 into three parts:

- 63474 1. An initial, possibly empty, sequence of white-space characters (as specified by `isspace()`)
 63475 2. A subject sequence interpreted as an integer represented in some radix determined by the
 63476 value of *base*
 63477 3. A final string of one or more unrecognized characters, including the terminating NUL
 63478 character of the input string

63479 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
 63480 result.

63481 If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant,
 63482 octal constant, or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A
 63483 decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An
 63484 octal constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to
 63485 '7' only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 63486 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

63487 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 63488 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 63489 by a '+' or '-' sign. The letters from 'a' (or 'A') to 'z' (or 'Z') inclusive are ascribed the
 63490 values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the
 63491 value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and
 63492 digits, following the sign if present.

63493 The subject sequence is defined as the longest initial subsequence of the input string, starting
 63494 with the first non-white-space character that is of the expected form. The subject sequence shall
 63495 contain no characters if the input string is empty or consists entirely of white-space characters,
 63496 or if the first non-white-space character is other than a sign or a permissible letter or digit.

63497 If the subject sequence has the expected form and the value of *base* is 0, the sequence of
 63498 characters starting with the first digit shall be interpreted as an integer constant. If the subject
 63499 sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the
 63500 base for conversion, ascribing to each letter its value as given above. If the subject sequence
 63501 begins with a minus sign, the value resulting from the conversion shall be negated. A pointer to
 63502 the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null
 63503 pointer.

63504 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
63505 accepted.

63506 If the subject sequence is empty or does not have the expected form, no conversion shall be
63507 performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr*
63508 is not a null pointer.

63509 CX The *strtol()* function shall not change the setting of *errno* if successful.

63510 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns
63511 on success, an application wishing to check for error situations should set *errno* to 0, then call
63512 *strtol()* or *strtoll()*, then check *errno*.

63513 RETURN VALUE

63514 Upon successful completion, these functions shall return the converted value, if any. If no
63515 CX conversion could be performed, 0 shall be returned and *errno* may be set to [EINVAL]. If the
63516 correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX}
63517 shall be returned and *errno* set to [ERANGE].

63518 ERRORS

63519 These functions shall fail if:

63520 CX [EINVAL] The value of *base* is not supported.

63521 [ERANGE] The value to be returned is not representable.

63522 These functions may fail if:

63523 CX [EINVAL] No conversion could be performed.

63524 EXAMPLES

63525 None.

63526 APPLICATION USAGE

63527 None.

63528 RATIONALE

63529 None.

63530 FUTURE DIRECTIONS

63531 None.

63532 SEE ALSO

63533 *fscanf()*, *isalpha()*, *strtod()*, *strtol()*

63534 XBD <stdlib.h>

63535 CHANGE HISTORY

63536 First released in Issue 4. Derived from the ANSI C standard.

63537 Issue 5

63538 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

63539 Issue 6

63540 Extensions beyond the ISO C standard are marked.

63541 The following new requirements on POSIX implementations derive from alignment with the
63542 Single UNIX Specification:

- 63543 • The [EINVAL] error condition is added for when the value of *base* is not supported.

63544 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
63545 added if no conversion could be performed.

strtoul()

63546

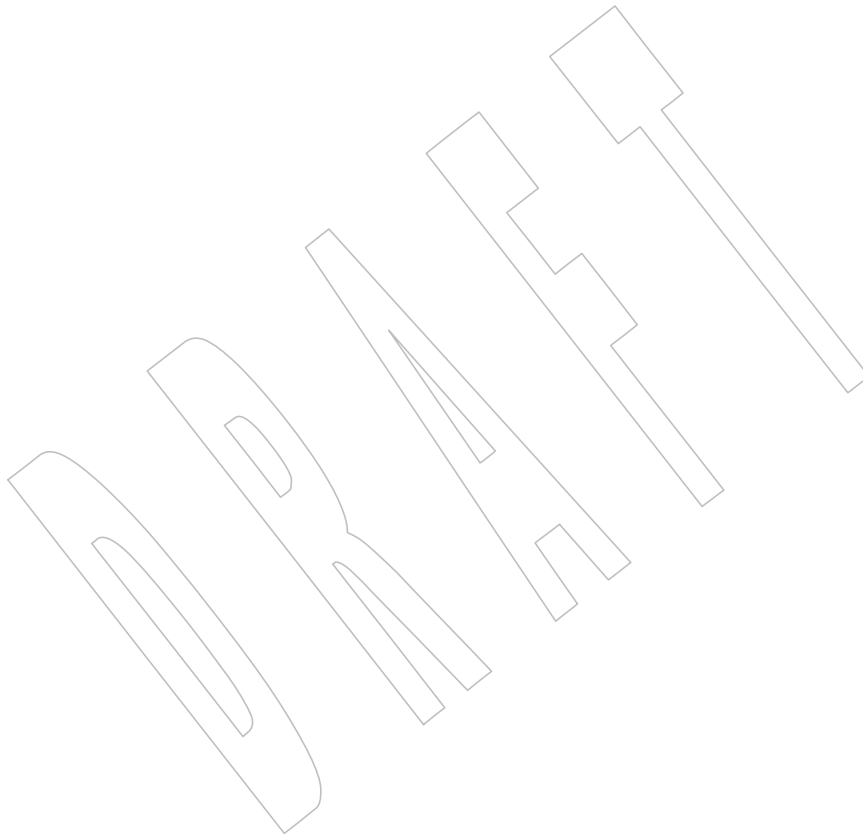
The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

63547

- The *strtoul()* prototype is updated.

63548

- The *strtoull()* function is added.

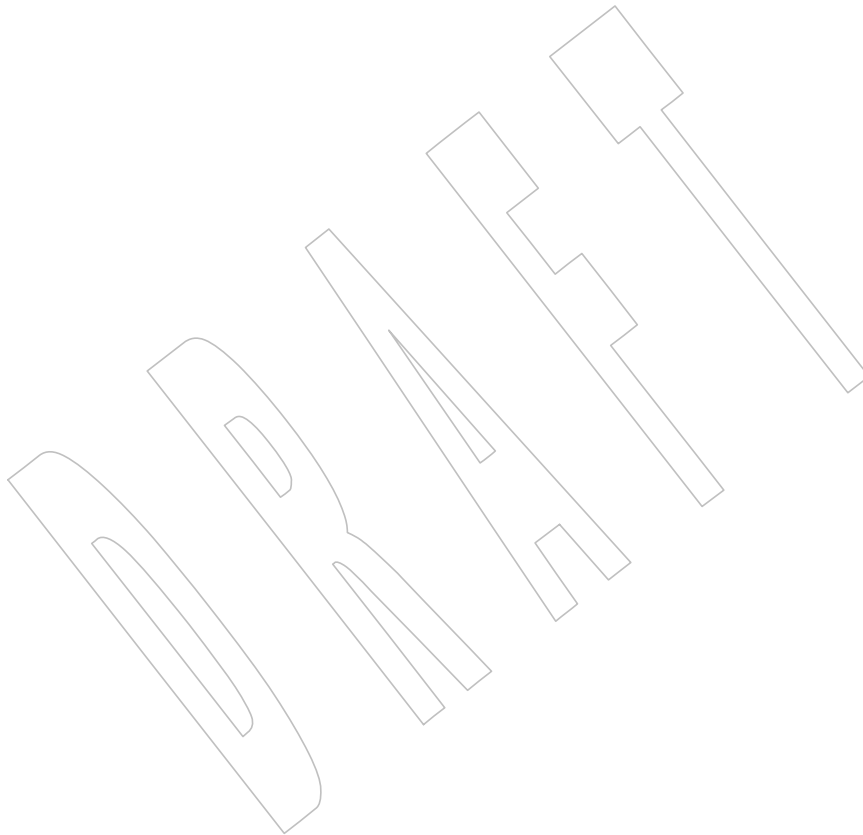


63549 **NAME**
63550 strtoumax — convert a string to an integer type

63551 **SYNOPSIS**
63552 #include <inttypes.h>

63553 uintmax_t strtoumax(const char *restrict *nptr*, char **restrict *endptr*,
63554 int *base*);

63555 **DESCRIPTION**
63556 Refer to *strtoimax()*.



63557 **NAME**
 63558 `strxfrm, strxfrm_l` — string transformation

63559 **SYNOPSIS**
 63560 `#include <string.h>`
 63561 `size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);`
 63562 CX `size_t strxfrm_l(char *restrict s1, const char *restrict s2,`
 63563 `size_t n, locale_t locale);`

63564 DESCRIPTION

63565 CX For `strxfrm()`: The functionality described on this reference page is aligned with the ISO C
 63566 standard. Any conflict between the requirements described here and the ISO C standard is
 63567 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

63568 CX The `strxfrm()` and `strxfrm_l()` functions shall transform the string pointed to by `s2` and place the
 63569 resulting string into the array pointed to by `s1`. The transformation is such that if `strcmp()` is
 63570 applied to two transformed strings, it shall return a value greater than, equal to, or less than 0,
 63571 corresponding to the result of `strcoll()` or `strcoll_l()`, respectively, applied to the same two
 63572 original strings with the same locale. No more than `n` bytes are placed into the resulting array
 63573 pointed to by `s1`, including the terminating NUL character. If `n` is 0, `s1` is permitted to be a null
 63574 pointer. If copying takes place between objects that overlap, the behavior is undefined.

63575 CX The `strxfrm()` and `strxfrm_l()` functions shall not change the setting of `errno` if successful.

63576 Since no return value is reserved to indicate an error, an application wishing to check for error
 63577 situations should set `errno` to 0, then call `strxfrm()` or `strxfrm_l()`, then check `errno`.

63578 RETURN VALUE

63579 CX Upon successful completion, `strxfrm()` and `strxfrm_l()` shall return the length of the
 63580 transformed string (not including the terminating NUL character). If the value returned is `n` or
 63581 more, the contents of the array pointed to by `s1` are unspecified.

63582 CX On error, `strxfrm()` and `strxfrm_l()` may set `errno` but no return value is reserved to indicate an
 63583 error.

63584 ERRORS

63585 These functions may fail if:

63586 CX [EINVAL] The string pointed to by the `s2` argument contains characters outside the
 63587 domain of the collating sequence.

63588 The `strxfrm_l()` function may fail if:

63589 CX [EINVAL] `locale` is not a valid locale object.

63590 EXAMPLES

63591 None.

63592 APPLICATION USAGE

63593 The transformation function is such that two transformed strings can be ordered by `strcmp()` as
 63594 appropriate to collating sequence information in the locale of the process (category
 63595 `LC_COLLATE`).

63596 The fact that when `n` is 0 `s1` is permitted to be a null pointer is useful to determine the size of the
 63597 `s1` array prior to making the transformation.

63598
63599
63600
63601
63602
63603
63604
63605
63606
63607
63608
63609
63610
63611
63612
63613
63614
63615
63616
63617
63618

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp(), *strcoll()*

XBD <**string.h**>

CHANGE HISTORY

First released in Issue 3. Included for alignment with the ISO C standard.

Issue 5

The DESCRIPTION is updated to indicate that *errno* does not change if the function is successful.

Issue 6

Extensions beyond the ISO C standard are marked.

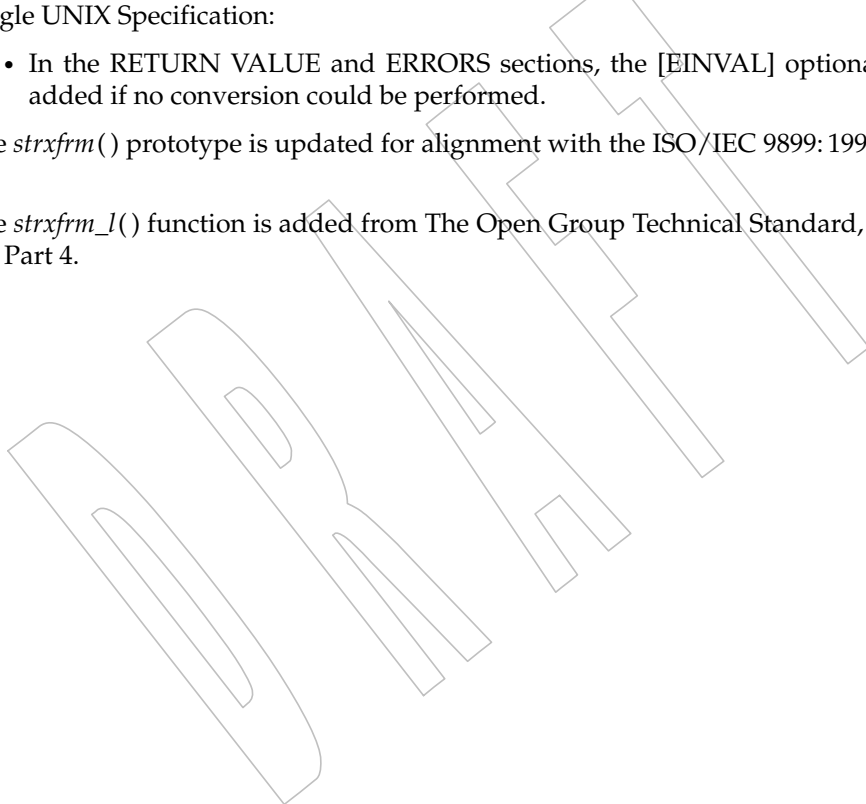
The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

The *strxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

Issue 7

The *strxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



63619 **NAME**63620 `swab` — swap bytes63621 **SYNOPSIS**

```
63622 XSI #include <unistd.h>
63623 void swab(const void *restrict src, void *restrict dest,
63624          ssize_t nbytes);
```

63625 **DESCRIPTION**

63626 The `swab()` function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to
 63627 by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, `swab()`
 63628 copies and exchanges *nbytes*–1 bytes and the disposition of the last byte is unspecified. If
 63629 copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is
 63630 negative, `swab()` does nothing.

63631 **RETURN VALUE**

63632 None.

63633 **ERRORS**

63634 No errors are defined.

63635 **EXAMPLES**

63636 None.

63637 **APPLICATION USAGE**

63638 None.

63639 **RATIONALE**

63640 None.

63641 **FUTURE DIRECTIONS**

63642 None.

63643 **SEE ALSO**63644 XBD [<unistd.h>](#)63645 **CHANGE HISTORY**

63646 First released in Issue 1. Derived from Issue 1 of the SVID.

63647 **Issue 6**

63648 The **restrict** keyword is added to the `swab()` prototype for alignment with the
 63649 ISO/IEC 9899:1999 standard.

63650 **NAME**
63651 `swprintf` — print formatted wide-character output

63652 **SYNOPSIS**

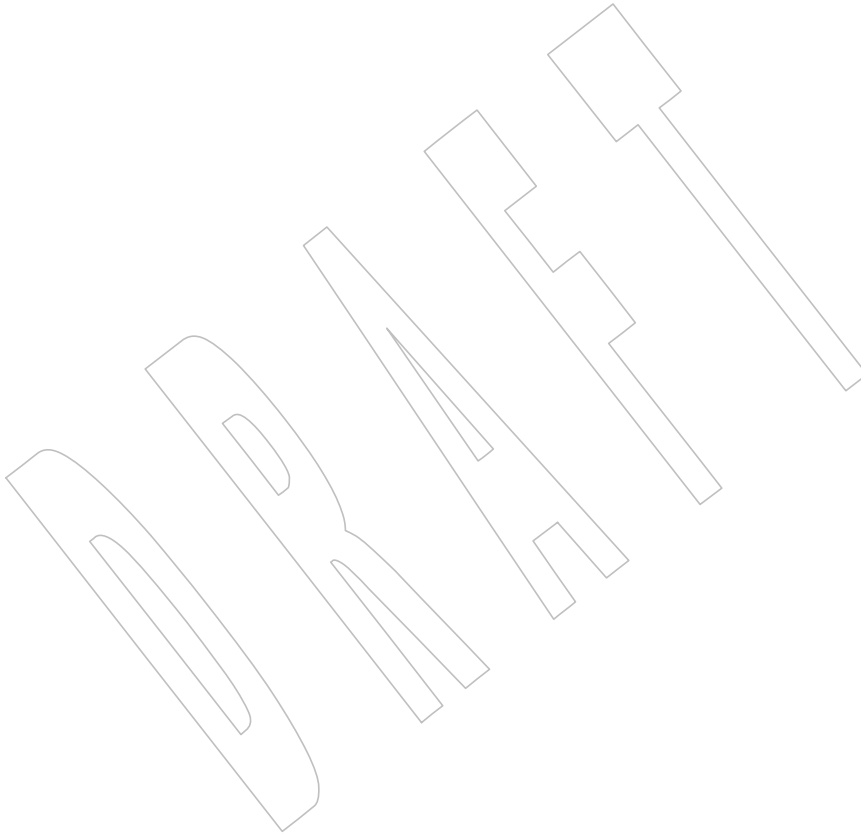
63653 `#include <stdio.h>`

63654 `#include <wchar.h>`

63655 `int swprintf(wchar_t *restrict ws, size_t n,`
63656 `const wchar_t *restrict format, ...);`

63657 **DESCRIPTION**

63658 Refer to *fwprintf()*.



63659 **NAME**
63660 `swscanf` — convert formatted wide-character input

63661 **SYNOPSIS**
63662 `#include <stdio.h>`
63663 `#include <wchar.h>`
63664 `int swscanf(const wchar_t *restrict ws,`
63665 `const wchar_t *restrict format, ...);`

63666 **DESCRIPTION**
63667 Refer to *fwscanf()*.



63668 **NAME**
 63669 `symlink, symlinkat` — make a symbolic link relative to directory file descriptor

63670 **SYNOPSIS**
 63671 `#include <unistd.h>`
 63672 `int symlink(const char *path1, const char *path2);`
 63673 `int symlinkat(const char *path1, int fd, const char *path2);`

63674 **DESCRIPTION**
 63675 The `symlink()` function shall create a symbolic link called `path2` that contains the string pointed to
 63676 by `path1` (`path2` is the name of the symbolic link created, `path1` is the string contained in the
 63677 symbolic link).

63678 The string pointed to by `path1` shall be treated only as a character string and shall not be
 63679 validated as a pathname.

63680 If the `symlink()` function fails for any reason other than [EIO], any file named by `path2` shall be
 63681 unaffected.

63682 The `symlinkat()` function shall be equivalent to the `symlink()` function except in the case where
 63683 `path2` specifies a relative path. In this case the symbolic link is created relative to the directory
 63684 associated with the file descriptor `fd` instead of the current working directory. It is unspecified
 63685 whether directory searches are permitted based on whether the file was opened with search
 63686 permission or on the current permissions of the directory underlying the file descriptor.

63687 If `symlinkat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working
 63688 directory is used and the behavior shall be identical to a call to `symlink()`.

63689 **RETURN VALUE**
 63690 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 63691 return `-1` and set `errno` to indicate the error.

63692 **ERRORS**
 63693 These functions shall fail if:

63694 63695 63696	[EACCES]	Write permission is denied in the directory where the symbolic link is being created, or search permission is denied for a component of the path prefix of <code>path2</code> .
63697	[EEXIST]	The <code>path2</code> argument names an existing file or symbolic link.
63698	[EIO]	An I/O error occurs while reading from or writing to the file system.
63699 63700	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <code>path2</code> argument.
63701 63702 63703 63704	[ENAMETOOLONG]	The length of the <code>path2</code> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX} or the length of the <code>path1</code> argument is longer than {SYMLINK_MAX}.
63705 63706	[ENOENT]	A component of <code>path2</code> does not name an existing file or <code>path2</code> is an empty string.
63707 63708 63709 63710 63711	[ENOSPC]	The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which shall contain the link, or the file system is out of file-allocation resources.

- 63712 [ENOTDIR] A component of the path prefix of *path2* is not a directory.
- 63713 [EROFS] The new symbolic link would reside on a read-only file system.
- 63714 The *symlinkat()* function shall fail if:
- 63715 [EBADF] The *path2* argument does not specify an absolute path and the *fd* argument is
63716 neither AT_FDCWD nor a valid file descriptor open for reading.

63717 These functions may fail if:

- 63718 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
63719 resolution of the *path2* argument.

63720 [ENAMETOOLONG]

- 63721 As a result of encountering a symbolic link in resolution of the *path2*
63722 argument, the length of the substituted pathname string exceeded
63723 {PATH_MAX} bytes (including the terminating null byte), or the length of the
63724 string pointed to by *path1* exceeded {SYMLINK_MAX}.

63725 The *symlinkat()* function may fail if:

- 63726 [ENOTDIR] The *path2* argument is not an absolute path and *fd* is neither AT_FDCWD nor a
63727 file descriptor associated with a directory.

63728 EXAMPLES

63729 None.

63730 APPLICATION USAGE

63731 Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a
63732 hard link guarantees the existence of a file, even after the original name has been removed. A
63733 symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not
63734 exist when the link is created. A symbolic link can cross file system boundaries.

63735 Normal permission checks are made on each component of the symbolic link pathname during
63736 its resolution.

63737 RATIONALE

63738 Since POSIX.1-200x does not require any association of file times with symbolic links, there is no
63739 requirement that file times be updated by *symlink()*.

63740 The purpose of the *symlinkat()* function is to create symbolic links in directories other than the
63741 current working directory without exposure to race conditions. Any part of the path of a file
63742 could be changed in parallel to a call to *symlink()*, resulting in unspecified behavior. By opening
63743 a file descriptor for the target directory and using the *symlinkat()* function it can be guaranteed
63744 that the created symbolic link is located relative to the desired directory.

63745 FUTURE DIRECTIONS

63746 None.

63747 SEE ALSO

63748 *fdopendir()*, *fstatat()*, *lchown()*, *link*, *open()*, *readlink()*, *rename()*, *unlink*

63749 XBD <[unistd.h](#)>

63750 CHANGE HISTORY

63751 First released in Issue 4, Version 2.

63752 Issue 5

63753 Moved from X/OPEN UNIX extension to BASE.

63754

Issue 6

63755

The following changes were made to align with the IEEE P1003.1a draft standard:

63756

- The DESCRIPTION text is updated.

63757

- The [ELOOP] optional error condition is added.

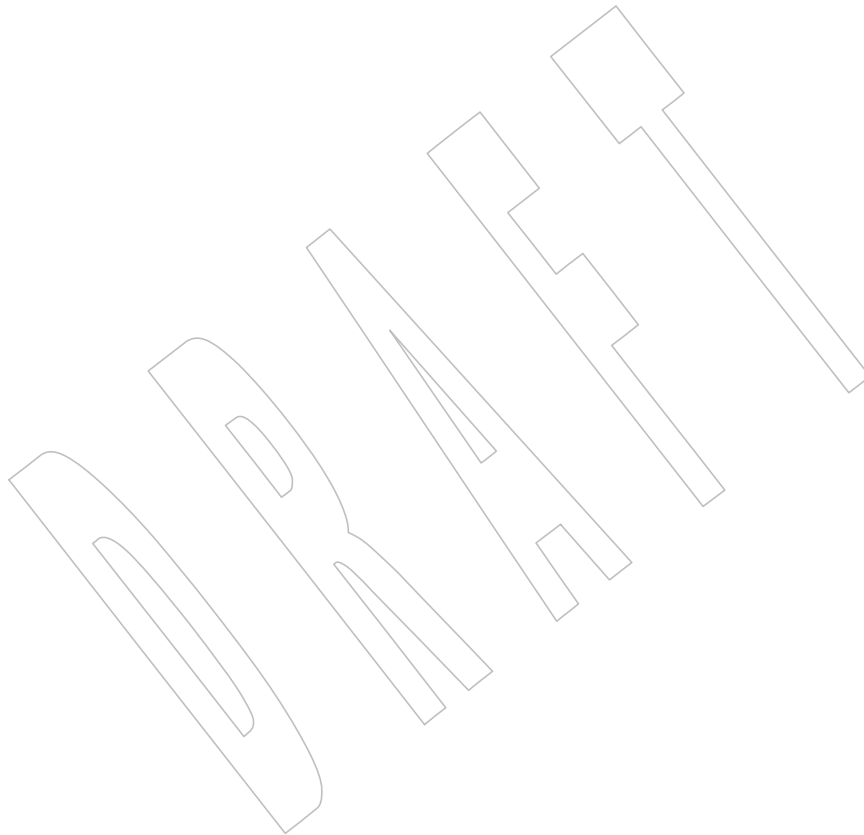
63758

Issue 7

63759

The *symlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 2.

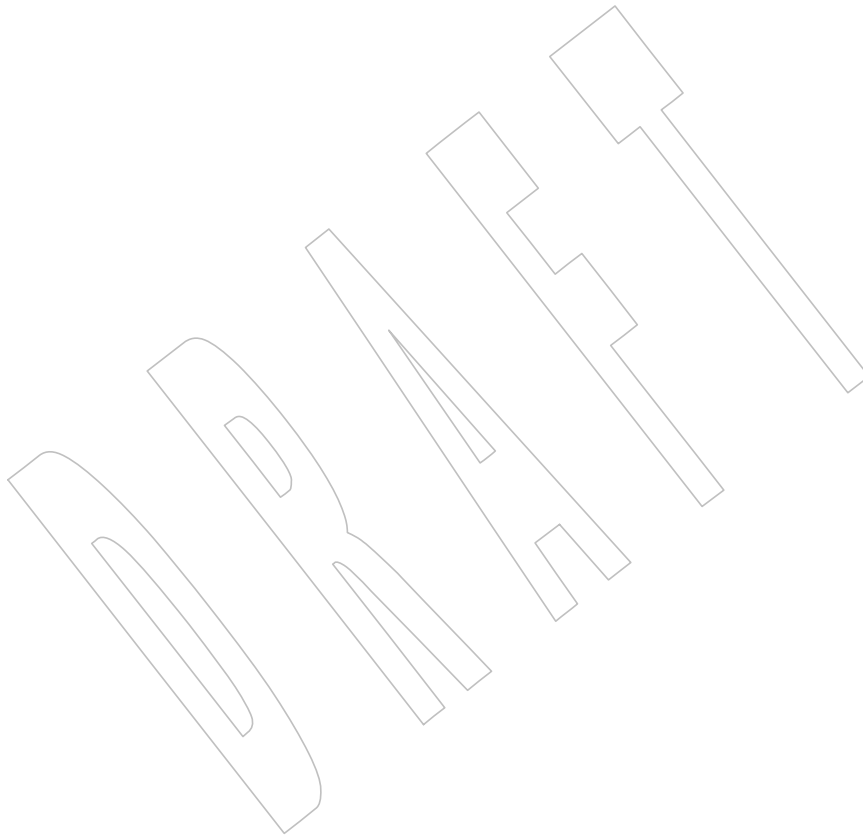
63760



63761 **NAME**
63762 `symlinkat` — make a symbolic link relative to directory file descriptor

63763 **SYNOPSIS**
63764 `#include <unistd.h>`
63765 `int symlinkat(const char *path1, int fd, const char *path2);`

63766 **DESCRIPTION**
63767 Refer to *symlink()*.



63768 **NAME**
 63769 sync — schedule file system updates

63770 **SYNOPSIS**

63771 XSI `#include <unistd.h>`
 63772 `void sync(void);`

63773 **DESCRIPTION**

63774 The *sync()* function shall cause all information in memory that updates file systems to be
 63775 scheduled for writing out to all file systems.

63776 The writing, although scheduled, is not necessarily complete upon return from *sync()*.

63777 **RETURN VALUE**

63778 The *sync()* function shall not return a value.

63779 **ERRORS**

63780 No errors are defined.

63781 **EXAMPLES**

63782 None.

63783 **APPLICATION USAGE**

63784 None.

63785 **RATIONALE**

63786 None.

63787 **FUTURE DIRECTIONS**

63788 None.

63789 **SEE ALSO**

63790 *fsync()*

63791 XBD `<unistd.h>`

63792 **CHANGE HISTORY**

63793 First released in Issue 4, Version 2.

63794 **Issue 5**

63795 Moved from X/OPEN UNIX extension to BASE.

63796 **NAME**
63797 sysconf — get configurable system variables

63798 **SYNOPSIS**
63799 #include <unistd.h>

63800 long sysconf(int name);

63801 **DESCRIPTION**

63802 The *sysconf()* function provides a method for the application to determine the current value of a
63803 configurable system limit or option (*variable*). The implementation shall support all of the
63804 variables listed in the following table and may support others.

63805 The *name* argument represents the system variable to be queried. The following table lists the
63806 minimal set of system variables from <limits.h> or <unistd.h> that can be returned by *sysconf()*,
63807 and the symbolic constants defined in <unistd.h> that are the corresponding values used for
63808 *name*.

Variable	Value of Name
{AIO_LISTIO_MAX}	_SC_AIO_LISTIO_MAX
{AIO_MAX}	_SC_AIO_MAX
{AIO_PRIO_DELTA_MAX}	_SC_AIO_PRIO_DELTA_MAX
{ARG_MAX}	_SC_ARG_MAX
{ATEXIT_MAX}	_SC_ATEXIT_MAX
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{CHILD_MAX}	_SC_CHILD_MAX
Clock ticks/second	_SC_CLK_TCK
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{DELAYTIMER_MAX}	_SC_DELAYTIMER_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{HOST_NAME_MAX}	_SC_HOST_NAME_MAX
{IOV_MAX}	_SC_IOV_MAX
{LINE_MAX}	_SC_LINE_MAX
{LOGIN_NAME_MAX}	_SC_LOGIN_NAME_MAX
{NGROUPS_MAX}	_SC_NGROUPS_MAX
Initial size of <i>getgrgid_r()</i> and <i>getgrnam_r()</i> data buffers	_SC_GETGR_R_SIZE_MAX
Initial size of <i>getpwuid_r()</i> and <i>getpwnam_r()</i> data buffers	_SC_GETPW_R_SIZE_MAX
{MQ_OPEN_MAX}	_SC_MQ_OPEN_MAX
{MQ_PRIO_MAX}	_SC_MQ_PRIO_MAX
{OPEN_MAX}	_SC_OPEN_MAX
_POSIX_ADVISORY_INFO	_SC_ADVISORY_INFO
_POSIX_BARRIERS	_SC_BARRIERS
_POSIX_ASYNCHRONOUS_IO	_SC_ASYNCHRONOUS_IO
_POSIX_CLOCK_SELECTION	_SC_CLOCK_SELECTION
_POSIX_CPUTIME	_SC_CPUTIME
_POSIX_FSYNC	_SC_FSYNC
_POSIX_IPV6	_SC_IPV6
_POSIX_JOB_CONTROL	_SC_JOB_CONTROL

	Variable	Value of Name
63844	_POSIX_MAPPED_FILES	_SC_MAPPED_FILES
63845	_POSIX_MEMLOCK	_SC_MEMLOCK
63846	_POSIX_MEMLOCK_RANGE	_SC_MEMLOCK_RANGE
63847	_POSIX_MEMORY_PROTECTION	_SC_MEMORY_PROTECTION
63848	_POSIX_MESSAGE_PASSING	_SC_MESSAGE_PASSING
63849	_POSIX_MONOTONIC_CLOCK	_SC_MONOTONIC_CLOCK
63850	_POSIX_PRIORITIZED_IO	_SC_PRIORITIZED_IO
63851	_POSIX_PRIORITY_SCHEDULING	_SC_PRIORITY_SCHEDULING
63852	_POSIX_RAW_SOCKETS	_SC_RAW_SOCKETS
63853	_POSIX_READER_WRITER_LOCKS	_SC_READER_WRITER_LOCKS
63854	_POSIX_REALTIME_SIGNALS	_SC_REALTIME_SIGNALS
63855	_POSIX_REGEX	_SC_REGEX
63856	_POSIX_SAVED_IDS	_SC_SAVED_IDS
63857	_POSIX_SEMAPHORES	_SC_SEMAPHORES
63858	_POSIX_SHARED_MEMORY_OBJECTS	_SC_SHARED_MEMORY_OBJECTS
63859	_POSIX_SHELL	_SC_SHELL
63860	_POSIX_SPAWN	_SC_SPAWN
63861	_POSIX_SPIN_LOCKS	_SC_SPIN_LOCKS
63862	_POSIX_SPORADIC_SERVER	_SC_SPORADIC_SERVER
63863	_POSIX_SS_REPL_MAX	_SC_SS_REPL_MAX
63864	_POSIX_SYNCHRONIZED_IO	_SC_SYNCHRONIZED_IO
63865	_POSIX_THREAD_ATTR_STACKADDR	_SC_THREAD_ATTR_STACKADDR
63866	_POSIX_THREAD_ATTR_STACKSIZE	_SC_THREAD_ATTR_STACKSIZE
63867	_POSIX_THREAD_CPUTIME	_SC_THREAD_CPUTIME
63868	_POSIX_THREAD_PRIO_INHERIT	_SC_THREAD_PRIO_INHERIT
63869	_POSIX_THREAD_PRIO_PROTECT	_SC_THREAD_PRIO_PROTECT
63870	_POSIX_THREAD_PRIORITY_SCHEDULING	_SC_THREAD_PRIORITY_SCHEDULING
63871	_POSIX_THREAD_PROCESS_SHARED	_SC_THREAD_PROCESS_SHARED
63872	_POSIX_THREAD_ROBUST_PRIO_INHERIT	_SC_THREAD_ROBUST_PRIO_INHERIT
63873	_POSIX_THREAD_ROBUST_PRIO_PROTECT	_SC_THREAD_ROBUST_PRIO_PROTECT
63874	_POSIX_THREAD_SAFE_FUNCTIONS	_SC_THREAD_SAFE_FUNCTIONS
63875	_POSIX_THREAD_SPARADIC_SERVER	_SC_THREAD_SPARADIC_SERVER
63876	_POSIX_THREADS	_SC_THREADS
63877	_POSIX_TIMEOUTS	_SC_TIMEOUTS
63878	_POSIX_TIMERS	_SC_TIMERS
63879	_POSIX_TRACE	_SC_TRACE
63880	_POSIX_TRACE_EVENT_FILTER	_SC_TRACE_EVENT_FILTER
63881	_POSIX_TRACE_EVENT_NAME_MAX	_SC_TRACE_EVENT_NAME_MAX
63882	_POSIX_TRACE_INHERIT	_SC_TRACE_INHERIT
63883	_POSIX_TRACE_LOG	_SC_TRACE_LOG
63884	_POSIX_TRACE_NAME_MAX	_SC_TRACE_NAME_MAX
63885	_POSIX_TRACE_SYS_MAX	_SC_TRACE_SYS_MAX
63886	_POSIX_TRACE_USER_EVENT_MAX	_SC_TRACE_USER_EVENT_MAX
63887	_POSIX_TYPED_MEMORY_OBJECTS	_SC_TYPED_MEMORY_OBJECTS
63888	_POSIX_VERSION	_SC_VERSION
63889	_POSIX_V7_ILP32_OFF32	_SC_V7_ILP32_OFF32
63890	_POSIX_V7_ILP32_OFFBIG	_SC_V7_ILP32_OFFBIG
63891	_POSIX_V7_LP64_OFF64	_SC_V7_LP64_OFF64
63892	_POSIX_V7_LPBIG_OFFBIG	_SC_V7_LPBIG_OFFBIG
63893		

	Variable	Value of Name
63894		
63895	OB	
63896		
63897		
63898		
63899		
63900		
63901		
63902		
63903		
63904		
63905		
63906		
63907		
63908		
63909		
63910		
63911		
63912		
63913		
63914		
63915		
63916		
63917		
63918		
63919		
63920		
63921		
63922		
63923		
63924		
63925		
63926		
63927		
63928		
63929		
63930		
63931		
63932		
63933		
63934		
63935		
63936		
63937		

RETURN VALUE

63938
63939 If *name* is an invalid value, *sysconf()* shall return -1 and set *errno* to indicate the error. If the
63940 variable corresponding to *name* has no limit, *sysconf()* shall return -1 without changing the value
63941 of *errno*. Note that indefinite limits do not imply infinite limits; see **<limits.h>**.

63942 Otherwise, *sysconf()* shall return the current variable value on the system. The value returned
63943 shall not be more restrictive than the corresponding value described to the application when it
63944 was compiled with the implementation's **<limits.h>** or **<unistd.h>**. The value shall not change
63945 during the lifetime of the calling process, except that *sysconf(SC_OPEN_MAX)* may return
63946 different values before and after a call to *setrlimit()* which changes the RLIMIT_NOFILE soft

63947

limit.

63948

If the variable corresponding to *name* is dependent on an unsupported option, the results are unspecified.

63949

63950

ERRORS

63951

The *sysconf()* function shall fail if:

63952

[EINVAL] The value of the *name* argument is invalid.

63953

EXAMPLES

63954

None.

63955

APPLICATION USAGE

63956

As -1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf()*, and, if it returns -1 , check to see if *errno* is non-zero.

63957

63958

63959

Application developers should check whether an option, such as `_POSIX_TRACE`, is supported prior to obtaining and using values for related variables, such as `_POSIX_TRACE_NAME_MAX`.

63960

63961

RATIONALE

63962

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to *pathconf()* and *fpathconf()*.

63963

63964

63965

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of POSIX.1-200x, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in `<limits.h>` or `<unistd.h>`.

63966

63967

63968

63969

63970

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

63971

63972

63973

Two examples are offered:

63974

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it were to rely solely on the `<unistd.h>` value published in this volume of POSIX.1-200x.

63975

63976

63977

63978

2. International applications vendors occasionally require knowledge of the number of clock ticks per second. Without these facilities, they would be required to either distribute their applications partially in source form or to have 50 Hz and 60 Hz versions for the various countries in which they operate.

63979

63980

63981

63982

It is the knowledge that many applications are actually distributed widely in executable form that leads to this facility. If limited to the most restrictive values in the headers, such applications would have to be prepared to accept the most limited environments offered by the smallest microcomputers. Although this is entirely portable, there was a consensus that they should be able to take advantage of the facilities offered by large systems, without the restrictions associated with source and object distributions.

63983

63984

63985

63986

63987

63988

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application developer.

63989

63990

63991

63992

63993

This runtime facility is not meant to provide ever-changing values that applications have to

63994 check multiple times. The values are seen as changing no more frequently than once per system
 63995 initialization, such as by a system administrator or operator with an automatic configuration
 63996 program. This volume of POSIX.1-200x specifies that they shall not change within the lifetime of
 63997 the process.

63998 Some values apply to the system overall and others vary at the file system or directory level. The
 63999 latter are described in *fpathconf()*.

64000 Note that all values returned must be expressible as integers. String values were considered, but
 64001 the additional flexibility of this approach was rejected due to its added complexity of
 64002 implementation and use.

64003 Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say,
 64004 allocate arrays. The *sysconf()* function returns a negative value to show that this symbolic
 64005 constant is not even defined in this case.

64006 Similar to *pathconf()*, this permits the implementation not to have a limit. When one resource is
 64007 infinite, returning an error indicating that some other resource limit has been reached is
 64008 conforming behavior.

64009 FUTURE DIRECTIONS

64010 None.

64011 SEE ALSO

64012 *confstr()*, *fpathconf()*

64013 XBD [<limits.h>](#), [<unistd.h>](#)

64014 XCU *getconf*

64015 CHANGE HISTORY

64016 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64017 Issue 5

64018 The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX
 64019 Threads Extension.

64020 The `_XBS_` variables and name values are added to the table of system variables in the
 64021 DESCRIPTION. These are all marked EX.

64022 Issue 6

64023 The symbol `CLK_TCK` is obsolescent and removed. It is replaced with the phrase “clock ticks
 64024 per second”.

64025 The symbol {`PASS_MAX`} is removed.

64026 The following changes were made to align with the IEEE P1003.1a draft standard:

- 64027 • Table entries are added for the following variables: `_SC_REGEX`, `_SC_SHELL`,
- 64028 `_SC_REGEX_VERSION`, `_SC_SYMLINK_MAX`.

64029 The following *sysconf()* variables and their associated names are added for alignment with
 64030 IEEE Std 1003.1d-1999:

```

64031     _POSIX_ADVISORY_INFO
64032     _POSIX_CPUTIME
64033     _POSIX_SPAWN
64034     _POSIX_SPORADIC_SERVER
64035     _POSIX_THREAD_CPUTIME
64036     _POSIX_THREAD_SPORADIC_SERVER
64037     _POSIX_TIMEOUTS
  
```

64038 The following changes are made to the DESCRIPTION for alignment with IEEE Std 1003.1j-2000:

64039 • A statement expressing the dependency of support for some system variables on

64040 implementation options is added.

64041 • The following system variables are added:

64042 _POSIX_BARRIERS

64043 _POSIX_CLOCK_SELECTION

64044 _POSIX_MONOTONIC_CLOCK

64045 _POSIX_READER_WRITER_LOCKS

64046 _POSIX_SPIN_LOCKS

64047 _POSIX_TYPED_MEMORY_OBJECTS

64048 The following system variables are added for alignment with IEEE Std 1003.2d-1994:

64049 _POSIX2_PBS

64050 _POSIX2_PBS_ACCOUNTING

64051 _POSIX2_PBS_LOCATE

64052 _POSIX2_PBS_MESSAGE

64053 _POSIX2_PBS_TRACK

64054 The following *sysconf()* variables and their associated names are added for alignment with

64055 IEEE Std 1003.1q-2000:

64056 _POSIX_TRACE

64057 _POSIX_TRACE_EVENT_FILTER

64058 _POSIX_TRACE_INHERIT

64059 _POSIX_TRACE_LOG

64060 The macros associated with the *c89* programming models are marked LEGACY, and new

64061 equivalent macros associated with *c99* are introduced.

64062 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/62 is applied, updating the

64063 DESCRIPTION to denote that the *_PC** and *_SC** symbols are now required to be supported. A

64064 corresponding change has been made in the Base Definitions volume of POSIX.1-200x. The

64065 deletion in the second paragraph removes some duplicated text. Additional symbols that were

64066 erroneously omitted from this reference page have been added.

64067 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/63 is applied, making it clear in the

64068 RETURN VALUE section that the value returned for *sysconf(_SC_OPEN_MAX)* may change if a

64069 call to *setrlimit()* adjusts the RLIMIT_NOFILE soft limit.

64070 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/134 is applied, updating the

64071 DESCRIPTION to remove an erroneous entry for *_POSIX_SYMLOOP_MAX*. This corrects an

64072 error in IEEE Std 1003.1-2001/Cor 1-2002.

64073 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/135 is applied, removing

64074 *_POSIX_FILE_LOCKING*, *_POSIX_MULTI_PROCESS*, *_POSIX2_C_VERSION*, and

64075 *_XOPEN_XCU_VERSION* (and their associated *_SC_** variables) from the DESCRIPTION and

64076 APPLICATION USAGE sections.

64077 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/136 is applied, adding the following

64078 constants (and their associated *_SC_** variables) to the DESCRIPTION:

64079 _POSIX_SS_REPL_MAX
 64080 _POSIX_TRACE_EVENT_NAME_MAX
 64081 _POSIX_TRACE_NAME_MAX
 64082 _POSIX_TRACE_SYS_MAX
 64083 _POSIX_TRACE_USER_EVENT_MAX

64084 The RETURN VALUE and APPLICATION USAGE sections are updated to note that if variables
 64085 are dependent on unsupported options, the results are unspecified.

64086 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/137 is applied, removing
 64087 _REGEX_VERSION and _SC_REGEX_VERSION.

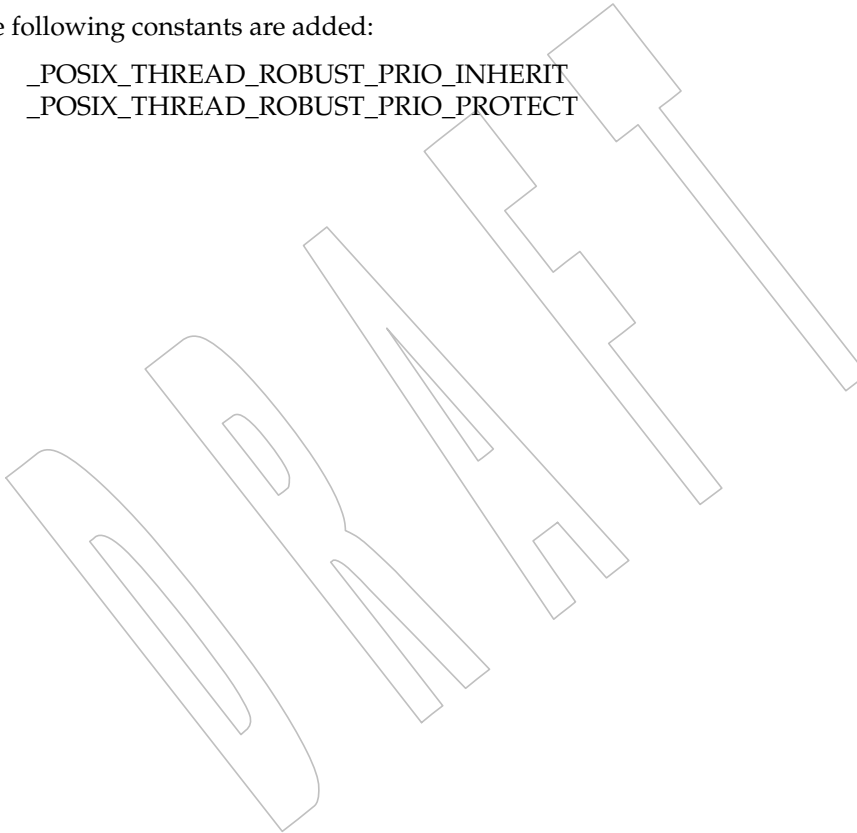
64088 **Issue 7**

64089 SD5-XSH-ERN-166 is applied, changing “Maximum size” to “Initial size” for the “Maximum +
 64090 size of ...” entries in the table in the DESCRIPTION. +

64091 The variables for the supported programming environments are updated to be V7 and the
 64092 LEGACY variables are removed.

64093 The following constants are added:

64094 _POSIX_THREAD_ROBUST_PRIO_INHERIT
 64095 _POSIX_THREAD_ROBUST_PRIO_PROTECT



64096 **NAME**
64097 syslog — log a message

64098 **SYNOPSIS**

64099 XSI `#include <syslog.h>`
64100 `void syslog(int priority, const char *message, ... /* argument */);`

64101 **DESCRIPTION**

64102 Refer to *closelog()*.

64103 **NAME**

64104 system — issue a command

64105 **SYNOPSIS**

64106 #include <stdlib.h>

64107 int system(const char *command);

64108 **DESCRIPTION**

64109 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64110 conflict between the requirements described here and the ISO C standard is unintentional. This
 64111 volume of POSIX.1-200x defers to the ISO C standard.

64112 If *command* is a null pointer, the *system()* function shall determine whether the host environment
 64113 has a command processor. If *command* is not a null pointer, the *system()* function shall pass the
 64114 string pointed to by *command* to that command processor to be executed in an implementation-
 64115 defined manner; this might then cause the program calling *system()* to behave in a non-
 64116 conforming manner or to terminate.

64117 CX The *system()* function shall behave as if a child process were created using *fork()*, and the child
 64118 process invoked the *sh* utility using *execl()* as follows:

64119

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

64120 where <shell path> is an unspecified pathname for the *sh* utility. It is unspecified whether the
 64121 handlers registered with *pthread_atfork()* are called as part of the creation of the child process.

64122 The *system()* function shall ignore the SIGINT and SIGQUIT signals, and shall block the
 64123 SIGCHLD signal, while waiting for the command to terminate. If this might cause the
 64124 application to miss a signal that would have killed it, then the application should examine the
 64125 return value from *system()* and take whatever action is appropriate to the application if the
 64126 command terminated due to receipt of a signal.

64127 The *system()* function shall not affect the termination status of any child of the calling processes
 64128 other than the process or processes it itself creates.

64129 The *system()* function shall not return until the child process has terminated.

64130 The *system()* function need not be thread-safe. A function that is not required to be thread-safe is
 64131 not required to be reentrant.

64132 **RETURN VALUE**

64133 If *command* is a null pointer, *system()* shall return non-zero to indicate that a command processor
 64134 CX is available, or zero if none is available. The *system()* function shall always return non-zero
 64135 when *command* is NULL.

64136 CX If *command* is not a null pointer, *system()* shall return the termination status of the command
 64137 language interpreter in the format specified by *waitpid()*. The termination status shall be as
 64138 defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents
 64139 the command language interpreter from executing after the child process is created, the return
 64140 value from *system()* shall be as if the command language interpreter had terminated using
 64141 *exit(127)* or *_exit(127)*. If a child process cannot be created, or if the termination status for the
 64142 command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to
 64143 indicate the error.

64144 **ERRORS**64145 CX The *system()* function may set *errno* values as described by *fork()*.64146 In addition, *system()* may fail if:64147 CX [ECHILD] The status of the child process created by *system()* is no longer available.64148 **EXAMPLES**

64149 None.

64150 **APPLICATION USAGE**64151 If the return value of *system()* is not -1 , its value can be decoded through the use of the macros
64152 described in `<sys/wait.h>`. For convenience, these macros are also provided in `<stdlib.h>`.64153 Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting
64154 for the child to terminate, the handling of signals in the executed command is as specified by
64155 *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system()* is
64156 called, then the child is started with SIGINT handling set to SIG_DFL.64157 Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two
64158 processes reading from the same terminal, for example) when the executed command ignores or
64159 catches one of the signals. It is also usually the correct action when the user has given a
64160 command to the application to be executed synchronously (as in the '!' command in many
64161 interactive applications). In either case, the signal should be delivered only to the child process,
64162 not to the application itself. There is one situation where ignoring the signals might have less
64163 than the desired effect. This is when the application uses *system()* to perform some task invisible
64164 to the user. If the user typed the interrupt character (" ^C ", for example) while *system()* is being
64165 used in this way, one would expect the application to be killed, but only the executed command
64166 is killed. Applications that use *system()* in this way should carefully check the return status from
64167 *system()* to see if the executed command was successful, and should take appropriate action
64168 when the command fails.64169 Blocking SIGCHLD while waiting for the child to terminate prevents the application from
64170 catching the signal and obtaining status from *system()*'s child process before *system()* can get the
64171 status itself.64172 The context in which the utility is ultimately executed may differ from that in which *system()*
64173 was called. For example, file descriptors that have the FD_CLOEXEC flag set are closed, and the
64174 process ID and parent process ID are different. Also, if the executed utility changes its
64175 environment variables or its current working directory, that change is not reflected in the caller's
64176 context.64177 There is no defined way for an application to find the specific path for the shell. However,
64178 *confstr()* can provide a value for *PATH* that is guaranteed to find the *sh* utility.64179 Using the *system()* function in more than one thread in a process or when the SIGCHLD signal is
64180 being manipulated by more than one thread in a process may produce unexpected results.64181 **RATIONALE**64182 The *system()* function should not be used by programs that have set user (or group) ID
64183 privileges. The *fork()* and *exec* family of functions (except *execlp()* and *execvp()*), should be used
64184 instead. This prevents any unforeseen manipulation of the environment of the user that could
64185 cause execution of commands not anticipated by the calling program.64186 There are three levels of specification for the *system()* function. The ISO C standard gives the
64187 most basic. It requires that the function exists, and defines a way for an application to query
64188 whether a command language interpreter exists. It says nothing about the command language
64189 or the environment in which the command is interpreted.64190 POSIX.1-200x places additional restrictions on *system()*. It requires that if there is a command
64191 language interpreter, the environment must be as specified by *fork()* and *exec*. This ensures, for

64192 example, that *close-on-exec* works, that file locks are not inherited, and that the process ID is
 64193 different. It also specifies the return value from *system()* when the command line can be run,
 64194 thus giving the application some information about the command's completion status.

64195 Finally, POSIX.1-200x requires the command to be interpreted as in the shell command language
 64196 defined in the Shell and Utilities volume of POSIX.1-200x.

64197 Note that, *system(NULL)* is required to return non-zero, indicating that there is a command
 64198 language interpreter. At first glance, this would seem to conflict with the ISO C standard which
 64199 allows *system(NULL)* to return zero. There is no conflict, however. A system must have a
 64200 command language interpreter, and is non-conforming if none is present. It is therefore
 64201 permissible for the *system()* function on such a system to implement the behavior specified by
 64202 the ISO C standard as long as it is understood that the implementation does not conform to
 64203 POSIX.1-200x if *system(NULL)* returns zero.

64204 It was explicitly decided that when *command* is NULL, *system()* should not be required to check
 64205 to make sure that the command language interpreter actually exists with the correct mode, that
 64206 there are enough processes to execute it, and so on. The call *system(NULL)* could, theoretically,
 64207 check for such problems as too many existing child processes, and return zero. However, it
 64208 would be inappropriate to return zero due to such a (presumably) transient condition. If some
 64209 condition exists that is not under the control of this application and that would cause any
 64210 *system()* call to fail, that system has been rendered non-conforming.

64211 Early drafts required, or allowed, *system()* to return with *errno* set to [EINTR] if it was
 64212 interrupted with a signal. This error return was removed, and a requirement that *system()* not
 64213 return until the child has terminated was added. This means that if a *waitpid()* call in *system()*
 64214 exits with *errno* set to [EINTR], *system()* must reissue the *waitpid()*. This change was made for
 64215 two reasons:

- 64216 1. There is no way for an application to clean up if *system()* returns [EINTR], short of calling
 64217 *wait()*, and that could have the undesirable effect of returning the status of children other
 64218 than the one started by *system()*.
- 64219 2. While it might require a change in some historical implementations, those
 64220 implementations already have to be changed because they use *wait()* instead of *waitpid()*.

64221 Note that if the application is catching SIGCHLD signals, it will receive such a signal before a
 64222 successful *system()* call returns.

64223 To conform to POSIX.1-200x, *system()* must use *waitpid()*, or some similar function, instead of
 64224 *wait()*.

64225 The following code sample illustrates how *system()* might be implemented on an
 64226 implementation conforming to POSIX.1-200x.

```
64227 #include <signal.h>
64228 int system(const char *cmd)
64229 {
64230     int stat;
64231     pid_t pid;
64232     struct sigaction sa, savintr, savequit;
64233     sigset_t saveblock;
64234     if (cmd == NULL)
64235         return(1);
64236     sa.sa_handler = SIG_IGN;
64237     sigemptyset(&sa.sa_mask);
64238     sa.sa_flags = 0;
64239     sigemptyset(&savintr.sa_mask);
64240     sigemptyset(&savequit.sa_mask);
```

```

64241     sigaction(SIGINT, &sa, &saveintr);
64242     sigaction(SIGQUIT, &sa, &savequit);
64243     sigaddset(&sa.sa_mask, SIGCHLD);
64244     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
64245     if ((pid = fork()) == 0) {
64246         sigaction(SIGINT, &saveintr, (struct sigaction *)0);
64247         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
64248         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
64249         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
64250         _exit(127);
64251     }
64252     if (pid == -1) {
64253         stat = -1; /* errno comes from fork() */
64254     } else {
64255         while (waitpid(pid, &stat, 0) == -1) {
64256             if (errno != EINTR){
64257                 stat = -1;
64258                 break;
64259             }
64260         }
64261     }
64262     sigaction(SIGINT, &saveintr, (struct sigaction *)0);
64263     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
64264     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
64265     return(stat);
64266 }

```

64267 Note that, while a particular implementation of *system()* (such as the one above) can assume a
64268 particular path for the shell, such a path is not necessarily valid on another system. The above
64269 example is not portable, and is not intended to be.

64270 One reviewer suggested that an implementation of *system()* might want to use an environment
64271 variable such as *SHELL* to determine which command interpreter to use. The supposed
64272 implementation would use the default command interpreter if the one specified by the
64273 environment variable was not available. This would allow a user, when using an application that
64274 prompts for command lines to be processed using *system()*, to specify a different command
64275 interpreter. Such an implementation is discouraged. If the alternate command interpreter did not
64276 follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-200x,
64277 then changing *SHELL* would render *system()* non-conforming. This would affect applications
64278 that expected the specified behavior from *system()*, and since the Shell and Utilities volume of
64279 POSIX.1-200x does not mention that *SHELL* affects *system()*, the application would not know
64280 that it needed to unset *SHELL*.

64281 FUTURE DIRECTIONS

64282 None.

64283 SEE ALSO

64284 *exec*, *pipe()*, *pthread_atfork()*, *wait*

64285 XBD [<limits.h>](#), [<signal.h>](#), [<stdlib.h>](#), [<sys/wait.h>](#)

64286 XCU *sh*

64287 CHANGE HISTORY

64288 First released in Issue 1. Derived from Issue 1 of the SVID.

64289

Issue 6

64290

Extensions beyond the ISO C standard are marked.

64291

Issue 7

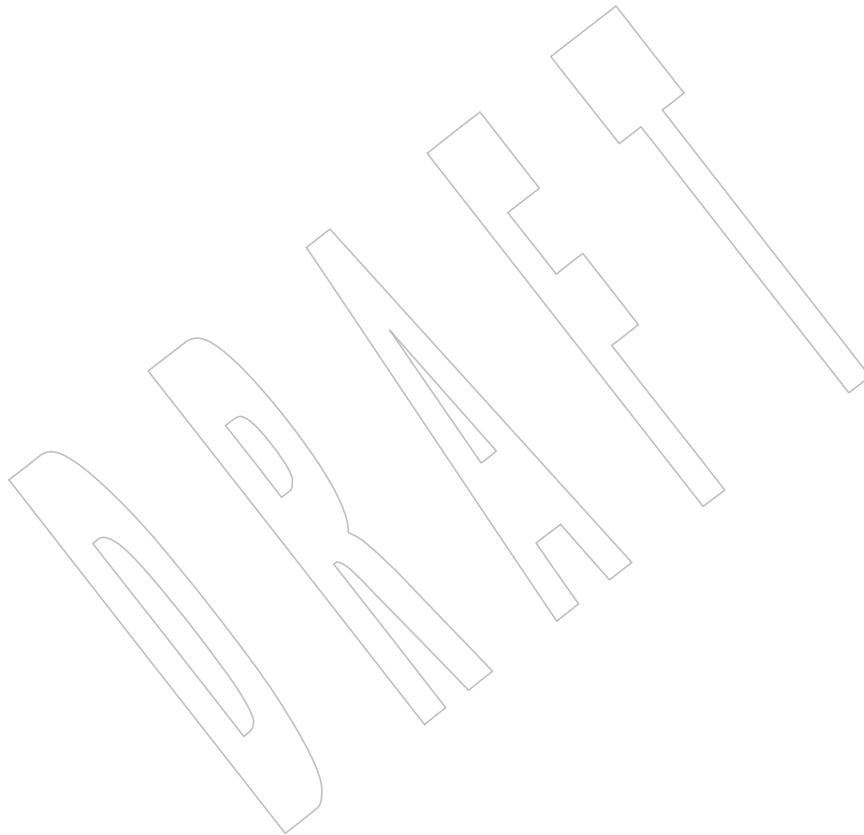
64292

SD5-XSH-ERN-30 is applied.

64293

Austin Group Interpretation 1003.1-2001 #055 is applied, clarifying the thread-safety of this function and treatment of *at_fork()* handlers.

64294



64295 **NAME**
 64296 tan, tanf, tanl — tangent function

64297 **SYNOPSIS**
 64298 #include <math.h>
 64299 double tan(double x);
 64300 float tanf(float x);
 64301 long double tanl(long double x);

64302 DESCRIPTION

64303 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64304 conflict between the requirements described here and the ISO C standard is unintentional. This
 64305 volume of POSIX.1-200x defers to the ISO C standard.

64306 These functions shall compute the tangent of their argument x , measured in radians.

64307 An application wishing to check for error situations should set *errno* to zero and call
 64308 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 64309 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 64310 zero, an error has occurred.

64311 RETURN VALUE

64312 Upon successful completion, these functions shall return the tangent of x .

64313 If the correct value would cause underflow, and is not representable, a range error may occur,
 64314 MX and either 0.0 (if supported), or an implementation-defined value shall be returned.

64315 MX If x is NaN, a NaN shall be returned.

64316 If x is ± 0 , x shall be returned.

64317 If x is subnormal, a range error may occur and x should be returned.

64318 If x is $\pm\text{Inf}$, a domain error shall occur, and either a NaN (if supported), or an implementation-
 64319 defined value shall be returned.

64320 If the correct value would cause underflow, and is representable, a range error may occur and
 64321 the correct value shall be returned.

64322 XSI If the correct value would cause overflow, a range error shall occur and *tan()*, *tanf()*, and *tanl()*
 64323 shall return $\pm\text{HUGE_VAL}$, $\pm\text{HUGE_VALF}$, and $\pm\text{HUGE_VALL}$, respectively, with the same sign
 64324 as the correct value of the function.

64325 ERRORS

64326 These functions shall fail if:

64327 MX **Domain Error** The value of x is $\pm\text{Inf}$.

64328 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64329 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 64330 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 64331 shall be raised.

64332 XSI **Range Error** The result overflows

64333 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64334 then *errno* shall be set to [ERANGE]. If the integer expression
 64335 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 64336 floating-point exception shall be raised.

64337 These functions may fail if:

64338 MX Range Error The result underflows, or the value of x is subnormal.

64339 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
64340 then *errno* shall be set to [ERANGE]. If the integer expression
64341 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
64342 floating-point exception shall be raised.

64343 EXAMPLES

64344 Taking the Tangent of a 45-Degree Angle

```
64345 #include <math.h>
64346 ...
64347 double radians = 45.0 * M_PI / 180;
64348 double result;
64349 ...
64350 result = tan (radians);
```

64351 APPLICATION USAGE

64352 There are no known floating-point representations such that for a normal argument, $\tan(x)$ is
64353 either overflow or underflow.

64354 These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from
64355 0.0.

64356 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
64357 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

64358 RATIONALE

64359 None.

64360 FUTURE DIRECTIONS

64361 None.

64362 SEE ALSO

64363 [atan\(\)](#), [feclearexcept\(\)](#), [fetestexcept\(\)](#), [isnan\(\)](#)

64364 XBD [Section 4.19](#) (on page 104), [<math.h>](#)

64365 CHANGE HISTORY

64366 First released in Issue 1. Derived from Issue 1 of the SVID.

64367 Issue 5

64368 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes
64369 in previous issues.

64370 Issue 6

64371 The *tanf()* and *tanl()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

64372 The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are
64373 revised to align with the ISO/IEC 9899:1999 standard.

64374 IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are
64375 marked.

64376 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/64 is applied, correcting the last
64377 paragraph in the RETURN VALUE section.

64378 **NAME**
 64379 tanh, tanhf, tanhl — hyperbolic tangent functions

64380 **SYNOPSIS**
 64381 #include <math.h>
 64382 double tanh(double x);
 64383 float tanhf(float x);
 64384 long double tanhl(long double x);

64385 **DESCRIPTION**
 64386 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 64387 conflict between the requirements described here and the ISO C standard is unintentional. This
 64388 volume of POSIX.1-200x defers to the ISO C standard.

64389 These functions shall compute the hyperbolic tangent of their argument x .

64390 An application wishing to check for error situations should set *errno* to zero and call
 64391 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 64392 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 64393 zero, an error has occurred.

64394 **RETURN VALUE**
 64395 Upon successful completion, these functions shall return the hyperbolic tangent of x .

64396 MX If x is NaN, a NaN shall be returned.
 64397 If x is ± 0 , x shall be returned.
 64398 If x is $\pm\text{Inf}$, ± 1 shall be returned.
 64399 If x is subnormal, a range error may occur and x should be returned.

64400 **ERRORS**
 64401 These functions may fail if:
 64402 MX **Range Error** The value of x is subnormal.
 64403 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 64404 then *errno* shall be set to [ERANGE]. If the integer expression
 64405 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 64406 floating-point exception shall be raised.

64407 **EXAMPLES**
 64408 None.

64409 **APPLICATION USAGE**
 64410 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 64411 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

64412 **RATIONALE**
 64413 None.

64414 **FUTURE DIRECTIONS**
 64415 None.

64416 **SEE ALSO**
 64417 *atanh()*, *feclearexcept()*, *fetestexcept()*, *isnan()*, *tan()*
 64418 XBD Section 4.19 (on page 104), <math.h>

64419
64420

64421
64422
64423

64424
64425

64426
64427

64428
64429

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

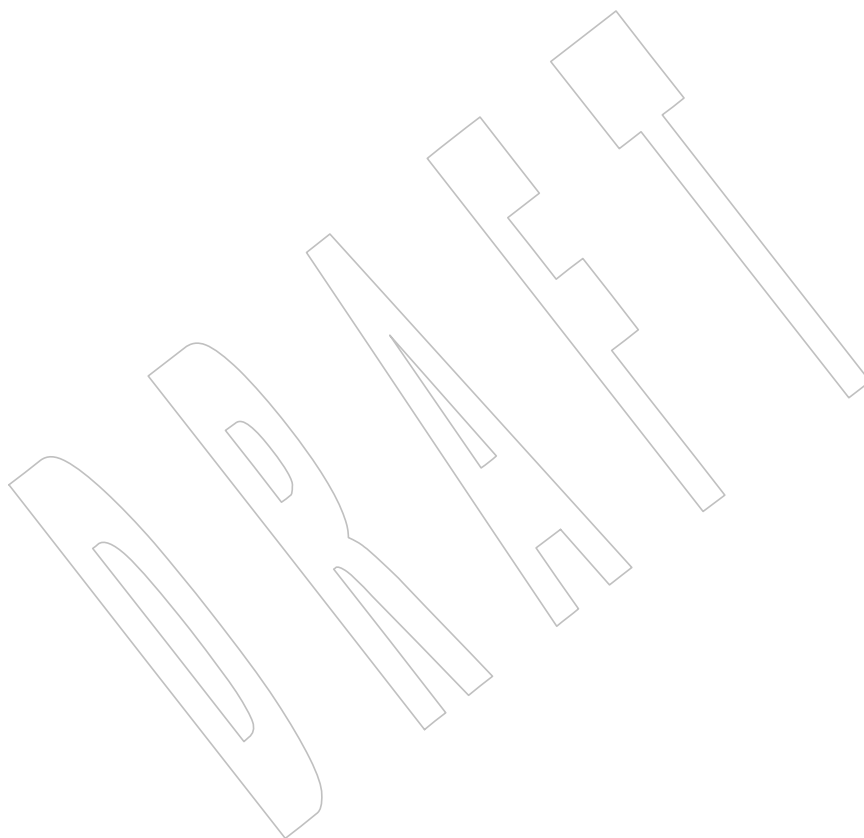
The DESCRIPTION is updated to indicate how an application should check for an error. This text was previously published in the APPLICATION USAGE section.

Issue 6

The *tanhf()* and *tanhll()* functions are added for alignment with the ISO/IEC 9899:1999 standard.

The DESCRIPTION, RETURN VALUE, ERRORS, and APPLICATION USAGE sections are revised to align with the ISO/IEC 9899:1999 standard.

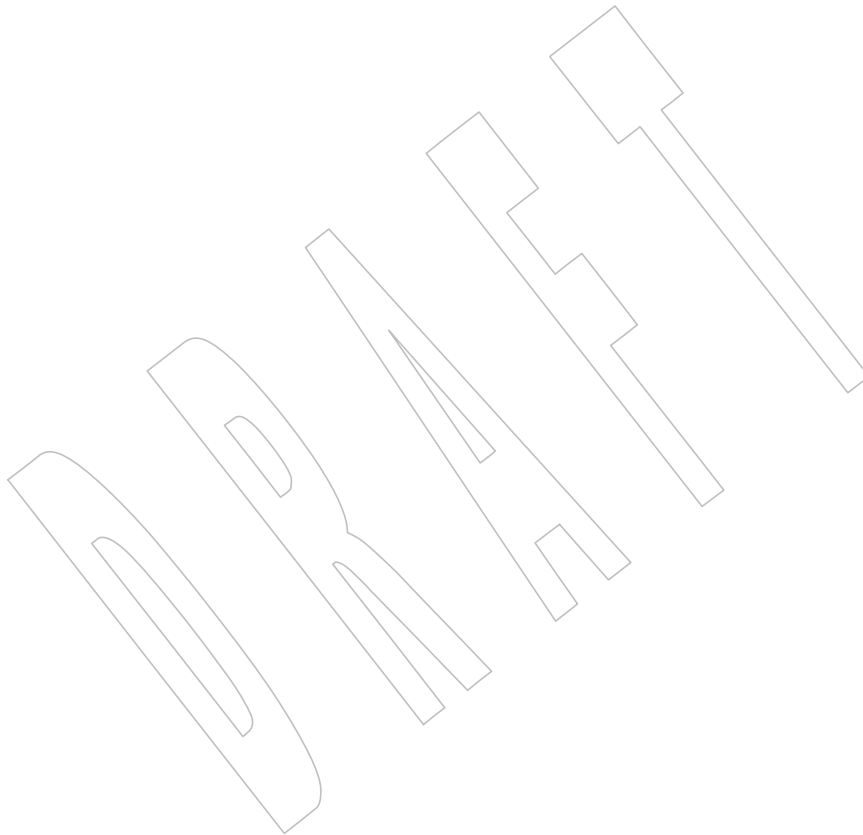
IEC 60559:1989 standard floating-point extensions over the ISO/IEC 9899:1999 standard are marked.



64430 **NAME**
64431 tanl — tangent function

64432 **SYNOPSIS**
64433 #include <math.h>
64434 long double tanl(long double x);

64435 **DESCRIPTION**
64436 Refer to *tan()*.



64437 **NAME**64438 `tcdrain` — wait for transmission of output64439 **SYNOPSIS**64440 `#include <termios.h>`64441 `int tcdrain(int fildev);`64442 **DESCRIPTION**64443 The `tcdrain()` function shall block until all output written to the object referred to by *fildev* is
64444 transmitted. The *fildev* argument is an open file descriptor associated with a terminal.64445 Any attempts to use `tcdrain()` from a process which is a member of a background process group
64446 on a *fildev* associated with its controlling terminal, shall cause the process group to be sent a
64447 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
64448 shall be allowed to perform the operation, and no signal is sent.64449 **RETURN VALUE**64450 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
64451 indicate the error.64452 **ERRORS**64453 The `tcdrain()` function shall fail if:64454 [EBADF] The *fildev* argument is not a valid file descriptor.64455 [EINTR] A signal interrupted `tcdrain()`.64456 [ENOTTY] The file associated with *fildev* is not a terminal.64457 The `tcdrain()` function may fail if:64458 [EIO] The process group of the writing process is orphaned, and the writing process
64459 is not ignoring or blocking SIGTTOU.64460 **EXAMPLES**

64461 None.

64462 **APPLICATION USAGE**

64463 None.

64464 **RATIONALE**

64465 None.

64466 **FUTURE DIRECTIONS**

64467 None.

64468 **SEE ALSO**64469 [*tcflush\(\)*](#)64470 XBD [Chapter 11](#) (on page 185), [<termios.h>](#), [<unistd.h>](#) |64471 **CHANGE HISTORY**

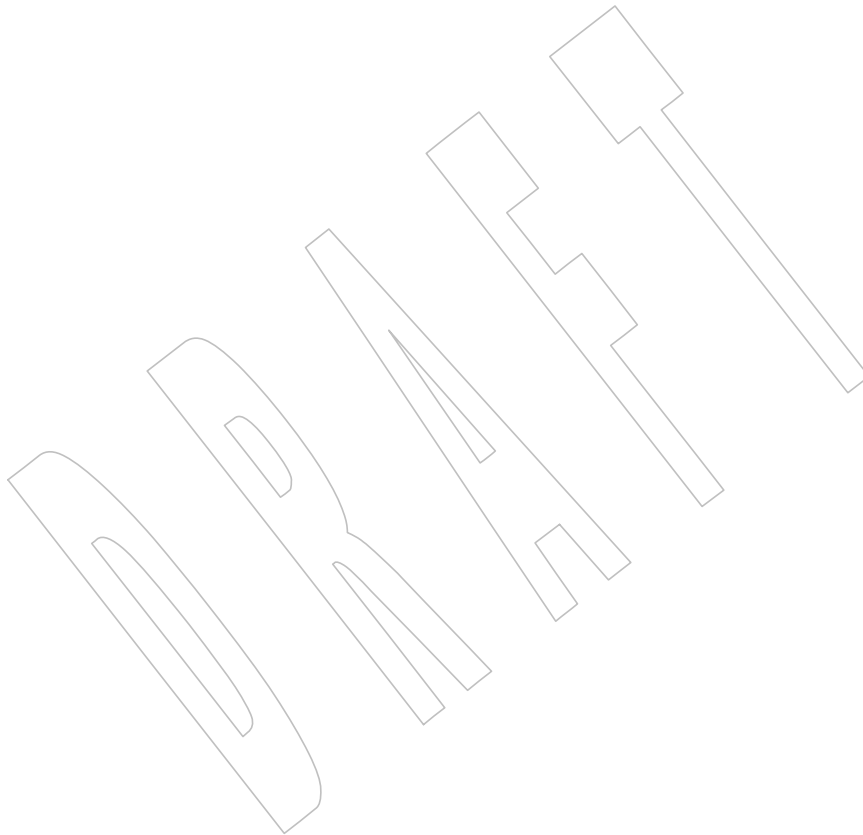
64472 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64473
64474
64475
64476
64477
64478

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the final paragraph is no longer conditional on `_POSIX_JOB_CONTROL`. This is a FIPS requirement.
- The [EIO] error is added.



64479 **NAME**

64480 tcflow — suspend or restart the transmission or reception of data

64481 **SYNOPSIS**

64482 #include <termios.h>

64483 int tcflow(int *fildev*, int *action*);64484 **DESCRIPTION**64485 The *tcflow()* function shall suspend or restart transmission or reception of data on the object
64486 referred to by *fildev*, depending on the value of *action*. The *fildev* argument is an open file
64487 descriptor associated with a terminal.

- 64488 • If *action* is TCOOFF, output shall be suspended.
- 64489 • If *action* is TCOON, suspended output shall be restarted.
- 64490 • If *action* is TCIOFF and *fildev* refers to a terminal device, the system shall transmit a STOP |
64491 character, which is intended to cause the terminal device to stop transmitting data to the |
64492 system. If *fildev* is associated with a pseudo-terminal, the STOP character need not be |
64493 transmitted.
- 64494 • If *action* is TCION and *fildev* refers to a terminal device, the system shall transmit a START |
64495 character, which is intended to cause the terminal device to start transmitting data to the |
64496 system. If *fildev* is associated with a pseudo-terminal, the START character need not be |
64497 transmitted.

64498 The default on the opening of a terminal file is that neither its input nor its output are
64499 suspended.64500 Attempts to use *tcflow()* from a process which is a member of a background process group on a
64501 *fildev* associated with its controlling terminal, shall cause the process group to be sent a
64502 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
64503 shall be allowed to perform the operation, and no signal is sent.64504 **RETURN VALUE**64505 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
64506 indicate the error.64507 **ERRORS**64508 The *tcflow()* function shall fail if:

- 64509 [EBADF] The *fildev* argument is not a valid file descriptor.
- 64510 [EINVAL] The *action* argument is not a supported value.
- 64511 [ENOTTY] The file associated with *fildev* is not a terminal.

64512 The *tcflow()* function may fail if:

- 64513 [EIO] The process group of the writing process is orphaned, and the writing process
64514 is not ignoring or blocking SIGTTOU.

64515
64516
64517
64518
64519
64520
64521
64522
64523
64524
64525
64526
64527
64528
64529
64530
64531
64532
64533
64534

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcsendbreak()

XBD [Chapter 11](#) (on page 185), [<termios.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The [EIO] error is added.

Issue 7

SD5-XSH-ERN-190 is applied, clarifying in the DESCRIPTION the transmission of START and + STOP characters.

64535 **NAME**

64536 tcflush — flush non-transmitted output data, non-read input data, or both

64537 **SYNOPSIS**

64538 #include <termios.h>

64539 int tcflush(int *fildev*, int *queue_selector*);64540 **DESCRIPTION**64541 Upon successful completion, *tcflush()* shall discard data written to the object referred to by *fildev*
64542 (an open file descriptor associated with a terminal) but not transmitted, or data received but not
64543 read, depending on the value of *queue_selector*:

- 64544
- If *queue_selector* is TCIFLUSH, it shall flush data received but not read.
 - If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.
 - If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data
64547 written but not transmitted.

64548 Attempts to use *tcflush()* from a process which is a member of a background process group on a
64549 *fildev* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU
64550 signal. If the calling process is blocking or ignoring SIGTTOU signals, the process shall be
64551 allowed to perform the operation, and no signal is sent.64552 **RETURN VALUE**64553 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
64554 indicate the error.64555 **ERRORS**64556 The *tcflush()* function shall fail if:

- 64557 [EBADF] The
- fildev*
- argument is not a valid file descriptor.
-
- 64558 [EINVAL] The
- queue_selector*
- argument is not a supported value.
-
- 64559 [ENOTTY] The file associated with
- fildev*
- is not a terminal.

64560 The *tcflush()* function may fail if:

- 64561 [EIO] The process group of the writing process is orphaned, and the writing process
-
- 64562 is not ignoring or blocking SIGTTOU.

64563 **EXAMPLES**

64564 None.

64565 **APPLICATION USAGE**

64566 None.

64567 **RATIONALE**

64568 None.

64569 **FUTURE DIRECTIONS**

64570 None.

64571 **SEE ALSO**64572 [tcdrain\(\)](#)64573 XBD Chapter 11 (on page 185), [<termios.h>](#), [<unistd.h>](#)

64574

CHANGE HISTORY

64575

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64576

Issue 6

64577

The Open Group Corrigendum U035/1 is applied. In the ERRORS and APPLICATION USAGE sections, references to *tcflow()* are replaced with *tcflush()*.

64578

64579

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

64580

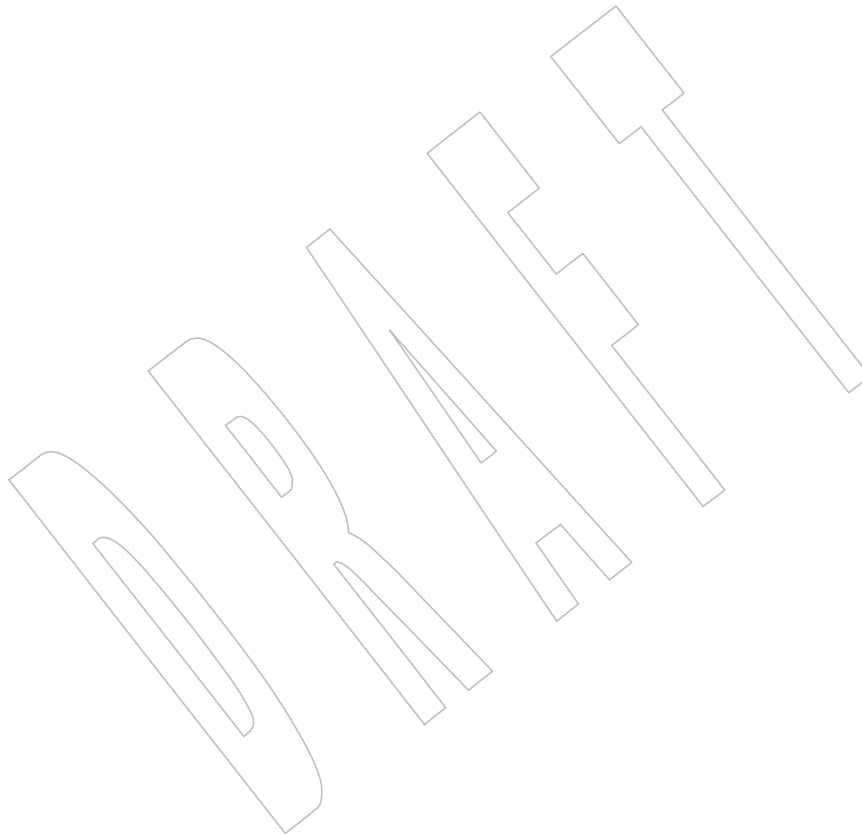
64581

- In the DESCRIPTION, the final paragraph is no longer conditional on `_POSIX_JOB_CONTROL`. This is a FIPS requirement.

64582

- The [EIO] error is added.

64583



64584 **NAME**

64585 tcgetattr — get the parameters associated with the terminal

64586 **SYNOPSIS**

64587 #include <termios.h>

64588 int tcgetattr(int *fildev*, struct termios **termios_p*);64589 **DESCRIPTION**64590 The *tcgetattr()* function shall get the parameters associated with the terminal referred to by *fildev* and store them in the **termios** structure referenced by *termios_p*. The *fildev* argument is an open file descriptor associated with a terminal.64593 The *termios_p* argument is a pointer to a **termios** structure.64594 The *tcgetattr()* operation is allowed from any process.64595 If the terminal device supports different input and output baud rates, the baud rates stored in the **termios** structure returned by *tcgetattr()* shall reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).64600 **RETURN VALUE**64601 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.64603 **ERRORS**64604 The *tcgetattr()* function shall fail if:64605 [EBADF] The *fildev* argument is not a valid file descriptor.64606 [ENOTTY] The file associated with *fildev* is not a terminal.64607 **EXAMPLES**

64608 None.

64609 **APPLICATION USAGE**

64610 None.

64611 **RATIONALE**64612 Care must be taken when changing the terminal attributes. Applications should always do a *tcgetattr()*, save the **termios** structure values returned, and then do a *tcsetattr()*, changing only the necessary fields. The application should use the values saved from the *tcgetattr()* to reset the terminal state whenever it is done with the terminal. This is necessary because terminal attributes apply to the underlying port and not to each individual open instance; that is, all processes that have used the terminal see the latest attribute changes.

64618 A program that uses these functions should be written to catch all signals and take other appropriate actions to ensure that when the program terminates, whether planned or not, the terminal device's state is restored to its original state.

64621 Existing practice dealing with error returns when only part of a request can be honored is based on calls to the *ioctl()* function. In historical BSD and System V implementations, the corresponding *ioctl()* returns zero if the requested actions were semantically correct, even if some of the requested changes could not be made. Many existing applications assume this behavior and would no longer work correctly if the return value were changed from zero to -1 in this case.

64627 Note that either specification has a problem. When zero is returned, it implies everything

64628 succeeded even if some of the changes were not made. When `-1` is returned, it implies
64629 everything failed even though some of the changes were made.

64630 Applications that need all of the requested changes made to work properly should follow
64631 `tcsetattr()` with a call to `tcgetattr()` and compare the appropriate field values.

64632 FUTURE DIRECTIONS

64633 None.

64634 SEE ALSO

64635 [tcsetattr\(\)](#)

64636 XBD [Chapter 11](#) (on page 185), [<termios.h>](#)

64637 CHANGE HISTORY

64638 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64639 Issue 6

64640 In the DESCRIPTION, the rate returned as the input baud rate shall be the output rate.
64641 Previously, the number zero was also allowed but was obsolescent.

DRAFT

64642 **NAME**

64643 tcgetpgrp — get the foreground process group ID

64644 **SYNOPSIS**

64645 #include <unistd.h>

64646 pid_t tcgetpgrp(int *fildev*);64647 **DESCRIPTION**64648 The *tcgetpgrp()* function shall return the value of the process group ID of the foreground process
64649 group associated with the terminal.64650 If there is no foreground process group, *tcgetpgrp()* shall return a value greater than 1 that does
64651 not match the process group ID of any existing process group.64652 The *tcgetpgrp()* function is allowed from a process that is a member of a background process
64653 group; however, the information may be subsequently changed by a process that is a member of
64654 a foreground process group.64655 **RETURN VALUE**64656 Upon successful completion, *tcgetpgrp()* shall return the value of the process group ID of the
64657 foreground process associated with the terminal. Otherwise, `-1` shall be returned and *errno* set to
64658 indicate the error.64659 **ERRORS**64660 The *tcgetpgrp()* function shall fail if:64661 [EBADF] The *fildev* argument is not a valid file descriptor.64662 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
64663 controlling terminal.64664 **EXAMPLES**

64665 None.

64666 **APPLICATION USAGE**

64667 None.

64668 **RATIONALE**

64669 None.

64670 **FUTURE DIRECTIONS**

64671 None.

64672 **SEE ALSO**64673 [setsid\(\)](#), [setpgid\(\)](#), [tcsetpgrp\(\)](#)64674 XBD [<sys/types.h>](#), [<unistd.h>](#)64675 **CHANGE HISTORY**

64676 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

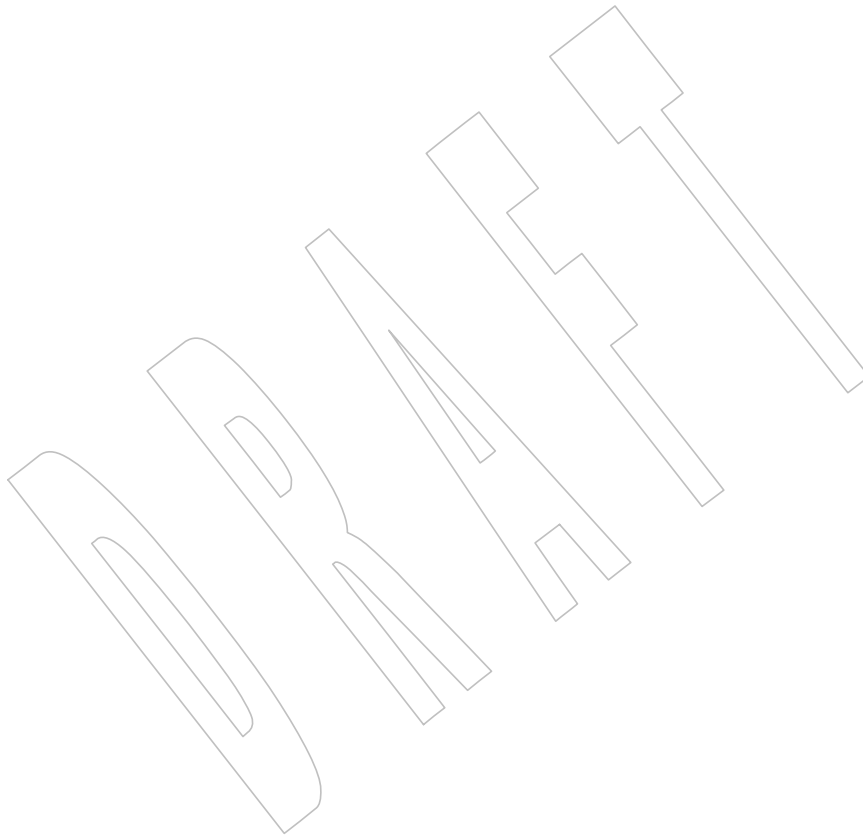
64677 **Issue 6**64678 In the SYNOPSIS, the optional include of the [<sys/types.h>](#) header is removed.64679 The following new requirements on POSIX implementations derive from alignment with the
64680 Single UNIX Specification:

- 64681
- The requirement to include [<sys/types.h>](#) has been removed. Although [<sys/types.h>](#) was
64682 required for conforming implementations of previous POSIX specifications, it was not
64683 required for UNIX applications.

64684

64685

- In the DESCRIPTION, text previously conditional on support for _POSIX_JOB_CONTROL is now mandatory. This is a FIPS requirement.



64686 **NAME**

64687 tcgetsid — get the process group ID for the session leader for the controlling terminal

64688 **SYNOPSIS**

64689 #include <termios.h>

64690 pid_t tcgetsid(int *fildev*);64691 **DESCRIPTION**64692 The *tcgetsid()* function shall obtain the process group ID of the session for which the terminal
64693 specified by *fildev* is the controlling terminal.64694 **RETURN VALUE**64695 Upon successful completion, *tcgetsid()* shall return the process group ID of the session
64696 associated with the terminal. Otherwise, a value of (**pid_t**)-1 shall be returned and *errno* set to
64697 indicate the error.64698 **ERRORS**64699 The *tcgetsid()* function shall fail if:64700 [EBADF] The *fildev* argument is not a valid file descriptor.64701 [ENOTTY] The calling process does not have a controlling terminal, or the file is not the
64702 controlling terminal.64703 **EXAMPLES**

64704 None.

64705 **APPLICATION USAGE**

64706 None.

64707 **RATIONALE**

64708 None.

64709 **FUTURE DIRECTIONS**

64710 None.

64711 **SEE ALSO**64712 XBD <[termios.h](#)>64713 **CHANGE HISTORY**

64714 First released in Issue 4, Version 2.

64715 **Issue 5**

64716 Moved from X/OPEN UNIX extension to BASE.

64717 The [EACCES] error has been removed from the list of mandatory errors, and the description of
64718 [ENOTTY] has been reworded.64719 **Issue 7**

64720 SD5-XSH-ERN-180 is applied, clarifying the RETURN VALUE section. +

64721 The *tcgetsid()* function is moved from the XSI option to the Base.

64722 **NAME**
 64723 tcsendbreak — send a break for a specific duration

64724 **SYNOPSIS**
 64725 #include <termios.h>
 64726 int tcsendbreak(int *fildev*, int *duration*);

64727 **DESCRIPTION**
 64728 If the terminal is using asynchronous serial data transmission, *tcsendbreak()* shall cause
 64729 transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it
 64730 shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5
 64731 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period
 64732 of time.

64733 The *fildev* argument is an open file descriptor associated with a terminal.

64734 If the terminal is not using asynchronous serial data transmission, it is implementation-defined
 64735 whether *tcsendbreak()* sends data to generate a break condition or returns without taking any
 64736 action.

64737 Attempts to use *tcsendbreak()* from a process which is a member of a background process group
 64738 on a *fildev* associated with its controlling terminal shall cause the process group to be sent a
 64739 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
 64740 shall be allowed to perform the operation, and no signal is sent.

64741 **RETURN VALUE**
 64742 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 64743 indicate the error.

64744 **ERRORS**
 64745 The *tcsendbreak()* function shall fail if:
 64746 [EBADF] The *fildev* argument is not a valid file descriptor.
 64747 [ENOTTY] The file associated with *fildev* is not a terminal.
 64748 The *tcsendbreak()* function may fail if:
 64749 [EIO] The process group of the writing process is orphaned, and the writing process
 64750 is not ignoring or blocking SIGTTOU.

64751 **EXAMPLES**
 64752 None.

64753 **APPLICATION USAGE**
 64754 None.

64755 **RATIONALE**
 64756 None.

64757 **FUTURE DIRECTIONS**
 64758 None.

64759 **SEE ALSO**
 64760 XBD Chapter 11 (on page 185), <termios.h>, <unistd.h>

64761

CHANGE HISTORY

64762

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64763

Issue 6

64764

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

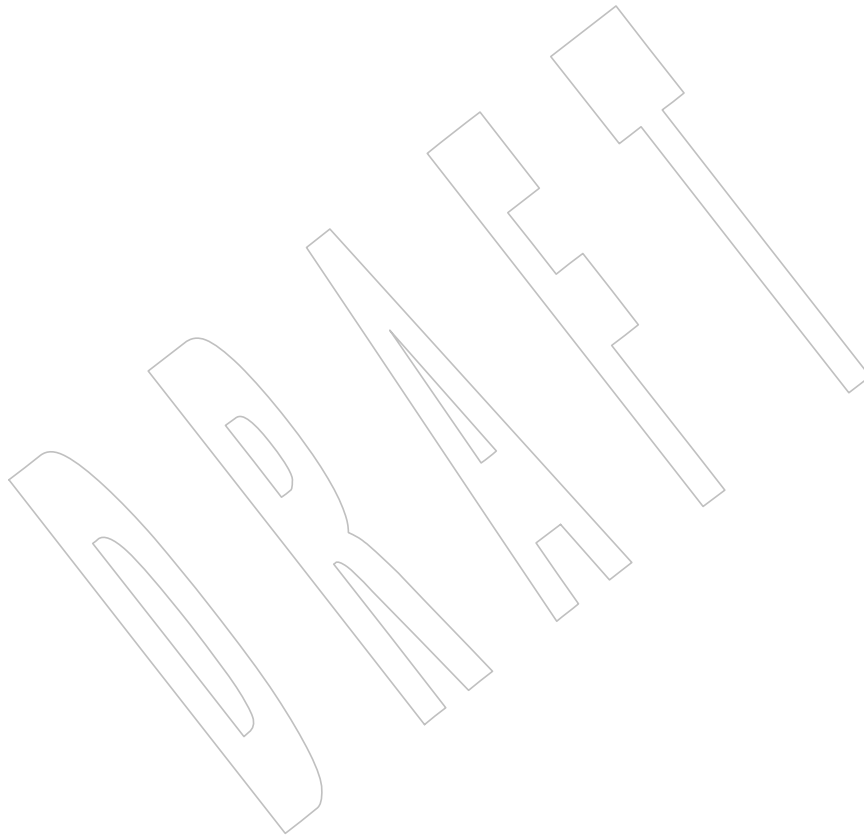
64765

- In the DESCRIPTION, text previously conditional on `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.
- The [EIO] error is added.

64766

64767

64768



64769 **NAME**
 64770 tcsetattr — set the parameters associated with the terminal

64771 **SYNOPSIS**
 64772 #include <termios.h>

64773 int tcsetattr(int *fildev*, int *optional_actions*,
 64774 const struct termios **termios_p*);

64775 **DESCRIPTION**

64776 The *tcsetattr()* function shall set the parameters associated with the terminal referred to by the
 64777 open file descriptor *fildev* (an open file descriptor associated with a terminal) from the **termios**
 64778 structure referenced by *termios_p* as follows:

- 64779 • If *optional_actions* is TCSANOW, the change shall occur immediately.
- 64780 • If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildev* is
 64781 transmitted. This function should be used when changing parameters that affect output.
- 64782 • If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildev* is
 64783 transmitted, and all input so far received but not read shall be discarded before the change
 64784 is made.

64785 If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud
 64786 rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the
 64787 line.

64788 If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud
 64789 rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

64790 The *tcsetattr()* function shall return successfully if it was able to perform any of the requested
 64791 actions, even if some of the requested actions could not be performed. It shall set all the
 64792 attributes that the implementation supports as requested and leave all the attributes not
 64793 supported by the implementation unchanged. If no part of the request can be honored, it shall
 64794 return -1 and set *errno* to [EINVAL]. If the input and output baud rates differ and are a
 64795 combination that is not supported, neither baud rate shall be changed. A subsequent call to
 64796 *tcgetattr()* shall return the actual state of the terminal device (reflecting both the changes made
 64797 and not made in the previous *tcsetattr()* call). The *tcsetattr()* function shall not change the values
 64798 found in the **termios** structure under any circumstances.

64799 The effect of *tcsetattr()* is undefined if the value of the **termios** structure pointed to by *termios_p*
 64800 was not derived from the result of a call to *tcgetattr()* on *fildev*; an application should modify
 64801 only fields and flags defined by this volume of POSIX.1-200x between the call to *tcgetattr()* and
 64802 *tcsetattr()*, leaving all other fields and flags unmodified.

64803 No actions defined by this volume of POSIX.1-200x, other than a call to *tcsetattr()* or a close of
 64804 the last file descriptor in the system associated with this terminal device, shall cause any of the
 64805 terminal attributes defined by this volume of POSIX.1-200x to change.

64806 If *tcsetattr()* is called from a process which is a member of a background process group on a *fildev*
 64807 associated with its controlling terminal:

- 64808 • If the calling process is blocking or ignoring SIGTTOU signals, the operation completes
 64809 normally and no signal is sent.
- 64810 • Otherwise, a SIGTTOU signal shall be sent to the process group.

tcsetattr()64811 **RETURN VALUE**

64812 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 64813 indicate the error.

64814 **ERRORS**

64815 The *tcsetattr()* function shall fail if:

64816 [EBADF] The *fildev* argument is not a valid file descriptor.

64817 [EINTR] A signal interrupted *tcsetattr()*.

64818 [EINVAL] The *optional_actions* argument is not a supported value, or an attempt was
 64819 made to change an attribute represented in the **termios** structure to an
 64820 unsupported value.

64821 [ENOTTY] The file associated with *fildev* is not a terminal.

64822 The *tcsetattr()* function may fail if:

64823 [EIO] The process group of the writing process is orphaned, and the writing process
 64824 is not ignoring or blocking SIGTTOU.

64825 **EXAMPLES**

64826 None.

64827 **APPLICATION USAGE**

64828 If trying to change baud rates, applications should call *tcsetattr()* then call *tcgetattr()* in order to
 64829 determine what baud rates were actually selected.

64830 **RATIONALE**

64831 The *tcsetattr()* function can be interrupted in the following situations:

- 64832 • It is interrupted while waiting for output to drain.
- 64833 • It is called from a process in a background process group and SIGTTOU is caught.

64834 See also the RATIONALE section in *tcgetattr()*.

64835 **FUTURE DIRECTIONS**

64836 Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be
 64837 supported in a future version of this volume of POSIX.1-200x.

64838 **SEE ALSO**

64839 *cfgetispeed()*, *tcgetattr()*

64840 XBD Chapter 11 (on page 185), **<termios.h>**, **<unistd.h>**

64841 **CHANGE HISTORY**

64842 First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

64843 **Issue 6**

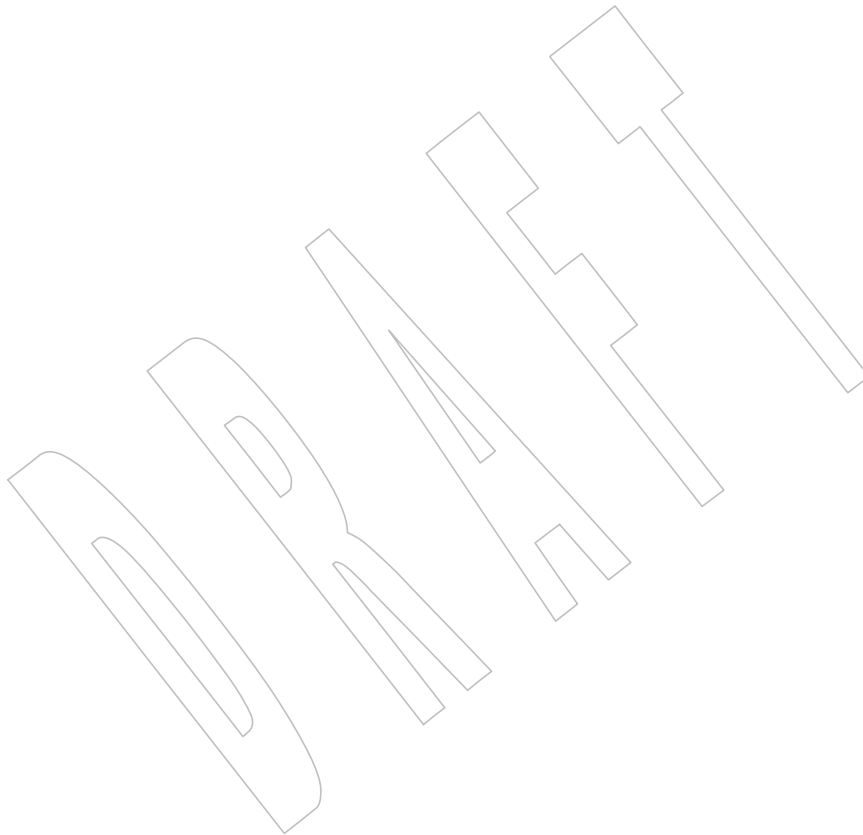
64844 The following new requirements on POSIX implementations derive from alignment with the
 64845 Single UNIX Specification:

- 64846 • In the DESCRIPTION, text previously conditional on **_POSIX_JOB_CONTROL** is now
 64847 mandated. This is a FIPS requirement.
- 64848 • The [EIO] error is added.

64849

64850

In the DESCRIPTION, the text describing use of *tcsetattr()* from a process which is a member of a background process group is clarified.



64851 **NAME**
 64852 `tcsetpgrp` — set the foreground process group ID

64853 **SYNOPSIS**
 64854 `#include <unistd.h>`
 64855 `int tcsetpgrp(int fildev, pid_t pgid_id);`

64856 **DESCRIPTION**
 64857 If the process has a controlling terminal, `tcsetpgrp()` shall set the foreground process group ID
 64858 associated with the terminal to *pgid_id*. The application shall ensure that the file associated with
 64859 *fildev* is the controlling terminal of the calling process and the controlling terminal is currently
 64860 associated with the session of the calling process. The application shall ensure that the value of
 64861 *pgid_id* matches a process group ID of a process in the same session as the calling process.

64862 Attempts to use `tcsetpgrp()` from a process which is a member of a background process group on
 64863 a *fildev* associated with its controlling terminal shall cause the process group to be sent a
 64864 SIGTTOU signal. If the calling process is blocking or ignoring SIGTTOU signals, the process
 64865 shall be allowed to perform the operation, and no signal is sent.

64866 **RETURN VALUE**
 64867 Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to
 64868 indicate the error.

64869 **ERRORS**
 64870 The `tcsetpgrp()` function shall fail if:

64871 [EBADF]	The <i>fildev</i> argument is not a valid file descriptor.
64872 [EINVAL]	This implementation does not support the value in the <i>pgid_id</i> argument.
64873 [ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.
64874 [EPERM]	The value of <i>pgid_id</i> is a value supported by the implementation, but does not match the process group ID of a process in the same session as the calling process.

64879 **EXAMPLES**
 64880 None.

64881 **APPLICATION USAGE**
 64882 None.

64883 **RATIONALE**
 64884 None.

64885 **FUTURE DIRECTIONS**
 64886 None.

64887 **SEE ALSO**
 64888 [tcgetpgrp\(\)](#)
 64889 XBD [<sys/types.h>](#), [<unistd.h>](#)

64890
64891
64892
64893
64894
64895
64896
64897
64898
64899
64900
64901
64902

CHANGE HISTORY

First released in Issue 3. Included for alignment with the POSIX.1-1988 standard.

Issue 6

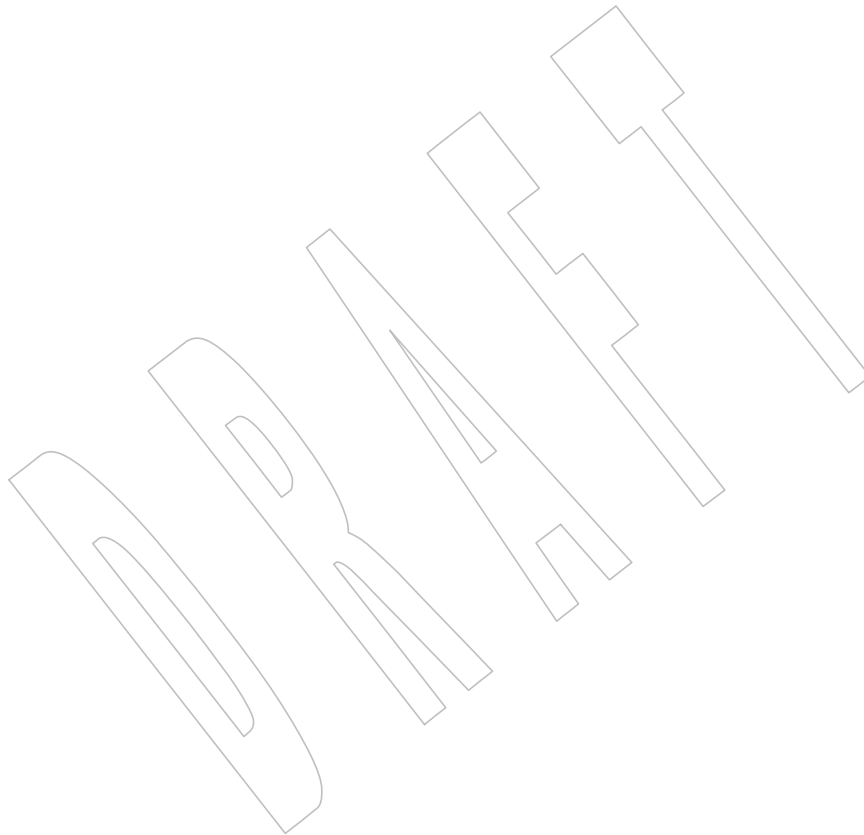
In the SYNOPSIS, the inclusion of `<sys/types.h>` is no longer required.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.
- In the DESCRIPTION and ERRORS sections, text previously conditional on `_POSIX_JOB_CONTROL` is now mandated. This is a FIPS requirement.

The normative text is updated to avoid use of the term “must” for application requirements.

The Open Group Corrigendum U047/4 is applied.



64903 **NAME**64904 `tdelete`, `tfind`, `tsearch`, `twalk` — manage a binary search tree64905 **SYNOPSIS**

```

64906 XSI      #include <search.h>
64907
64907 void *tdelete(const void *restrict key, void **restrict rootp,
64908             int(*compar)(const void *, const void *));
64909 void *tfind(const void *key, void *const *rootp,
64910           int(*compar)(const void *, const void *));
64911 void *tsearch(const void *key, void **rootp,
64912            int (*compar)(const void *, const void *));
64913 void twalk(const void *root,
64914          void (*action)(const void *, VISIT, int));

```

64915 **DESCRIPTION**

64916 The `tdelete()`, `tfind()`, `tsearch()`, and `twalk()` functions manipulate binary search trees.
64917 Comparisons are made with a user-supplied routine, the address of which is passed as the
64918 `compar` argument. This routine is called with two arguments, which are the pointers to the
64919 elements being compared. The application shall ensure that the user-supplied routine returns an
64920 integer less than, equal to, or greater than 0, according to whether the first argument is to be
64921 considered less than, equal to, or greater than the second argument. The comparison function
64922 need not compare every byte, so arbitrary data may be contained in the elements in addition to
64923 the values being compared.

64924 The `tsearch()` function shall build and access the tree. The `key` argument is a pointer to an element
64925 to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed
64926 to by `key`, a pointer to this found node shall be returned. Otherwise, the value pointed to by `key`
64927 shall be inserted (that is, a new node is created and the value of `key` is copied to this node), and a
64928 pointer to this node returned. Only pointers are copied, so the application shall ensure that the
64929 calling routine stores the data. The `rootp` argument points to a variable that points to the root
64930 node of the tree. A null pointer value for the variable pointed to by `rootp` denotes an empty tree;
64931 in this case, the variable shall be set to point to the node which shall be at the root of the new
64932 tree.

64933 Like `tsearch()`, `tfind()` shall search for a node in the tree, returning a pointer to it if found.
64934 However, if it is not found, `tfind()` shall return a null pointer. The arguments for `tfind()` are the
64935 same as for `tsearch()`.

64936 The `tdelete()` function shall delete a node from a binary search tree. The arguments are the same
64937 as for `tsearch()`. The variable pointed to by `rootp` shall be changed if the deleted node was the
64938 root of the tree. The `tdelete()` function shall return a pointer to the parent of the deleted node, or
64939 a null pointer if the node is not found.

64940 The `twalk()` function shall traverse a binary search tree. The `root` argument is a pointer to the root
64941 node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below
64942 that node.) The argument `action` is the name of a routine to be invoked at each node. This routine
64943 is, in turn, called with three arguments. The first argument shall be the address of the node being
64944 visited. The structure pointed to by this argument is unspecified and shall not be modified by
64945 the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-
64946 element to access the element stored in the node. The second argument shall be a value from an
64947 enumeration data type:

```

64948 typedef enum { preorder, postorder, endorder, leaf } VISIT;

```

64949 (defined in `<search.h>`), depending on whether this is the first, second, or third time that the
 64950 node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a
 64951 leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

64952 If the calling function alters the pointer to the root, the result is undefined.

64953 RETURN VALUE

64954 If the node is found, both `tsearch()` and `tfind()` shall return a pointer to it. If not, `tfind()` shall
 64955 return a null pointer, and `tsearch()` shall return a pointer to the inserted item.

64956 A null pointer shall be returned by `tsearch()` if there is not enough space available to create a new
 64957 node.

64958 A null pointer shall be returned by `tdelete()`, `tfind()`, and `tsearch()` if `rootp` is a null pointer on
 64959 entry.

64960 The `tdelete()` function shall return a pointer to the parent of the deleted node, or a null pointer if
 64961 the node is not found.

64962 The `twalk()` function shall not return a value.

64963 ERRORS

64964 No errors are defined.

64965 EXAMPLES

64966 The following code reads in strings and stores structures containing a pointer to each string and
 64967 a count of its length. It then walks the tree, printing out the stored strings and their lengths in
 64968 alphabetical order.

```

64969 #include <search.h>
64970 #include <string.h>
64971 #include <stdio.h>
64972 #define STRSZ 10000
64973 #define NODSZ 500
64974 struct node { /* Pointers to these are stored in the tree. */
64975     char *string;
64976     int length;
64977 };
64978 char string_space[STRSZ]; /* Space to store strings. */
64979 struct node nodes[NODSZ]; /* Nodes to store. */
64980 void *root = NULL; /* This points to the root. */
64981 int main(int argc, char *argv[])
64982 {
64983     char *strptr = string_space;
64984     struct node *nodeptr = nodes;
64985     void print_node(const void *, VISIT, int);
64986     int i = 0, node_compare(const void *, const void *);
64987     while (gets(strptr) != NULL && i++ < NODSZ) {
64988         /* Set node. */
64989         nodeptr->string = strptr;
64990         nodeptr->length = strlen(strptr);
64991         /* Put node into the tree. */
64992         (void) tsearch((void *)nodeptr, (void **)&root,
64993             node_compare);
64994         /* Adjust pointers, so we do not overwrite tree. */
64995         strptr += nodeptr->length + 1;

```

```

64996         nodeptr++;
64997     }
64998     twalk(root, print_node);
64999     return 0;
65000 }
65001 /*
65002  * This routine compares two nodes, based on an
65003  * alphabetical ordering of the string field.
65004  */
65005 int
65006 node_compare(const void *node1, const void *node2)
65007 {
65008     return strcmp(((const struct node *) node1)->string,
65009                 ((const struct node *) node2)->string);
65010 }
65011 /*
65012  * This routine prints out a node, the second time
65013  * twalk encounters it or if it is a leaf.
65014  */
65015 void
65016 print_node(const void *ptr, VISIT order, int level)
65017 {
65018     const struct node *p = *(const struct node **) ptr;
65019     if (order == postorder || order == leaf) {
65020         (void) printf("string = %s, length = %d\n",
65021                     p->string, p->length);
65022     }
65023 }

```

APPLICATION USAGE

The *root* argument to *twalk()* is one level of indirection less than the *rootp* arguments to *tdelete()* and *tsearch()*.

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *tsearch()* function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which could result in some confusion over the meaning of **postorder**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate(), *lsearch()*

XBD <[search.h](#)>

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

65041

Issue 5

65042

The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in previous issues.

65043

65044

Issue 6

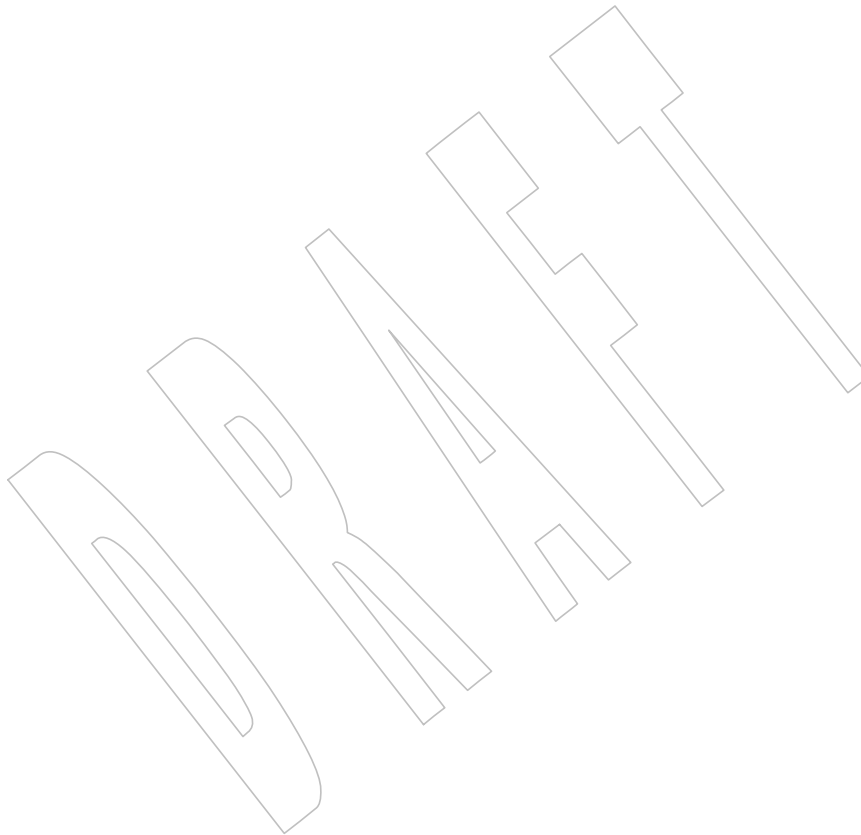
65045

The normative text is updated to avoid use of the term “must” for application requirements.

65046

The **restrict** keyword is added to the *tdelete()* prototype for alignment with the ISO/IEC 9899:1999 standard.

65047



65048 **NAME**
 65049 tellmdir — current location of a named directory stream

65050 **SYNOPSIS**

65051 XSI #include <dirent.h>
 65052 long tellmdir(DIR *dirp);

65053 **DESCRIPTION**

65054 The *tellmdir()* function shall obtain the current location associated with the directory stream
 65055 specified by *dirp*.

65056 If the most recent operation on the directory stream was a *seekdir()*, the directory position
 65057 returned from the *tellmdir()* shall be the same as that supplied as a *loc* argument for *seekdir()*.

65058 **RETURN VALUE**

65059 Upon successful completion, *tellmdir()* shall return the current location of the specified directory
 65060 stream.

65061 **ERRORS**

65062 No errors are defined.

65063 **EXAMPLES**

65064 None.

65065 **APPLICATION USAGE**

65066 None.

65067 **RATIONALE**

65068 None.

65069 **FUTURE DIRECTIONS**

65070 None.

65071 **SEE ALSO**

65072 *fdopendir()*, *readdir()*, *seekdir()*

65073 XBD <dirent.h>

65074 **CHANGE HISTORY**

65075 First released in Issue 2.

65076 **NAME**

65077 tempnam — create a name for a temporary file

65078 **SYNOPSIS**

```
65079 OB XSI #include <stdio.h>
65080 char *tempnam(const char *dir, const char *pfx);
```

65081 **DESCRIPTION**65082 The *tempnam()* function shall generate a pathname that may be used for a temporary file.

65083 The *tempnam()* function allows the user to control the choice of a directory. The *dir* argument
 65084 points to the name of the directory in which the file is to be created. If *dir* is a null pointer or
 65085 points to a string which is not a name for an appropriate directory, the path prefix defined as
 65086 P_tmpdir in the <stdio.h> header shall be used. If that directory is not accessible, an
 65087 implementation-defined directory may be used.

65088 Many applications prefer their temporary files to have certain initial letter sequences in their
 65089 names. The *pfx* argument should be used for this. This argument may be a null pointer or point
 65090 to a string of up to five bytes to be used as the beginning of the filename.

65091 Some implementations of *tempnam()* may use *tmpnam()* internally. On such implementations, if
 65092 called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

65093 **RETURN VALUE**

65094 Upon successful completion, *tempnam()* shall allocate space for a string, put the generated
 65095 pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a
 65096 subsequent call to *free()*. Otherwise, it shall return a null pointer and set *errno* to indicate the
 65097 error.

65098 **ERRORS**65099 The *tempnam()* function shall fail if:

65100 [ENOMEM] Insufficient storage space is available.

65101 **EXAMPLES**65102 **Generating a Pathname**

65103 The following example generates a pathname for a temporary file in directory */tmp*, with the
 65104 prefix *file*. After the filename has been created, the call to *free()* deallocates the space used to
 65105 store the filename.

```
65106 #include <stdio.h>
65107 #include <stdlib.h>
65108 ...
65109 char *directory = "/tmp";
65110 char *fileprefix = "file";
65111 char *file;
65112 file = tempnam(directory, fileprefix);
65113 free(file);
```

65114 **APPLICATION USAGE**

65115 This function only creates pathnames. It is the application's responsibility to create and remove
 65116 the files. Between the time a pathname is created and the file is opened, it is possible for some
 65117 other process to create a file with the same name. Applications may find *tmpfile()* more useful.

tempnam()

65118 Applications should use the *tmpfile()*, *mkdtemp()*, or *mkstemp()* functions instead of the
 65119 obsolescent *tempnam()* function.

RATIONALE

65121 None.

FUTURE DIRECTIONS

65122 The *tempnam()* function may be removed in a future version.
 65123

SEE ALSO

65124 *fopen()*, *free()*, *open()*, *tmpfile()*, *tmpnam()*, *unlink*

65126 XBD <stdio.h>

CHANGE HISTORY

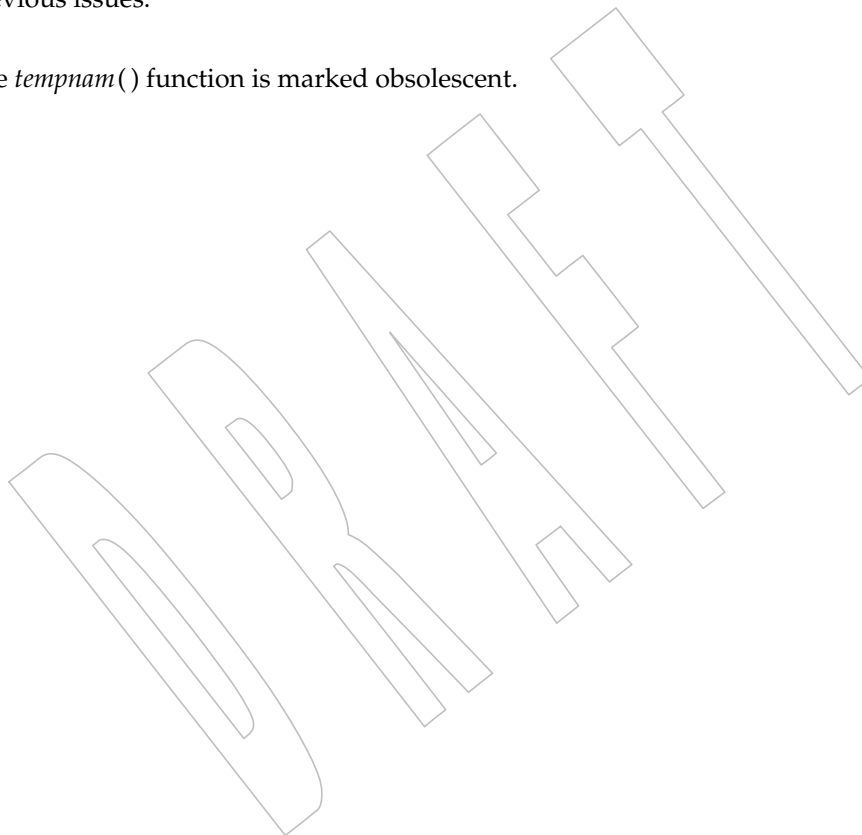
65127 First released in Issue 1. Derived from Issue 1 of the SVID.
 65128

Issue 5

65129 The last paragraph of the DESCRIPTION was included as an APPLICATION USAGE note in
 65130 previous issues.
 65131

Issue 7

65132 The *tempnam()* function is marked obsolescent.
 65133



65134 **NAME**
65135 `tfind` — search binary search tree

65136 **SYNOPSIS**

```
65137 XSI #include <search.h>  
65138 void *tfind(const void *key, void *const *rootp,  
65139 int (*compar)(const void *, const void *));
```

65140 **DESCRIPTION**

65141 Refer to [tdelete\(\)](#).

65142 **NAME**

65143 tgamma, tgammaf, tgammal — compute gamma() function

65144 **SYNOPSIS**

```
65145 #include <math.h>
65146
65146 double tgamma(double x);
65147 float tgammaf(float x);
65148 long double tgammal(long double x);
```

65149 **DESCRIPTION**

65150 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 65151 conflict between the requirements described here and the ISO C standard is unintentional. This
 65152 volume of POSIX.1-200x defers to the ISO C standard.

65153 These functions shall compute the *gamma()* function of *x*.

65154 An application wishing to check for error situations should set *errno* to zero and call
 65155 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 65156 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 65157 zero, an error has occurred.

65158 **RETURN VALUE**

65159 Upon successful completion, these functions shall return *Gamma(x)*.

65160 CX If *x* is a negative integer, a **domain** error may occur and either a NaN (if supported) or an
 65161 MX implementation-defined value shall be returned. On systems that support the IEC 60559
 65162 Floating-Point option, a domain error shall occur and a NaN shall be returned.

65163 If *x* is ± 0 , *tgamma()*, *tgammaf()*, and *tgammal()* shall return \pm HUGE_VAL, \pm HUGE_VALF, and
 65164 MX \pm HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, a
 65165 pole error shall occur;

65166 CX otherwise, a **pole** error may occur.

65167 If the correct value would cause overflow, a range error shall occur and *tgamma()*, *tgammaf()*,
 65168 and *tgammal()* shall return \pm HUGE_VAL, \pm HUGE_VALF, or \pm HUGE_VALL, respectively, with
 65169 the same sign as the correct value of the function.

65170 MX If *x* is NaN, a NaN shall be returned.

65171 If *x* is +Inf, *x* shall be returned.

65172 If *x* is -Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-
 65173 defined value shall be returned.

65174 **ERRORS**

65175 These functions shall fail if:

65176 MX **Domain Error** The value of *x* is a negative integer, or *x* is -Inf.
 65177 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65178 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 65179 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 65180 shall be raised.

65181 MX **Pole Error** The value of *x* is zero.
 65182 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 65183 then *errno* shall be set to [ERANGE]. If the integer expression
 65184 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero

65185		floating-point exception shall be raised.
65186	Range Error	The value overflows.
65187		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
65188		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
65189		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the overflow
65190		floating-point exception shall be raised.
65191		These functions may fail if:
65192	Domain Error	The value of <i>x</i> is a negative integer.
65193		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
65194		then <i>errno</i> shall be set to [EDOM]. If the integer expression (<i>math_errhandling</i>
65195		& MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
65196		shall be raised.
65197	Pole Error	The value of <i>x</i> is zero.
65198		If the integer expression (<i>math_errhandling</i> & MATH_ERRNO) is non-zero,
65199		then <i>errno</i> shall be set to [ERANGE]. If the integer expression
65200		(<i>math_errhandling</i> & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
65201		floating-point exception shall be raised.

EXAMPLES

None.

APPLICATION USAGE

For IEEE Std 754-1985 **double**, overflow happens when $0 < x < 1/\text{DBL_MAX}$, and $171.7 < x$.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

This function is named *tgamma()* in order to avoid conflicts with the historical *gamma()* and *lgamma()* functions.

FUTURE DIRECTIONS

It is possible that the error response for a negative integer argument may be changed to a pole error and a return value of $\pm\text{Inf}$.

SEE ALSO

[*feclearexcept\(\)*](#), [*fetestexcept\(\)*](#), [*lgamma\(\)*](#)

XBD Section 4.19 (on page 104), [**<math.h>**](#)

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/65 is applied, correcting the third paragraph in the RETURN VALUE section.

Issue 7

ISO/IEC 9899:1999 standard, Technical Corrigendum 2 #52 (SD5-XSH-ERN-85) is applied.

65223 **NAME**
 65224 `time` — get time

65225 **SYNOPSIS**
 65226 `#include <time.h>`
 65227 `time_t time(time_t *tloc);`

65228 **DESCRIPTION**
 65229 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 65230 conflict between the requirements described here and the ISO C standard is unintentional. This
 65231 volume of POSIX.1-200x defers to the ISO C standard.

65232 CX The `time()` function shall return the value of time in seconds since the Epoch.
 65233 The `tloc` argument points to an area where the return value is also stored. If `tloc` is a null pointer,
 65234 no value is stored.

65235 **RETURN VALUE**
 65236 Upon successful completion, `time()` shall return the value of time. Otherwise, `(time_t)-1` shall be
 65237 returned.

65238 **ERRORS**
 65239 No errors are defined.

65240 **EXAMPLES**

65241 **Getting the Current Time**

65242 The following example uses the `time()` function to calculate the time elapsed, in seconds, since
 65243 the Epoch, `localtime()` to convert that value to a broken-down time, and `asctime()` to convert the
 65244 broken-down time values into a printable string.

```
65245 #include <stdio.h>
65246 #include <time.h>
65247
65248 int main(void)
65249 {
65250     time_t result;
65251
65252     result = time(NULL);
65253     printf("%s%ju secs since the Epoch\n",
65254           asctime(localtime(&result)),
65255           (uintmax_t)result);
65256     return(0);
65257 }
```

65256 This example writes the current time to `stdout` in a form like this:

```
65257 Wed Jun 26 10:32:15 1996
65258 835810335 secs since the Epoch
```



```

65259     Timing an Event
65260     The following example gets the current time, prints it out in the user's format, and prints the
65261     number of minutes to an event being timed.
65262     #include <time.h>
65263     #include <stdio.h>
65264     ...
65265     time_t now;
65266     int minutes_to_event;
65267     ...
65268     time(&now);
65269     minutes_to_event = ...;
65270     printf("The time is ");
65271     puts(asctime(localtime(&now)));
65272     printf("There are %d minutes to the event.\n",
65273           minutes_to_event);
65274     ...

```

APPLICATION USAGE

None.

RATIONALE

The *time()* function returns a value in seconds (type **time_t**) while *times()* returns a set of values in clock ticks (type **clock_t**). Some historical implementations, such as 4.3 BSD, have mechanisms capable of returning more precise times (see below). A generalized timing scheme to unify these various timing mechanisms has been proposed but not adopted.

Implementations in which **time_t** is a 32-bit signed integer (many historical implementations) fail in the year 2038. POSIX.1-200x does not address this problem. However, the use of the **time_t** type is mandated in order to ease the eventual fix.

The use of the **<time.h>** header instead of **<sys/types.h>** allows compatibility with the ISO C standard.

Many historical implementations (including Version 7) and the 1984 /usr/group standard use **long** instead of **time_t**. This volume of POSIX.1-200x uses the latter type in order to agree with the ISO C standard.

4.3 BSD includes *time()* only as an alternate function to the more flexible *gettimeofday()* function.

FUTURE DIRECTIONS

In a future version of this volume of POSIX.1-200x, **time_t** is likely to be required to be capable of representing times far in the future. Whether this will be mandated as a 64-bit type or a requirement that a specific date in the future be representable (for example, 10000 AD) is not yet determined. Systems purchased after the approval of this volume of POSIX.1-200x should be evaluated to determine whether their lifetime will extend past 2038.

SEE ALSO

asctime(), *clock()*, *ctime()*, *difftime()*, *gettimeofday()*, *gmtime()*, *localtime()*, *mktime()*, *strftime()*, *strptime()*, *utime()*

XBD **<time.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

65303

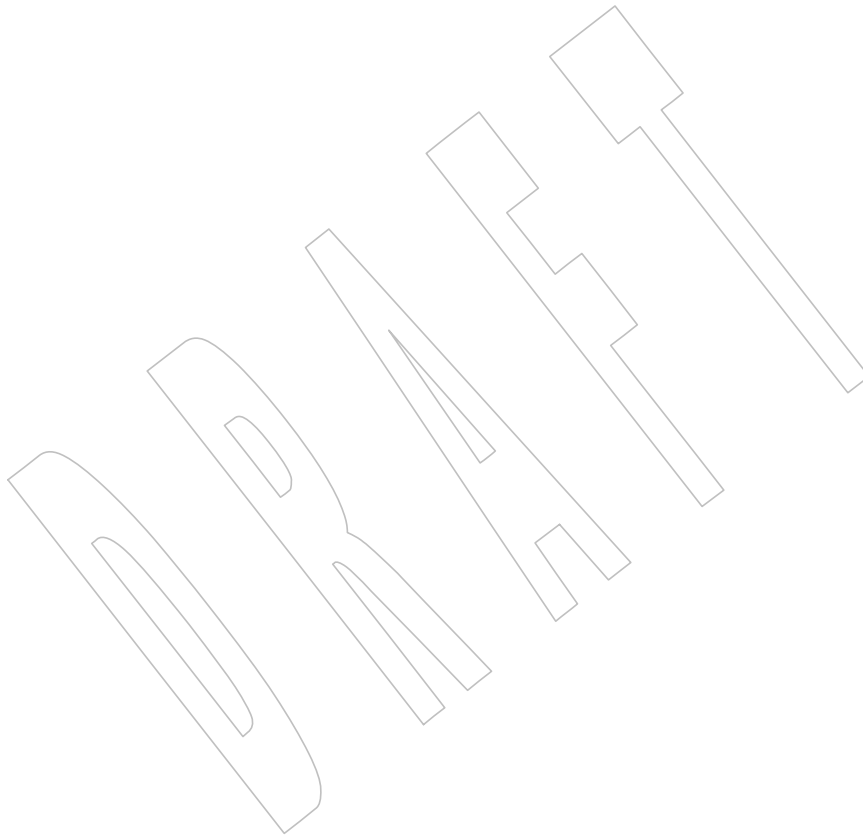
Issue 6

65304

Extensions beyond the ISO C standard are marked.

65305

The EXAMPLES, RATIONALE, and FUTURE DIRECTIONS sections are added.



65306 **NAME**

65307 timer_create — create a per-process timer

65308 **SYNOPSIS**

```
65309 CX      #include <signal.h>
65310      #include <time.h>
65311
65311      int timer_create(clockid_t clockid, struct sigevent *restrict evp,
65312                      timer_t *restrict timerid);
```

65313 **DESCRIPTION**

65314 The *timer_create()* function shall create a per-process timer using the specified clock, *clock_id*, as
 65315 the timing base. The *timer_create()* function shall return, in the location referenced by *timerid*, a
 65316 timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be
 65317 unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is
 65318 defined in **<time.h>**. The timer whose ID is returned shall be in a disarmed state upon return
 65319 from *timer_create()*.

65320 The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the
 65321 application, defines the asynchronous notification to occur as specified in [Section 2.4.1](#) (on page
 65322 463) when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument
 65323 pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL,
 65324 the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of
 65325 the timer ID.

65326 Each implementation shall define a set of clocks that can be used as timing bases for per-process
 65327 timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic
 65328 Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

65329 Per-process timers shall not be inherited by a child process across a *fork()* and shall be disarmed
 65330 and deleted by an *exec*.

65331 CPT If **_POSIX_CPUTIME** is defined, implementations shall support *clock_id* values representing the
 65332 CPU-time clock of the calling process.

65333 TCT If **_POSIX_THREAD_CPUTIME** is defined, implementations shall support *clock_id* values
 65334 representing the CPU-time clock of the calling thread.

65335 CPT|TCT It is implementation-defined whether a *timer_create()* function will succeed if the value defined
 65336 by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process
 65337 or thread invoking the function.

65338 TSA If *evp->sigev_notify* is SIGEV_THREAD and *sev->sigev_notify_attributes* is not NULL, if the
 65339 attribute pointed to by *sev->sigev_notify_attributes* has a thread stack address specified by a call
 65340 to *pthread_attr_setstack()*, the results are unspecified if the signal is generated more than once.

65341 **RETURN VALUE**

65342 If the call succeeds, *timer_create()* shall return zero and update the location referenced by *timerid*
 65343 to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function
 65344 shall return a value of -1 and set *errno* to indicate the error. The value of *timerid* is undefined if
 65345 an error occurs.

65346 **ERRORS**65347 The *timer_create()* function shall fail if:

timer_create()

System Interfaces

65348	[EAGAIN]	The system lacks sufficient signal queuing resources to honor the request.
65349	[EAGAIN]	The calling process has already created all of the timers it is allowed by this implementation.
65350		
65351	[EINVAL]	The specified clock ID is not defined.
65352	CPT TCT [ENOTSUP]	The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by <i>clock_id</i> and associated with a process or thread different from the process or thread invoking <i>timer_create()</i> .
65353		
65354		

EXAMPLES

None.

APPLICATION USAGE

If a timer is created which has *evp->sigev_sigev_notify* set to `SIGEV_THREAD` and the attribute pointed to by *evp->sigev_notify_attributes* has a thread stack address specified by a call to *pthread_attr_setstack()*, the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in response to a timer expiration are created detached, or in an unspecified way if the thread attribute's *detachstate* is `PTHREAD_CREATE_JOINABLE`. In neither case is it valid to call *pthread_join()*, which makes it impossible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

RATIONALE**Periodic Timer Overrun and Resource Allocation**

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an [EAGAIN] error.

Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may “overrun”; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one “signal queuing resource” that can be allocated at the time of the *timer_create()* function call.

A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space, associated with the *timerid*. The *timer_getoverrun()* function can then be implemented as a macro

65396 that uses the *timerid* argument (which may just be a pointer to a user space structure containing
 65397 the counter) to locate the overrun count with no system call overhead. Other implementations,
 65398 less concerned with this class of applications, can avoid the asynchronous update of user space
 65399 by maintaining the count in a system structure at the cost of the extra system call to obtain it.

65400 **Timer Expiration Signal Parameters**

65401 The Realtime Signals Extension option supports an application-specific datum that is delivered
 65402 to the extended signal handler. This value is explicitly specified by the application, along with
 65403 the signal number to be delivered, in a **sigevent** structure. The type of the application-defined
 65404 value can be either an integer constant or a pointer. This explicit specification of the value, as
 65405 opposed to always sending the timer ID, was selected based on existing practice.

65406 It is common practice for realtime applications (on non-POSIX systems or realtime extended
 65407 POSIX systems) to use the parameters of event handlers as the case label of a switch statement or
 65408 as a pointer to an application-defined data structure. Since *timer_ids* are dynamically allocated
 65409 by the *timer_create()* function, they can be used for neither of these functions without additional
 65410 application overhead in the signal handler; for example, to search an array of saved timer IDs to
 65411 associate the ID with a constant or application data structure.

65412 **FUTURE DIRECTIONS**

65413 None.

65414 **SEE ALSO**

65415 [*clock_getres\(\)*](#), [*timer_delete\(\)*](#), [*timer_getoverrun\(\)*](#)

65416 XBD [**<time.h>**](#)

65417 **CHANGE HISTORY**

65418 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

65419 **Issue 6**

65420 The *timer_create()* function is marked as part of the Timers option.

65421 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 65422 implementation does not support the Timers option.

65423 CPU-time clocks are added for alignment with IEEE Std 1003.1d-1999.

65424 The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by adding the
 65425 requirement for the CLOCK_MONOTONIC clock under the Monotonic Clock option.

65426 The **restrict** keyword is added to the *timer_create()* prototype for alignment with the
 65427 ISO/IEC 9899:1999 standard.

65428 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/138 is applied, updating the
 65429 DESCRIPTION and APPLICATION USAGE sections to describe the case when a timer is created
 65430 with the notification method set to SIGEV_THREAD.

65431 **Issue 7**

65432 The *timer_create()* function is moved from the Timers option to the Base.

timer_delete()65433 **NAME**

65434 timer_delete — delete a per-process timer

65435 **SYNOPSIS**

```
65436 CX #include <time.h>
65437 int timer_delete(timer_t timerid);
```

65438 **DESCRIPTION**

65439 The *timer_delete()* function deletes the specified timer, *timerid*, previously created by the
 65440 *timer_create()* function. If the timer is armed when *timer_delete()* is called, the behavior shall be
 65441 as if the timer is automatically disarmed before removal. The disposition of pending signals for
 65442 the deleted timer is unspecified.

65443 **RETURN VALUE**

65444 If successful, the *timer_delete()* function shall return a value of zero. Otherwise, the function shall
 65445 return a value of -1 and set *errno* to indicate the error.

65446 **ERRORS**65447 The *timer_delete()* function may fail if:65448 [EINVAL] The timer ID specified by *timerid* is not a valid timer ID.65449 **EXAMPLES**

65450 None.

65451 **APPLICATION USAGE**

65452 None.

65453 **RATIONALE**

65454 None.

65455 **FUTURE DIRECTIONS**

65456 None.

65457 **SEE ALSO**65458 [timer_create\(\)](#)65459 XBD [<time.h>](#)65460 **CHANGE HISTORY**

65461 First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

65462 **Issue 6**65463 The *timer_delete()* function is marked as part of the Timers option.

65464 The [ENOSYS] error condition has been removed as stubs need not be provided if an
 65465 implementation does not support the Timers option.

65466 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/139 is applied, updating the ERRORS
 65467 section so that the [EINVAL] error becomes optional.

65468 **Issue 7**65469 The *timer_delete()* function is moved from the Timers option to the Base.

65470 **NAME**

65471 timer_getoverrun, timer_gettime, timer_settime — per-process timers

65472 **SYNOPSIS**

```
65473 CX #include <time.h>
65474
65474 int timer_getoverrun(timer_t timerid);
65475 int timer_gettime(timer_t timerid, struct itimerspec *value);
65476 int timer_settime(timer_t timerid, int flags,
65477     const struct itimerspec *restrict value,
65478     struct itimerspec *restrict ovalue);
```

65479 **DESCRIPTION**

65480 The *timer_gettime()* function shall store the amount of time until the specified timer, *timerid*,
 65481 expires and the reload value of the timer into the space pointed to by the *value* argument. The
 65482 *it_value* member of this structure shall contain the amount of time before the timer expires, or
 65483 zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if
 65484 the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload
 65485 value last set by *timer_settime()*.

65486 The *timer_settime()* function shall set the time until the next expiration of the timer specified by
 65487 *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member
 65488 of *value* is non-zero. If the specified timer was already armed when *timer_settime()* is called, this
 65489 call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value*
 65490 is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending
 65491 expiration notifications is unspecified.

65492 If the flag `TIMER_ABSTIME` is not set in the argument *flags*, *timer_settime()* shall behave as if the
 65493 time until next expiration is set to be equal to the interval specified by the *it_value* member of
 65494 *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the
 65495 flag `TIMER_ABSTIME` is set in the argument *flags*, *timer_settime()* shall behave as if the time
 65496 until next expiration is set to be equal to the difference between the absolute time specified by
 65497 the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is,
 65498 the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*.
 65499 If the specified time has already passed, the function shall succeed and the expiration
 65500 notification shall be made.

65501 The reload value of the timer shall be set to the value specified by the *it_interval* member of
 65502 *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is
 65503 specified.

65504 Time values that are between two consecutive non-negative integer multiples of the resolution of
 65505 the specified timer shall be rounded up to the larger multiple of the resolution. Quantization
 65506 error shall not cause the timer to expire earlier than the rounded time value.

65507 If the argument *ovalue* is not NULL, the *timer_settime()* function shall store, in the location
 65508 referenced by *ovalue*, a value representing the previous amount of time before the timer would
 65509 have expired, or zero if the timer was disarmed, together with the previous timer reload value.
 65510 Timers shall not expire before their scheduled time.

65511 Only a single signal shall be queued to the process for a given timer at any point in time. When a
 65512 timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun
 65513 shall occur. When a timer expiration signal is delivered to or accepted by a process, the
 65514 *timer_getoverrun()* function shall return the timer expiration overrun count for the specified
 65515 timer. The overrun count returned contains the number of extra timer expirations that occurred

65516 between the time the signal was generated (queued) and when it was delivered or accepted, up
 65517 to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the
 65518 number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the
 65519 overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun()*
 65520 shall apply to the most recent expiration signal delivery or acceptance for the timer. If no
 65521 expiration signal has been delivered for the timer, the return value of *timer_getoverrun()* is
 65522 unspecified.

RETURN VALUE

65523 If the *timer_getoverrun()* function succeeds, it shall return the timer expiration overrun count as
 65524 explained above.
 65525

65526 If the *timer_gettime()* or *timer_settime()* functions succeed, a value of 0 shall be returned.

65527 If an error occurs for any of these functions, the value -1 shall be returned, and *errno* set to
 65528 indicate the error.

ERRORS

65529 The *timer_settime()* function shall fail if:

65531 [EINVAL] A *value* structure specified a nanosecond value less than zero or greater than
 65532 or equal to 1000 million, and the *it_value* member of that structure did not
 65533 specify zero seconds and nanoseconds.

65534 These functions may fail if:

65535 [EINVAL] The *timerid* argument does not correspond to an ID returned by *timer_create()*
 65536 but not yet deleted by *timer_delete()*.

65537 The *timer_settime()* function may fail if:

65538 [EINVAL] The *it_interval* member of *value* is not zero and the timer was created with
 65539 notification by creation of a new thread (*sigev_sigev_notify* was
 65540 SIGEV_THREAD) and a fixed stack address has been set in the thread
 65541 attribute pointed to by *sigev_notify_attributes*.

EXAMPLES

65542 None.
 65543

APPLICATION USAGE

65544 Using fixed stack addresses is problematic when timer expiration is signalled by the creation of a
 65545 new thread. Since it cannot be assumed that the thread created for one expiration is finished
 65546 before the next expiration of the timer, it could happen that two threads use the same memory as
 65547 a stack at the same time. This is invalid and produces undefined results.
 65548

RATIONALE

65549 Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The
 65550 inverse of this tick rate is the clock resolution, also called the clock granularity, which in either
 65551 case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for
 65552 these common rates. The granularity of practical clocks implies that if one reads a given clock
 65553 twice in rapid succession, one may get the same time value twice; and that timers must wait for
 65554 the next clock tick after the theoretical expiration time, to ensure that a timer never returns too
 65555 soon. Note also that the granularity of the clock may be significantly coarser than the resolution
 65556 of the data format used to set and get time and interval values. Also note that some
 65557 implementations may choose to adjust time and/or interval values to exactly match the ticks of
 65558 the underlying clock.
 65559

65560 This volume of POSIX.1-200x defines functions that allow an application to determine the
 65561 implementation-supported resolution for the clocks and requires an implementation to
 65562 document the resolution supported for timers and *nanosleep()* if they differ from the supported
 65563 clock resolution. This is more of a procurement issue than a runtime application issue.

65564
65565

65566
65567

65568

65569
65570

65571
65572
65573

65574
65575

65576
65577
65578

65579
65580
65581

65582
65583

65584
65585
65586

65587
65588
65589

65590
65591
65592

65593**FUTURE DIRECTIONS**

None.

SEE ALSO*clock_getres()*, *timer_create()*XBD <**time.h**>**CHANGE HISTORY**

First released in Issue 5. Included for alignment with the POSIX Realtime Extension.

Issue 6

The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are marked as part of the Timers option.

The [ENOSYS] error condition has been removed as stubs need not be provided if an implementation does not support the Timers option.

The [EINVAL] error condition is updated to include the following: “and the *it_value* member of that structure did not specify zero seconds and nanoseconds.” This change is for IEEE PASC Interpretation 1003.1 #89.

The DESCRIPTION for *timer_getoverrun()* is updated to clarify that “If no expiration signal has been delivered for the timer, or if the Realtime Signals Extension is not supported, the return value of *timer_getoverrun()* is unspecified”.

The **restrict** keyword is added to the *timer_settime()* prototype for alignment with the ISO/IEC 9899:1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/140 is applied, updating the ERRORS section so that the mandatory [EINVAL] error (“The *timerid* argument does not correspond to an ID returned by *timer_create()* but not yet deleted by *timer_delete()*”) becomes optional.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/141 is applied, updating the ERRORS section to include an optional [EINVAL] error for the case when a timer is created with the notification method set to SIGEV_THREAD. APPLICATION USAGE text is also added.

Issue 7

The *timer_getoverrun()*, *timer_gettime()*, and *timer_settime()* functions are moved from the Timers option to the Base.

Functionality relating to the Realtime Signals Extension option is moved to the Base.

65594 **NAME**
 65595 `times` — get process and waited-for child process times

65596 **SYNOPSIS**
 65597 `#include <sys/times.h>`
 65598 `clock_t times(struct tms *buffer);`

65599 **DESCRIPTION**
 65600 The `times()` function shall fill the `tms` structure pointed to by `buffer` with time-accounting
 65601 information. The `tms` structure is defined in `<sys/times.h>`.

65602 All times are measured in terms of the number of clock ticks used.

65603 The times of a terminated child process shall be included in the `tms_cutime` and `tms_cstime`
 65604 elements of the parent when `wait()`, `waitid()`, or `waitpid()` returns the process ID of this
 65605 terminated child. If a child process has not waited for its children, their times shall not be
 65606 included in its times.

- 65607 • The `tms_utime` structure member is the CPU time charged for the execution of user
 65608 instructions of the calling process.
- 65609 • The `tms_stime` structure member is the CPU time charged for execution by the system on
 65610 behalf of the calling process.
- 65611 • The `tms_cutime` structure member is the sum of the `tms_utime` and `tms_cutime` times of the
 65612 child processes.
- 65613 • The `tms_cstime` structure member is the sum of the `tms_stime` and `tms_cstime` times of the
 65614 child processes.

65615 **RETURN VALUE**
 65616 Upon successful completion, `times()` shall return the elapsed real time, in clock ticks, since an
 65617 arbitrary point in the past (for example, system start-up time). This point does not change from
 65618 one invocation of `times()` within the process to another. The return value may overflow the
 65619 possible range of type `clock_t`. If `times()` fails, `(clock_t)-1` shall be returned and `errno` set to
 65620 indicate the error.

65621 **ERRORS**
 65622 No errors are defined.

65623 EXAMPLES

65624 Timing a Database Lookup

65625 The following example defines two functions, `start_clock()` and `end_clock()`, that are used to time
 65626 a lookup. It also defines variables of type `clock_t` and `tms` to measure the duration of
 65627 transactions. The `start_clock()` function saves the beginning times given by the `times()` function.
 65628 The `end_clock()` function gets the ending times and prints the difference between the two times.

```
65629 #include <sys/times.h>
65630 #include <stdio.h>
65631 ...
65632 void start_clock(void);
65633 void end_clock(char *msg);
65634 ...
65635 static clock_t st_time;
65636 static clock_t en_time;
65637 static struct tms st_cpu;
```

```

65638     static struct tms en_cpu;
65639     ...
65640     void
65641     start_clock()
65642     {
65643         st_time = times(&st_cpu);
65644     }
65645     /* This example assumes that the result of each subtraction
65646        is within the range of values that can be represented in
65647        an integer type. */
65648     void
65649     end_clock(char *msg)
65650     {
65651         en_time = times(&en_cpu);
65652         fputs(msg, stdout);
65653         printf("Real Time: %jd, User Time %jd, System Time %jd\n",
65654             (intmax_t)(en_time - st_time),
65655             (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
65656             (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
65657     }

```

APPLICATION USAGE

Applications should use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per second as it may vary from system to system.

RATIONALE

The accuracy of the times reported is intentionally left unspecified to allow implementations flexibility in design, from uniprocessor to multi-processor networks.

The inclusion of times of child processes is recursive, so that a parent process may collect the total times of all of its descendants. But the times of a child are only added to those of its parent when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process can always see the total times of all its descendants; see also the discussion of the term “realtime” in [alarm\(\)](#).

If the type `clock_t` is defined to be a signed 32-bit integer, it overflows in somewhat more than a year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual systems that run continuously for longer than that. This volume of POSIX.1-200x permits an implementation to make the reference point for the returned value be the start-up time of the process, rather than system start-up time.

The term “charge” in this context has nothing to do with billing for services. The operating system accounts for time used in this way. That information must be correct, regardless of how that information is used.

FUTURE DIRECTIONS

None.

SEE ALSO

[alarm\(\)](#), [exec](#), [fork\(\)](#), [sysconf\(\)](#), [time](#), [wait](#), [waitid\(\)](#)

XBD [<sys/times.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

timezone()

65684 **NAME**
65685 timezone — difference from UTC and local standard time

SYNOPSIS

```
65686 XSI       #include <time.h>  
65687           extern long timezone;
```

DESCRIPTION

65689 Refer to [tzset\(\)](#).
65690

65691 **NAME**
 65692 tmpfile — create a temporary file

65693 **SYNOPSIS**
 65694 #include <stdio.h>
 65695 FILE *tmpfile(void);

65696 DESCRIPTION

65697 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 65698 conflict between the requirements described here and the ISO C standard is unintentional. This
 65699 volume of POSIX.1-200x defers to the ISO C standard.

65700 The *tmpfile()* function shall create a temporary file and open a corresponding stream. The file
 65701 shall be automatically deleted when all references to the file are closed. The file is opened as in
 65702 *fopen()* for update (*w+*), except that implementations may restrict the permissions, either by
 65703 clearing the file mode bits or setting them to the value `S_IRUSR | S_IWUSR`.

65704 CX In some implementations, a permanent file may be left behind if the process calling *tmpfile()* is
 65705 killed while it is processing a call to *tmpfile()*.

65706 An error message may be written to standard error if the stream cannot be opened.

65707 RETURN VALUE

65708 Upon successful completion, *tmpfile()* shall return a pointer to the stream of the file that is
 65709 CX created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

65710 ERRORS

65711 The *tmpfile()* function shall fail if:

65712 CX [EINTR] A signal was caught during *tmpfile()*.

65713 CX [EMFILE] All file descriptors available to the process are currently open.

65714 [EMFILE] {STREAM_MAX} streams are currently open in the calling process. +

65715 CX [ENFILE] The maximum allowable number of files is currently open in the system.

65716 CX [ENOSPC] The directory or file system which would contain the new file cannot be
 65717 expanded.

65718 CX [EOVERFLOW] The file is a regular file and the size of the file cannot be represented correctly
 65719 in an object of type `off_t`.

65720 The *tmpfile()* function may fail if:

65721 CX [EMFILE] {FOPEN_MAX} streams are currently open in the calling process.

65722 CX [ENOMEM] Insufficient storage space is available.

65723 EXAMPLES

65724 Creating a Temporary File

65725 The following example creates a temporary file for update, and returns a pointer to a stream for
 65726 the created file in the *fp* variable.

```
65727 #include <stdio.h>
65728 ...
65729 FILE *fp;
65730 fp = tmpfile ();
```

65731 **APPLICATION USAGE**

65732 It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the

65733 program (this limit may be shared with *tmpnam()*) and there should be no limit on the number

65734 simultaneously open other than this limit and any limit on the number of open file descriptors |

65735 or streams ({OPEN_MAX}, {FOPEN_MAX}, {STREAM_MAX}).

65736 **RATIONALE**

65737 None.

65738 **FUTURE DIRECTIONS**

65739 None.

65740 **SEE ALSO**

65741 *fopen()*, *mkdtemp()*, *tmpnam()*, *unlink*

65742 XBD <stdio.h> |

65743 **CHANGE HISTORY**

65744 First released in Issue 1. Derived from Issue 1 of the SVID.

65745 **Issue 5**

65746 Large File Summit extensions are added.

65747 The last two paragraphs of the DESCRIPTION were included as APPLICATION USAGE notes

65748 in previous issues.

65749 **Issue 6**

65750 Extensions beyond the ISO C standard are marked.

65751 The following new requirements on POSIX implementations derive from alignment with the

65752 Single UNIX Specification:

- 65753 • In the ERRORS section, the [Eoverflow] condition is added. This change is to support
- 65754 large files.
- 65755 • The [EMFILE] optional error condition is added.

65756 The APPLICATION USAGE section is added for alignment with the ISO/IEC 9899:1999

65757 standard.

65758 **Issue 7**

65759 Austin Group Interpretation 1003.1-2001 #025 is applied, clarifying that implementations may +

65760 restrict the permissions of the file created. +

65761 SD5-XBD-ERN-4 is applied, changing the definition of the [EMFILE] error.

65762 SD5-XSH-ERN-149 is applied, adding the mandatory [EMFILE] error condition for |

65763 {STREAM_MAX} streams open.

65764 **NAME**

65765 tmpnam — create a name for a temporary file

65766 **SYNOPSIS**

```
65767 OB #include <stdio.h>
65768 char *tmpnam(char *s);
```

65769 **DESCRIPTION**

65770 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 65771 conflict between the requirements described here and the ISO C standard is unintentional. This
 65772 volume of POSIX.1-200x defers to the ISO C standard.

65773 The *tmpnam()* function shall generate a string that is a valid filename and that is not the same as
 65774 the name of an existing file. The function is potentially capable of generating {TMP_MAX}
 65775 different strings, but any or all of them may already be in use by existing files and thus not be
 65776 suitable return values.

65777 The *tmpnam()* function generates a different string each time it is called from the same process,
 65778 up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is
 65779 implementation-defined.

65780 The implementation shall behave as if no function defined in this volume of POSIX.1-200x,
 65781 except *tempnam()*, calls *tmpnam()*.

65782 CX If the application uses any of the POSIX threads functions, the application shall ensure that the
 65783 *tmpnam()* function is called with a non-NULL parameter.

65784 **RETURN VALUE**

65785 Upon successful completion, *tmpnam()* shall return a pointer to a string. If no suitable string can
 65786 be generated, the *tmpnam()* function shall return a null pointer.

65787 If the argument *s* is a null pointer, *tmpnam()* shall leave its result in an internal static object and
 65788 return a pointer to that object. Subsequent calls to *tmpnam()* may modify the same object. If the
 65789 argument *s* is not a null pointer, it is presumed to point to an array of at least *L_tmpnam* **chars**;
 65790 *tmpnam()* shall write its result in that array and shall return the argument as its value.

65791 **ERRORS**

65792 No errors are defined.

65793 **EXAMPLES**65794 **Generating a Filename**65795 The following example generates a unique filename and stores it in the array pointed to by *ptr*.

```
65796 #include <stdio.h>
65797 ...
65798 char filename[L_tmpnam+1];
65799 char *ptr;

65800 ptr = tmpnam(filename);
```

65801 **APPLICATION USAGE**65802 This function only creates filenames. It is the application's responsibility to create and remove
65803 the files.65804 Between the time a pathname is created and the file is opened, it is possible for some other
65805 process to create a file with the same name. Applications may find *tmpfile()* more useful.

tmpnam()

65806 Applications should use the *tmpfile()*, *mkstemp()*, or *mkdtemp()* functions instead of the
65807 obsolescent *tmpnam()* function.

RATIONALE

65808 None.
65809

FUTURE DIRECTIONS

65810 The *tmpnam()* function may be removed in a future version.
65811

SEE ALSO

65812 *fopen()*, *open()*, *mkdtemp()*, *tempnam()*, *tmpfile()*, *unlink*
65813

65814 XBD <stdio.h> |

CHANGE HISTORY

65815 First released in Issue 1. Derived from Issue 1 of the SVID.
65816

Issue 5

65817 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.
65818

Issue 6

65819 Extensions beyond the ISO C standard are marked.
65820

65821 The normative text is updated to avoid use of the term “must” for application requirements.

65822 The DESCRIPTION is expanded for alignment with the ISO/IEC 9899:1999 standard.

65823 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/142 is applied, updating the
65824 DESCRIPTION to allow implementations of the *tempnam()* function to call *tmpnam()*.

Issue 7

65825 The *tmpnam()* function is marked obsolescent.
65826

65827 **NAME**
65828 toascii — translate an integer to a 7-bit ASCII character

65829 **SYNOPSIS**

```
65830 OB XSI #include <ctype.h>  
65831 int toascii(int c);
```

65832 **DESCRIPTION**

65833 The *toascii()* function shall convert its argument into a 7-bit ASCII character.

65834 **RETURN VALUE**

65835 The *toascii()* function shall return the value (*c* & 0x7f).

65836 **ERRORS**

65837 No errors are returned.

65838 **EXAMPLES**

65839 None.

65840 **APPLICATION USAGE**

65841 The *toascii()* function cannot be used portably in a localized application.

65842 **RATIONALE**

65843 None.

65844 **FUTURE DIRECTIONS**

65845 The *toascii()* function may be removed in a future version.

65846 **SEE ALSO**

65847 [isascii\(\)](#)

65848 XBD [<ctype.h>](#)

65849 **CHANGE HISTORY**

65850 First released in Issue 1. Derived from Issue 1 of the SVID.

65851 **Issue 7**

65852 The *toascii()* function is marked obsolescent.

65853 **NAME**
 65854 `tolower, tolower_l` — transliterate uppercase characters to lowercase

SYNOPSIS

```
65855     #include <ctype.h>
65856
65857     int tolower(int c);
65858 CX    int tolower_l(int c, locale_t locale);
```

DESCRIPTION

65859
 65860 CX For `tolower()`: The functionality described on this reference page is aligned with the ISO C
 65861 standard. Any conflict between the requirements described here and the ISO C standard is
 65862 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

65863 CX The `tolower()` and `tolower_l()` functions have as a domain a type `int`, the value of which is
 65864 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the
 65865 behavior is undefined. If the argument of `tolower()` or `tolower_l()` represents an uppercase letter,
 65866 and there exists a corresponding lowercase letter as defined by character type information in the
 65867 program locale or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the
 65868 result shall be the corresponding lowercase letter. All other arguments in the domain are
 65869 returned unchanged.

RETURN VALUE

65870
 65871 CX Upon successful completion, the `tolower()` and `tolower_l()` functions shall return the lowercase
 65872 letter corresponding to the argument passed; otherwise, they shall return the argument
 65873 unchanged.

ERRORS

65874
 65875 The `tolower_l()` function may fail if:

65876 CX `[EINVAL]` `locale` is not a valid locale object handle.

EXAMPLES

65877
 65878 None.

APPLICATION USAGE

65879
 65880 None.

RATIONALE

65881
 65882 None.

FUTURE DIRECTIONS

65883
 65884 None.

SEE ALSO

65885 [*setlocale\(\), uselocale\(\)*](#)
 65886
 65887 XBD [Chapter 7](#) (on page 121), [<ctype.h>](#), [<locale.h>](#)

CHANGE HISTORY

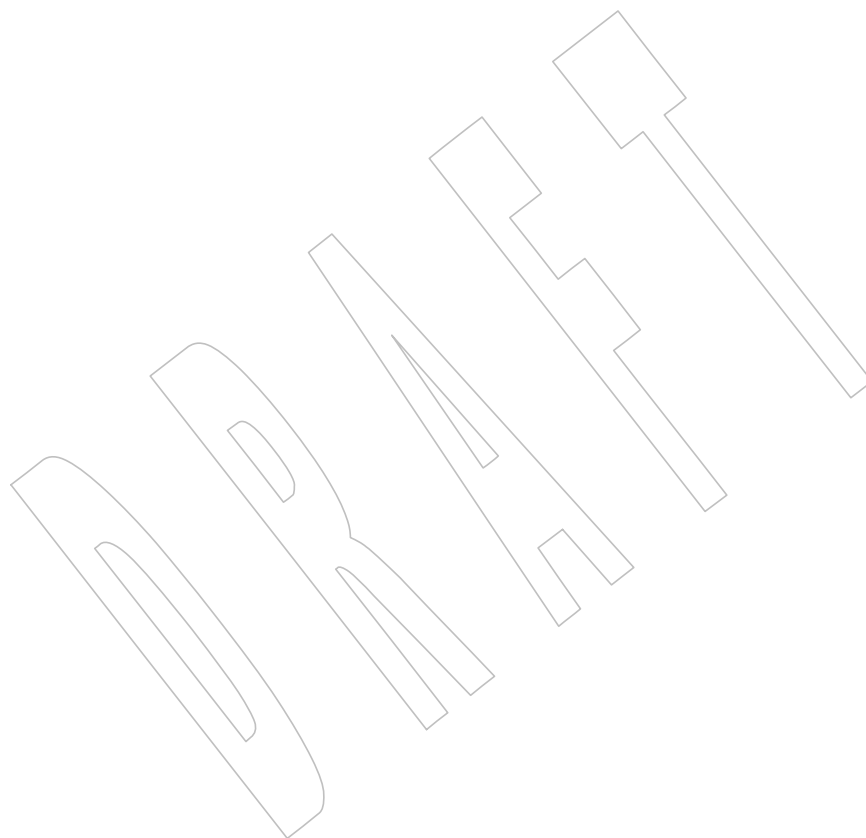
65888
 65889 First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

65890
 65891 Extensions beyond the ISO C standard are marked.

65892
65893
65894**Issue 7**

The *tolower_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.



65895 **NAME**65896 `toupper, toupper_l` — transliterate lowercase characters to uppercase65897 **SYNOPSIS**65898 `#include <ctype.h>`65899 `int toupper(int c);`65900 CX `int toupper_l(int c, locale_t locale);`65901 **DESCRIPTION**65902 CX For `toupper()`: The functionality described on this reference page is aligned with the ISO C
65903 standard. Any conflict between the requirements described here and the ISO C standard is
65904 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.65905 CX The `toupper()` and `toupper_l()` functions have as a domain a type **int**, the value of which is
65906 representable as an **unsigned char** or the value of EOF. If the argument has any other value, the
65907 behavior is undefined. If the argument of `toupper()` or `toupper_l()` represents a lowercase letter,
65908 and there exists a corresponding uppercase letter as defined by character type information in the
65909 program locale or in the locale represented by `locale`, respectively (category `LC_CTYPE`), the
65910 result shall be the corresponding uppercase letter. All other arguments in the domain are
65911 returned unchanged.65912 **RETURN VALUE**65913 CX Upon successful completion, `toupper()` and `toupper_l()` shall return the uppercase letter
65914 corresponding to the argument passed; otherwise, they shall return the argument unchanged. |65915 **ERRORS**65916 The `toupper_l()` function may fail if:65917 CX `[EINVAL]` `locale` is not a valid locale object handle.65918 **EXAMPLES**

65919 None.

65920 **APPLICATION USAGE**

65921 None.

65922 **RATIONALE**

65923 None.

65924 **FUTURE DIRECTIONS**

65925 None.

65926 **SEE ALSO**65927 [*setlocale\(\), uselocale\(\)*](#)65928 XBD [Chapter 7](#) (on page 121), [*<ctype.h>*](#), [*<locale.h>*](#) |65929 **CHANGE HISTORY**

65930 First released in Issue 1. Derived from Issue 1 of the SVID.

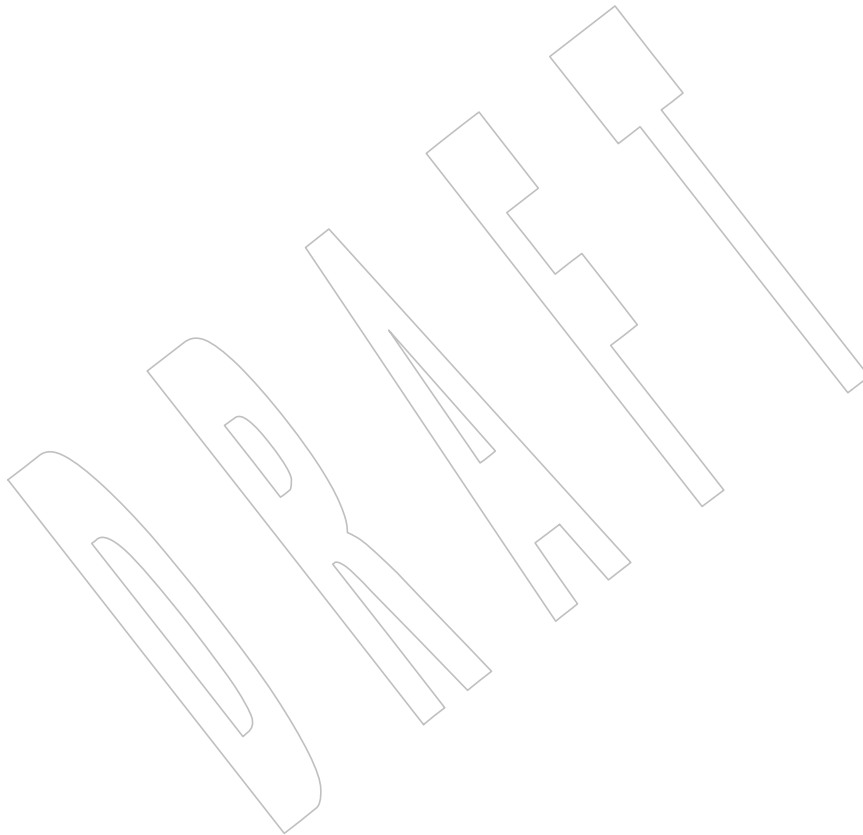
65931 **Issue 6**

65932 Extensions beyond the ISO C standard are marked.

65933 **Issue 7**

65934 SD5-XSH-ERN-181 is applied, clarifying the RETURN VALUE section. |

65935 The `toupper_l()` function is added from The Open Group Technical Standard, 2006, Extended API
65936 Set Part 4.



65937 **NAME**
 65938 towctrans, towctrans_l — wide-character transliteration

65939 **SYNOPSIS**

65940 #include <wctype.h>
 65941 wint_t towctrans(wint_t wc, wctrans_t desc);
 65942 CX wint_t towctrans_l(wint_t wc, wctrans_t desc,
 65943 locale_t locale);

65944 **DESCRIPTION**

65945 CX For *towctrans()*: The functionality described on this reference page is aligned with the ISO C
 65946 standard. Any conflict between the requirements described here and the ISO C standard is
 65947 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

65948 CX The *towctrans()* and *towctrans_l()* functions shall transliterate the wide-character code *wc* using
 65949 the mapping described by *desc*.

65950 CX The current setting of the *LC_CTYPE* category in the current locale of the process or in the locale
 65951 CX represented by *locale*, respectively, should be the same as during the call to *wctrans()* or
 65952 *wctrans_l()* that returned the value *desc*.

65953 If the value of *desc* is invalid (that is, not obtained by a call to *wctrans()* or *desc* is invalidated by a
 65954 subsequent call to *setlocale()* that has affected category *LC_CTYPE*), the result is unspecified.

65955 CX If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l()* with the same locale
 65956 object *locale*) the result is unspecified.

65957 CX An application wishing to check for error situations should set *errno* to 0 before calling
 65958 *towctrans()* or *towctrans_l()*.

65959 If *errno* is non-zero on return, an error has occurred.

65960 **RETURN VALUE**

65961 CX If successful, the *towctrans()* and *towctrans_l()* functions shall return the mapped value of *wc*
 65962 using the mapping described by *desc*. Otherwise, they shall return *wc* unchanged.

65963 **ERRORS**

65964 These functions may fail if:

65965 CX [EINVAL] *desc* contains an invalid transliteration descriptor.

65966 The *towctrans_l()* function may fail if:

65967 CX [EINVAL] *locale* is not a valid locale object handle.

65968 **EXAMPLES**

65969 None.

65970 **APPLICATION USAGE**

65971 The strings "tolower" and "toupper" are reserved for the standard mapping names. In the
 65972 table below, the functions in the left column are equivalent to the functions in the right column.

65973	towlower(<i>wc</i>)	towctrans(<i>wc</i> , wctrans("tolower"))
65974	towlower_l(<i>wc</i> , <i>locale</i>)	towctrans_l(<i>wc</i> , wctrans("tolower"), <i>locale</i>)
65975	towupper(<i>wc</i>)	towctrans(<i>wc</i> , wctrans("toupper"))
65976	towupper_l(<i>wc</i> , <i>locale</i>)	towctrans_l(<i>wc</i> , wctrans("toupper"), <i>locale</i>)

65977 **RATIONALE**

65978 None.

65979 **FUTURE DIRECTIONS**

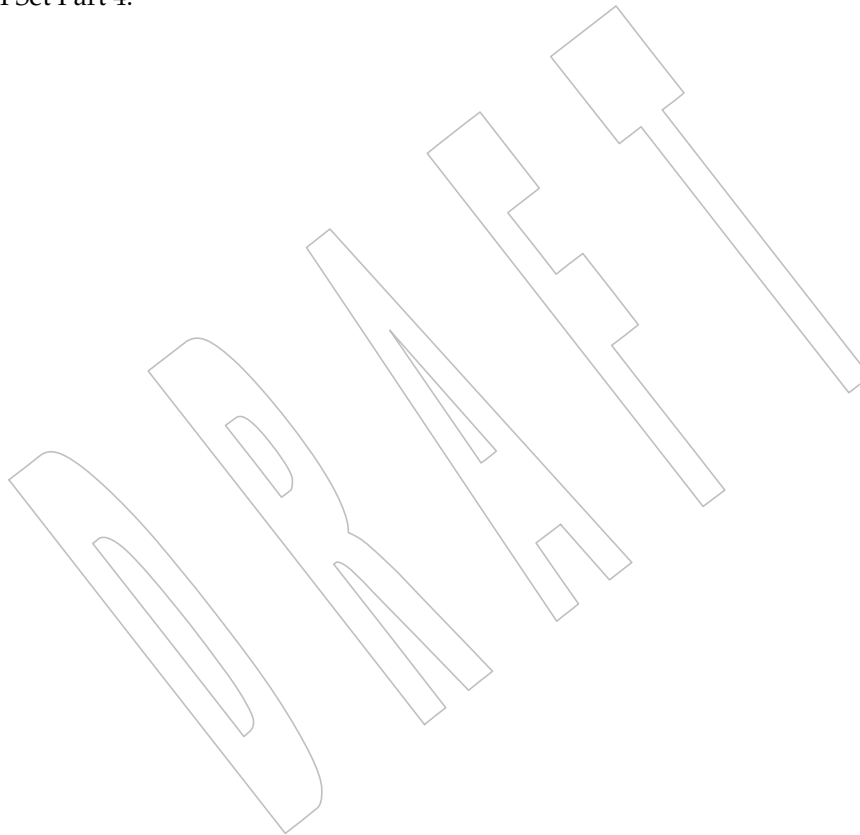
65980 None.

65981 **SEE ALSO**65982 *tolower()*, *toupper()*, *wctrans()*65983 XBD <*wctype.h*>65984 **CHANGE HISTORY**

65985 First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

65986 **Issue 6**

65987 Extensions beyond the ISO C standard are marked.

65988 **Issue 7**65989 The *towctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended
65990 API Set Part 4.

65991 **NAME**

65992 towlower, towlower_l — transliterate uppercase wide-character code to lowercase

65993 **SYNOPSIS**

65994 #include <wctype.h>

65995 wint_t towlower(wint_t wc);

65996 CX wint_t towlower_l(wint_t wc, locale_t locale);

65997 **DESCRIPTION**65998 CX For *towlower()*: The functionality described on this reference page is aligned with the ISO C
65999 standard. Any conflict between the requirements described here and the ISO C standard is
66000 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.66001 CX The *towlower()* and *towlower_l()* functions have as a domain a type **wint_t**, the value of which
66002 the application shall ensure is a character representable as a **wchar_t**, and a wide-character code
66003 corresponding to a valid character in the current locale or the value of WEOF. If the argument
66004 CX has any other value, the behavior is undefined. If the argument of *towlower()* or *towlower_l()*
66005 represents an uppercase wide-character code, and there exists a corresponding lowercase wide-
66006 CX character code as defined by character type information in the locale of the process or in the
66007 locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the
66008 corresponding lowercase wide-character code. All other arguments in the domain are returned
66009 unchanged.66010 **RETURN VALUE**66011 CX Upon successful completion, the *towlower()* and *towlower_l()* functions shall return the
66012 lowercase letter corresponding to the argument passed; otherwise, they shall return the
66013 argument unchanged.66014 **ERRORS**66015 The *towlower_l()* function may fail if:66016 CX [EINVAL] *locale* is not a valid locale object handle.66017 **EXAMPLES**

66018 None.

66019 **APPLICATION USAGE**

66020 None.

66021 **RATIONALE**

66022 None.

66023 **FUTURE DIRECTIONS**

66024 None.

66025 **SEE ALSO**66026 *setlocale()*, *uselocale()*

66027 XBD Chapter 7 (on page 121), <locale.h>, <wctype.h>

66028 **CHANGE HISTORY**

66029 First released in Issue 4.

66030 **Issue 5**

66031 The following change has been made in this version for alignment with |
66032 ISO/IEC 9899:1990/Amendment 1:1995 (E):

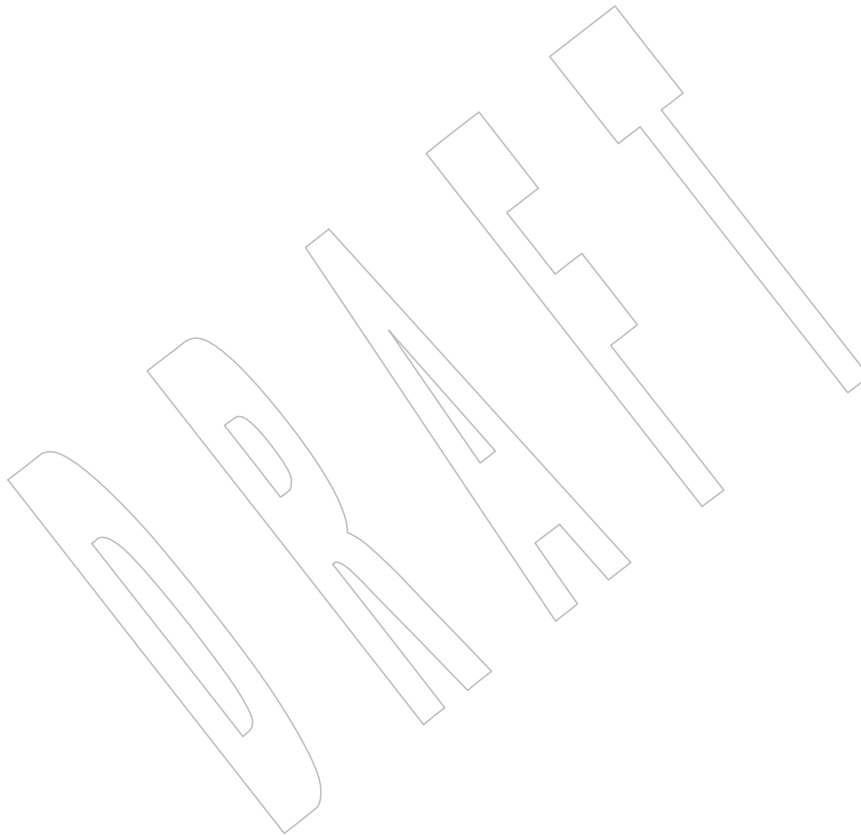
- 66033 • The SYNOPSIS has been changed to indicate that this function and associated data types
66034 are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

66035 **Issue 6**

66036 The normative text is updated to avoid use of the term “must” for application requirements.

66037 **Issue 7**

66038 The `tolower_l()` function is added from The Open Group Technical Standard, 2006, Extended
66039 API Set Part 4.



66040 **NAME**

66041 towupper, towupper_l — transliterate lowercase wide-character code to uppercase

66042 **SYNOPSIS**

66043 #include <wctype.h>

66044 wint_t towupper(wint_t wc);

66045 CX wint_t towupper_l(wint_t wc, locale_t locale);

66046 **DESCRIPTION**66047 CX For *towupper()*: The functionality described on this reference page is aligned with the ISO C
66048 standard. Any conflict between the requirements described here and the ISO C standard is
66049 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.66050 CX The *towupper()* and *towupper_l()* functions have as a domain a type **wint_t**, the value of which
66051 the application shall ensure is a character representable as a **wchar_t**, and a wide-character code
66052 corresponding to a valid character in the current locale or the value of WEOF. If the argument
66053 has any other value, the behavior is undefined. If the argument of *towupper()* or *towupper_l()*
66054 represents a lowercase wide-character code, and there exists a corresponding uppercase wide-
66055 character code as defined by character type information in the locale of the process or in the
66056 locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the
66057 corresponding uppercase wide-character code. All other arguments in the domain are returned
66058 unchanged.66059 **RETURN VALUE**66060 CX Upon successful completion, the *towupper()* and *towupper_l()* functions shall return the
66061 uppercase letter corresponding to the argument passed. Otherwise, they shall return the
66062 argument unchanged.66063 **ERRORS**66064 The *towupper_l()* function may fail if:66065 CX [EINVAL] *locale* is not a valid locale object handle.66066 **EXAMPLES**

66067 None.

66068 **APPLICATION USAGE**

66069 None.

66070 **RATIONALE**

66071 None.

66072 **FUTURE DIRECTIONS**

66073 None.

66074 **SEE ALSO**66075 *setlocale()*, *uselocale()*

66076 XBD Chapter 7 (on page 121), <locale.h>, <wctype.h>

66077 **CHANGE HISTORY**

66078 First released in Issue 4.

66079

Issue 5

66080

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

66081

66082

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the `<wctype.h>` header rather than `<wchar.h>`.

66083

66084

Issue 6

66085

The normative text is updated to avoid use of the term “must” for application requirements.

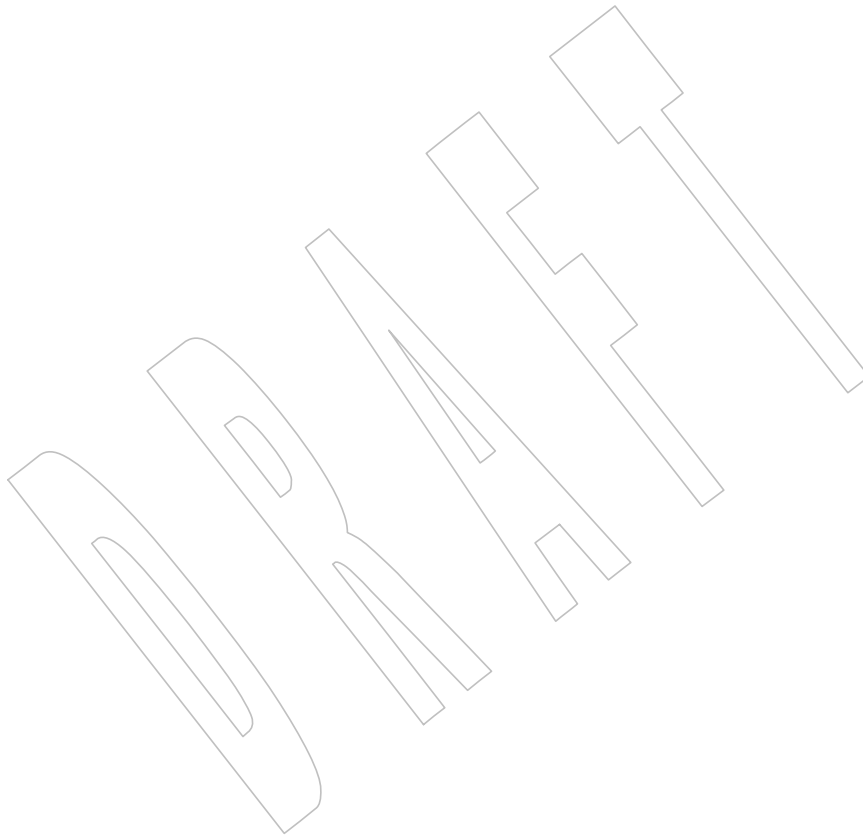
66086

Issue 7

66087

The `towupper_l()` function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

66088



66089 **NAME**
 66090 `trunc`, `truncf`, `truncl` — round to truncated integer value

66091 **SYNOPSIS**
 66092 `#include <math.h>`
 66093 `double trunc(double x);`
 66094 `float truncf(float x);`
 66095 `long double truncl(long double x);`

66096 **DESCRIPTION**
 66097 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 66098 conflict between the requirements described here and the ISO C standard is unintentional. This
 66099 volume of POSIX.1-200x defers to the ISO C standard.

66100 These functions shall round their argument to the integer value, in floating format, nearest to but
 66101 no larger in magnitude than the argument.

66102 **RETURN VALUE**
 66103 Upon successful completion, these functions shall return the truncated integer value.

66104 MX If x is NaN, a NaN shall be returned.
 66105 If x is ± 0 or $\pm \text{Inf}$, x shall be returned.

66106 **ERRORS**
 66107 No errors are defined.

66108 **EXAMPLES**
 66109 None.

66110 **APPLICATION USAGE**
 66111 None.

66112 **RATIONALE**
 66113 None.

66114 **FUTURE DIRECTIONS**
 66115 None.

66116 **SEE ALSO**
 66117 XBD [<math.h>](#)

66118 **CHANGE HISTORY**
 66119 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

66120 **NAME**

66121 truncate — truncate a file to a specified length

66122 **SYNOPSIS**

66123 #include <unistd.h>

66124 int truncate(const char *path, off_t length);

66125 **DESCRIPTION**66126 The *truncate()* function shall cause the regular file named by *path* to have a size which shall be
66127 equal to *length* bytes.66128 If the file previously was larger than *length*, the extra data is discarded. If the file was previously
66129 shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

66130 The application shall ensure that the process has write permission for the file.

66131 If the request would cause the file size to exceed the soft file size limit for the process, the
66132 request shall fail and the implementation shall generate the SIGXFSZ signal for the process.66133 The *truncate()* function shall not modify the file offset for any open file descriptions associated
66134 with the file. Upon successful completion, if the file size is changed, *truncate()* shall mark for
66135 update the last data modification and last file status change timestamps of the file, and the
66136 S_ISUID and S_ISGID bits of the file mode may be cleared.66137 **RETURN VALUE**66138 Upon successful completion, *truncate()* shall return 0. Otherwise, -1 shall be returned, and *errno*
66139 set to indicate the error.66140 **ERRORS**66141 The *truncate()* function shall fail if:

66142 [EINTR] A signal was caught during execution.

66143 [EINVAL] The *length* argument was less than 0.

66144 [EFBIG] or [EINVAL]

66145 The *length* argument was greater than the maximum file size.

66146 [EIO] An I/O error occurred while reading from or writing to a file system.

66147 [EACCES] A component of the path prefix denies search permission, or write permission
66148 is denied on the file.

66149 [EISDIR] The named file is a directory.

66150 [ELOOP] A loop exists in symbolic links encountered during resolution of the *path*
66151 argument.

66152 [ENAMETOOLONG]

66153 The length of the *path* argument exceeds {PATH_MAX} or a pathname
66154 component is longer than {NAME_MAX}.66155 [ENOENT] A component of *path* does not name an existing file or *path* is an empty string.66156 [ENOTDIR] A component of the path prefix of *path* is not a directory.

66157 [EROFS] The named file resides on a read-only file system.

truncate()

66158 The *truncate()* function may fail if:

66159 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
66160 resolution of the *path* argument.

66161 [ENAMETOOLONG]
66162 Pathname resolution of a symbolic link produced an intermediate result
66163 whose length exceeds {PATH_MAX}.

66164 **EXAMPLES**
66165 None.

66166 **APPLICATION USAGE**
66167 None.

66168 **RATIONALE**
66169 None.

66170 **FUTURE DIRECTIONS**
66171 None.

66172 **SEE ALSO**
66173 *open()*
66174 XBD <[unistd.h](#)>

66175 **CHANGE HISTORY**
66176 First released in Issue 4, Version 2.

66177 **Issue 5**
66178 Moved from X/OPEN UNIX extension to BASE.
66179 Large File Summit extensions are added.

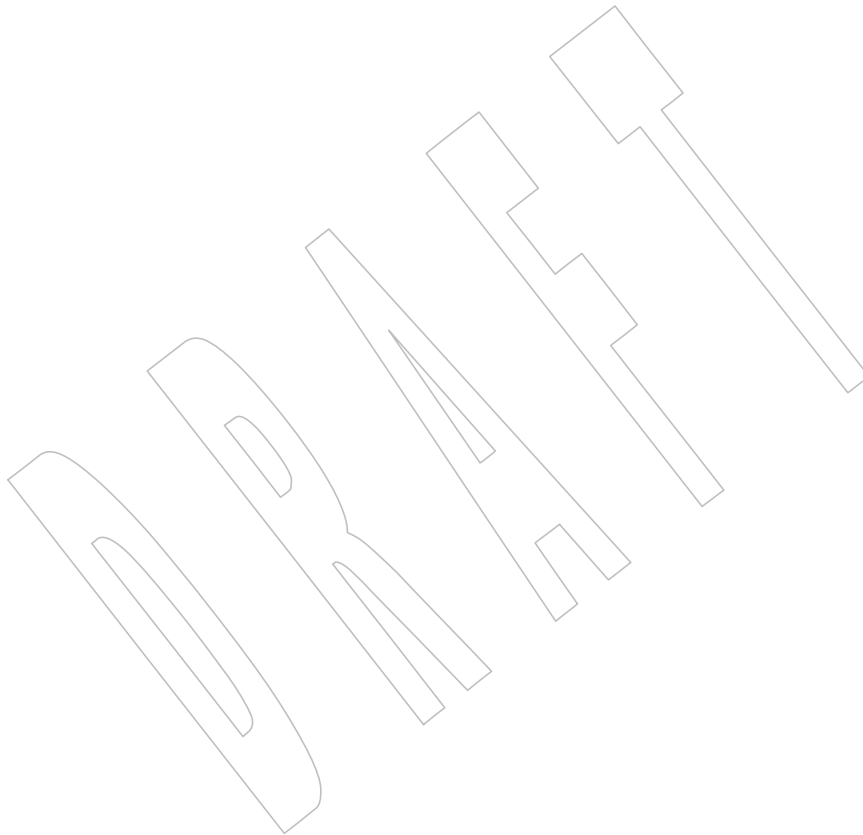
66180 **Issue 6**
66181 This reference page is split out from the *truncate()* reference page.
66182 The normative text is updated to avoid use of the term “must” for application requirements.
66183 The wording of the mandatory [ELOOP] error condition is updated, and a second optional
66184 [ELOOP] error condition is added.

66185 **Issue 7**
66186 The *truncate()* function is moved from the XSI option to the Base.
66187 Changes are made related to support for finegrained timestamps.

66188 **NAME**
66189 `truncf, trunc1` — round to truncated integer value

66190 **SYNOPSIS**
66191 `#include <math.h>`
66192 `float truncf(float x);`
66193 `long double trunc1(long double x);`

66194 **DESCRIPTION**
66195 Refer to *trunc()*.



tsearch()

66196 **NAME**
66197 `tsearch` — search a binary search tree

SYNOPSIS

```
66198 XSI #include <search.h>  
66200 void *tsearch(const void *key, void **rootp,  
66201 int (*compar)(const void *, const void *));
```

DESCRIPTION

66202 Refer to [tdelete\(\)](#).
66203

66204 **NAME**
 66205 `ttyname, ttyname_r` — find the pathname of a terminal

66206 **SYNOPSIS**
 66207 `#include <unistd.h>`
 66208 `char *ttyname(int fildev);`
 66209 `int ttyname_r(int fildev, char *name, size_t namesize);`

66210 **DESCRIPTION**
 66211 The `ttyname()` function shall return a pointer to a string containing a null-terminated pathname
 66212 of the terminal associated with file descriptor *fildev*. The return value may point to static data
 66213 whose content is overwritten by each call.

66214 The `ttyname()` function need not be thread-safe. A function that is not required to be thread-safe
 66215 is not required to be reentrant.

66216 The `ttyname_r()` function shall store the null-terminated pathname of the terminal associated
 66217 with the file descriptor *fildev* in the character array referenced by *name*. The array is *namesize*
 66218 characters long and should have space for the name and the terminating null character. The
 66219 maximum length of the terminal name shall be {TTY_NAME_MAX}.

66220 **RETURN VALUE**
 66221 Upon successful completion, `ttyname()` shall return a pointer to a string. Otherwise, a null
 66222 pointer shall be returned and *errno* set to indicate the error.

66223 If successful, the `ttyname_r()` function shall return zero. Otherwise, an error number shall be
 66224 returned to indicate the error.

66225 **ERRORS**
 66226 The `ttyname()` function may fail if:
 66227 [EBADF] The *fildev* argument is not a valid file descriptor.
 66228 [ENOTTY] The file associated with the *fildev* argument is not a terminal.

66229 The `ttyname_r()` function may fail if:
 66230 [EBADF] The *fildev* argument is not a valid file descriptor.
 66231 [ENOTTY] The file associated with the *fildev* argument is not a terminal.
 66232 [ERANGE] The value of *namesize* is smaller than the length of the string to be returned
 66233 including the terminating null character.

66234 **EXAMPLES**
 66235 None.

66236 **APPLICATION USAGE**
 66237 None.

66238 **RATIONALE**
 66239 The term “terminal” is used instead of the historical term “terminal device” in order to avoid a
 66240 reference to an undefined term.

66241 The thread-safe version places the terminal name in a user-supplied buffer and returns a non-
 66242 zero value if it fails. The non-thread-safe version may return the name in a static data area that
 66243 may be overwritten by each call.

66244

FUTURE DIRECTIONS

66245

None.

66246

SEE ALSO

66247

XBD <[unistd.h](#)>

66248

CHANGE HISTORY

66249

First released in Issue 1. Derived from Issue 1 of the SVID.

66250

Issue 5

66251

The *ttyname_r()* function is included for alignment with the POSIX Threads Extension.

66252

A note indicating that the *ttyname()* function need not be reentrant is added to the DESCRIPTION.

66253

66254

Issue 6

66255

The *ttyname_r()* function is marked as part of the Thread-Safe Functions option.

66256

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

66257

66258

- The statement that *errno* is set on error is added.

66259

- The [EBADF] and [ENOTTY] optional error conditions are added.

66260

Issue 7

66261

SD5-XSH-ERN-100 is applied, correcting the definition of the [ENOTTY] error condition.

66262

The *ttyname_r()* function is moved from the Thread-Safe Functions option to the Base.

66263 **NAME**
66264 twalk — traverse a binary search tree

66265 **SYNOPSIS**

```
66266 XSI #include <search.h>  
66267 void twalk(const void *root,  
66268           void (*action)(const void *, VISIT, int ));
```

66269 **DESCRIPTION**

66270 Refer to *tdelete()*.

66271 **NAME**
 66272 daylight, timezone, tzname, tzset — set timezone conversion information

66273 **SYNOPSIS**
 66274 #include <time.h>

66275 XSI extern int daylight;
 66276 extern long timezone;
 66277 CX extern char *tzname[2];
 66278 void tzset(void);

66279 **DESCRIPTION**
 66280 The *tzset()* function shall use the value of the environment variable *TZ* to set time conversion
 66281 information used by *ctime()*, *localtime()*, *mktime()*, and *strftime()*. If *TZ* is absent from the
 66282 environment, implementation-defined default timezone information shall be used.

66283 The *tzset()* function shall set the external variable *tzname* as follows:

66284 tzname[0] = "std";
 66285 tzname[1] = "dst";

66286 where *std* and *dst* are as described in XBD Chapter 8 (on page 159).

66287 XSI The *tzset()* function also shall set the external variable *daylight* to 0 if Daylight Savings Time
 66288 conversions should never be applied for the timezone in use; otherwise, non-zero. The external
 66289 variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time
 66290 (UTC) and local standard time.

66291 **RETURN VALUE**
 66292 The *tzset()* function shall not return a value.

66293 **ERRORS**
 66294 No errors are defined.

66295 **EXAMPLES**
 66296 Example *TZ* variables and their timezone differences are given in the table below:

<i>TZ</i>	<i>timezone</i>
EST5EDT	5*60*60
GMT0	0*60*60
JST-9	-9*60*60
MET-1MEST	-1*60*60
MST7MDT	7*60*60
PST8PDT	8*60*60

66304 **APPLICATION USAGE**
 66305 None.

66306 **RATIONALE**
 66307 None.

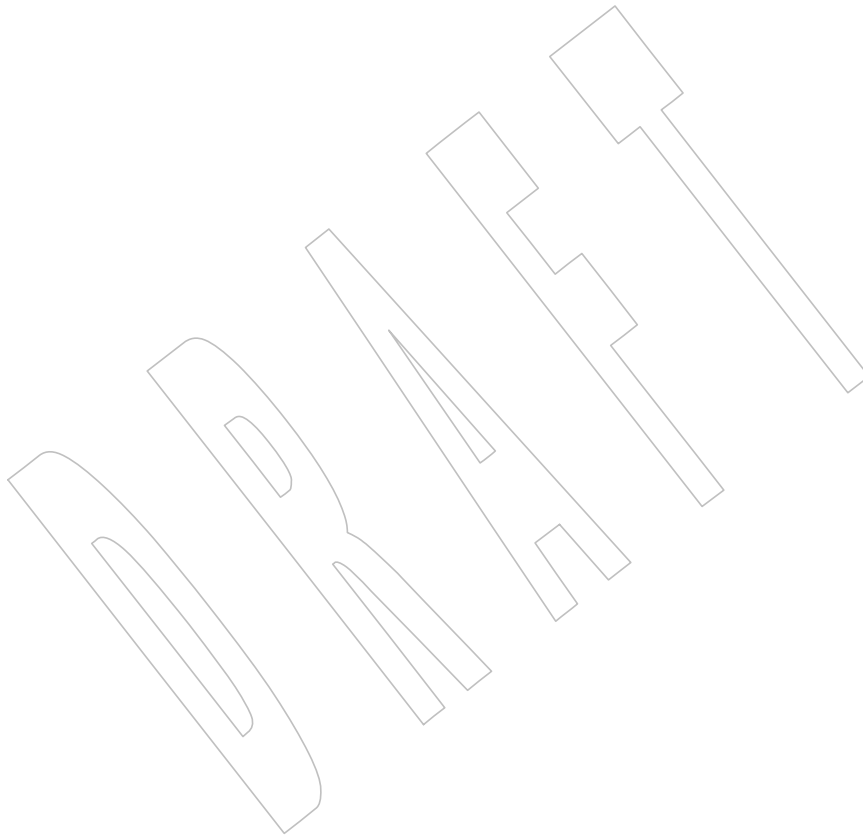
66308 **FUTURE DIRECTIONS**
 66309 None.

66310
66311
66312
66313
66314
66315
66316**SEE ALSO***ctime()*, *localtime()*, *mktime()*, *strftime()*XBD Chapter 8 (on page 159), **<time.h>****CHANGE HISTORY**

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

The example is corrected.



66317 **NAME**66318 `ulimit` — get and set process limits66319 **SYNOPSIS**

```
66320 OB XSI #include <ulimit.h>
66321 long ulimit(int cmd, ...);
```

66322 **DESCRIPTION**

66323 The `ulimit()` function shall control process limits. The process limits that can be controlled by
 66324 this function include the maximum size of a single file that can be written (this is equivalent to
 66325 using `setrlimit()` with `RLIMIT_FSIZE`). The `cmd` values, defined in `<ulimit.h>`, include:

66326 `UL_GETFSIZE` Return the file size limit (`RLIMIT_FSIZE`) of the process. The limit shall be in
 66327 units of 512-byte blocks and shall be inherited by child processes. Files of any
 66328 size can be read. The return value shall be the integer part of the soft file size
 66329 limit divided by 512. If the result cannot be represented as a **long**, the result is
 66330 unspecified.

66331 `UL_SETFSIZE` Set the file size limit for output operations of the process to the value of the
 66332 second argument, taken as a **long**, multiplied by 512. If the result would
 66333 overflow an `rlim_t`, the actual value set is unspecified. Any process may
 66334 decrease its own limit, but only a process with appropriate privileges may
 66335 increase the limit. The return value shall be the integer part of the new file size
 66336 limit divided by 512.

66337 The `ulimit()` function shall not change the setting of `errno` if successful.

66338 As all return values are permissible in a successful situation, an application wishing to check for
 66339 error situations should set `errno` to 0, then call `ulimit()`, and, if it returns `-1`, check to see if `errno` is
 66340 non-zero.

66341 **RETURN VALUE**

66342 Upon successful completion, `ulimit()` shall return the value of the requested limit. Otherwise, `-1`
 66343 shall be returned and `errno` set to indicate the error.

66344 **ERRORS**

66345 The `ulimit()` function shall fail and the limit shall be unchanged if:

66346 `[EINVAL]` The `cmd` argument is not valid.
 66347 `[EPERM]` A process not having appropriate privileges attempts to increase its file size
 66348 limit.

66349 **EXAMPLES**

66350 None.

66351 **APPLICATION USAGE**

66352 Since the `ulimit()` function uses type **long** rather than `rlim_t`, this function is not sufficient for file
 66353 sizes on many current systems. Applications should use the `getrlimit()` or `setrlimit()` functions
 66354 instead of the obsolescent `ulimit()` function.

66355 **RATIONALE**

66356 None.

66357

FUTURE DIRECTIONS

66358

The *ulimit()* function may be removed in a future version.

66359

SEE ALSO

66360

exec, *getrlimit()*, *write*

66361

XBD <**ulimit.h**>

66362

CHANGE HISTORY

66363

First released in Issue 1. Derived from Issue 1 of the SVID.

66364

Issue 5

66365

In the description of `UL_SETFSIZE`, the text is corrected to refer to `rlim_t` rather than the spurious `rlimit_t`.

66366

66367

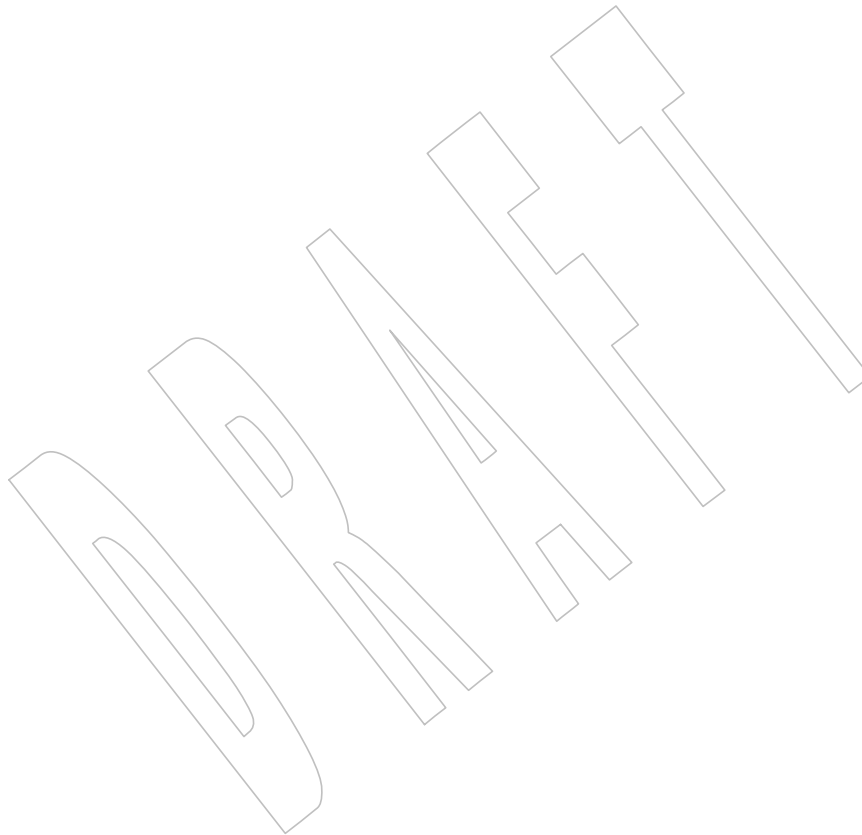
The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

66368

Issue 7

66369

The *ulimit()* function is marked obsolescent.



66370 **NAME**
 66371 `umask` — set and get the file mode creation mask

66372 **SYNOPSIS**
 66373 `#include <sys/stat.h>`
 66374 `mode_t umask(mode_t cmask);`

66375 **DESCRIPTION**
 66376 The `umask()` function shall set the file mode creation mask of the process to `cmask` and return the
 66377 previous value of the mask. Only the file permission bits of `cmask` (see `<sys/stat.h>`) are used; the
 66378 meaning of the other bits is implementation-defined.

66379 The file mode creation mask of the process is used to turn off permission bits in the `mode`
 66380 argument supplied during calls to the following functions:

- 66381 • `open()`, `openat()`, `creat()`, `mkdir()`, `mkdirat()`, `mkfifo()`, and `mkfifoat()`
- 66382 XSI • `mknod()`, `mknodat()`
- 66383 MSG • `mq_open()`
- 66384 • `sem_open()`

66385 Bit positions that are set in `cmask` are cleared in the mode of the created file.

66386 **RETURN VALUE**
 66387 The file permission bits in the value returned by `umask()` shall be the previous value of the file
 66388 mode creation mask. The state of any other bits in that value is unspecified, except that a
 66389 subsequent call to `umask()` with the returned value as `cmask` shall leave the state of the mask the
 66390 same as its state before the first call, including any unspecified use of those bits.

66391 **ERRORS**
 66392 No errors are defined.

66393 **EXAMPLES**
 66394 None.

66395 **APPLICATION USAGE**
 66396 None.

66397 **RATIONALE**
 66398 Unsigned argument and return types for `umask()` were proposed. The return type and the
 66399 argument were both changed to **mode_t**.

66400 Historical implementations have made use of additional bits in `cmask` for their implementation-
 66401 defined purposes. The addition of the text that the meaning of other bits of the field is
 66402 implementation-defined permits these implementations to conform to this volume of
 66403 POSIX.1-200x.

66404 **FUTURE DIRECTIONS**
 66405 None.

66406 **SEE ALSO**
 66407 `creat()`, `exec`, `mkdir`, `mkfifo`, `mknod()`, `mq_open()`, `open()`, `sem_open()`
 66408 XBD `<sys/stat.h>`, `<sys/types.h>`

66409

CHANGE HISTORY

66410

First released in Issue 1. Derived from Issue 1 of the SVID.

66411

Issue 6

66412

In the SYNOPSIS, the optional include of the `<sys/types.h>` header is removed.

66413

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

66414

66415

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

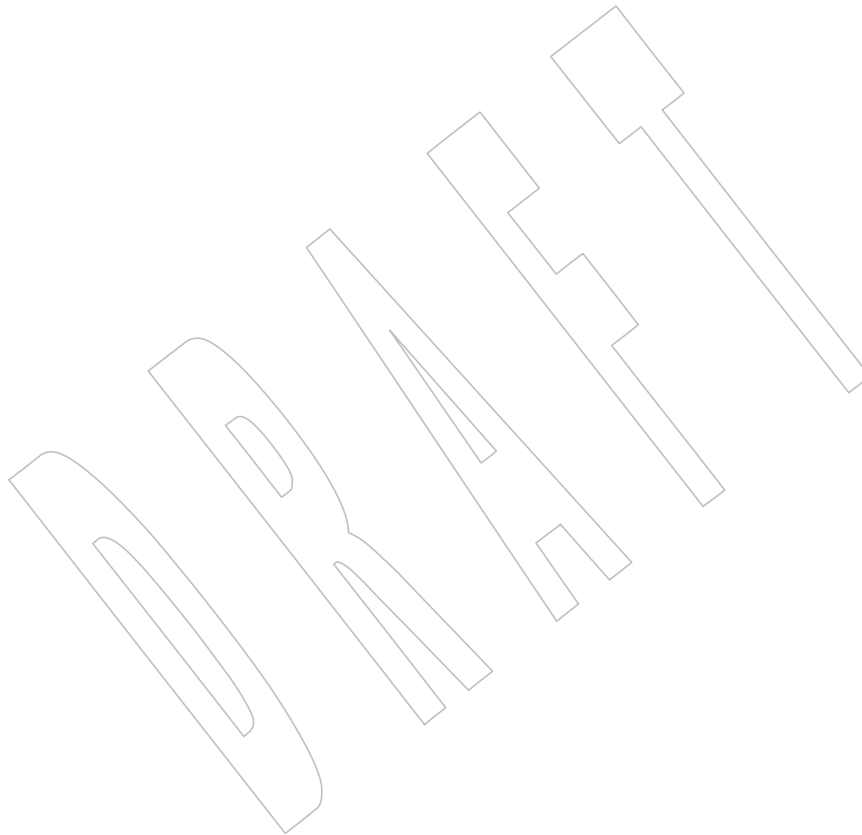
66416

66417

66418

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/143 is applied, adding the `mknod()`, `mq_open()`, and `sem_open()` functions to the DESCRIPTION and SEE ALSO sections.

66419



66420 **NAME**66421 `uname` — get the name of the current system66422 **SYNOPSIS**66423 `#include <sys/utsname.h>`66424 `int uname(struct utsname *name);`66425 **DESCRIPTION**66426 The `uname()` function shall store information identifying the current system in the structure
66427 pointed to by `name`.66428 The `uname()` function uses the **utsname** structure defined in `<sys/utsname.h>`.66429 The `uname()` function shall return a string naming the current system in the character array
66430 `sysname`. Similarly, `nodename` shall contain the name of this node within an implementation-
66431 defined communications network. The arrays `release` and `version` shall further identify the
66432 operating system. The array `machine` shall contain a name that identifies the hardware that the
66433 system is running on.

66434 The format of each member is implementation-defined.

66435 **RETURN VALUE**66436 Upon successful completion, a non-negative value shall be returned. Otherwise, `-1` shall be
66437 returned and `errno` set to indicate the error.66438 **ERRORS**

66439 No errors are defined.

66440 **EXAMPLES**

66441 None.

66442 **APPLICATION USAGE**66443 The inclusion of the `nodename` member in this structure does not imply that it is sufficient
66444 information for interfacing to communications networks.66445 **RATIONALE**66446 The values of the structure members are not constrained to have any relation to the version of
66447 this volume of POSIX.1-200x implemented in the operating system. An application should
66448 instead depend on `_POSIX_VERSION` and related constants defined in `<unistd.h>`.66449 This volume of POSIX.1-200x does not define the sizes of the members of the structure and
66450 permits them to be of different sizes, although most implementations define them all to be the
66451 same size: eight bytes plus one byte for the string terminator. That size for `nodename` is not
66452 enough for use with many networks.66453 The `uname()` function originated in System III, System V, and related implementations, and it
66454 does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in
66455 those historical implementations.66456 4.3 BSD has `gethostname()` and `gethostid()`, which return a symbolic name and a numeric value,
66457 respectively. There are related `sethostname()` and `sethostid()` functions that are used to set the
66458 values the other two functions return. The former functions are included in this specification, the
66459 latter are not.

66460

FUTURE DIRECTIONS

66461

None.

66462

SEE ALSO

66463

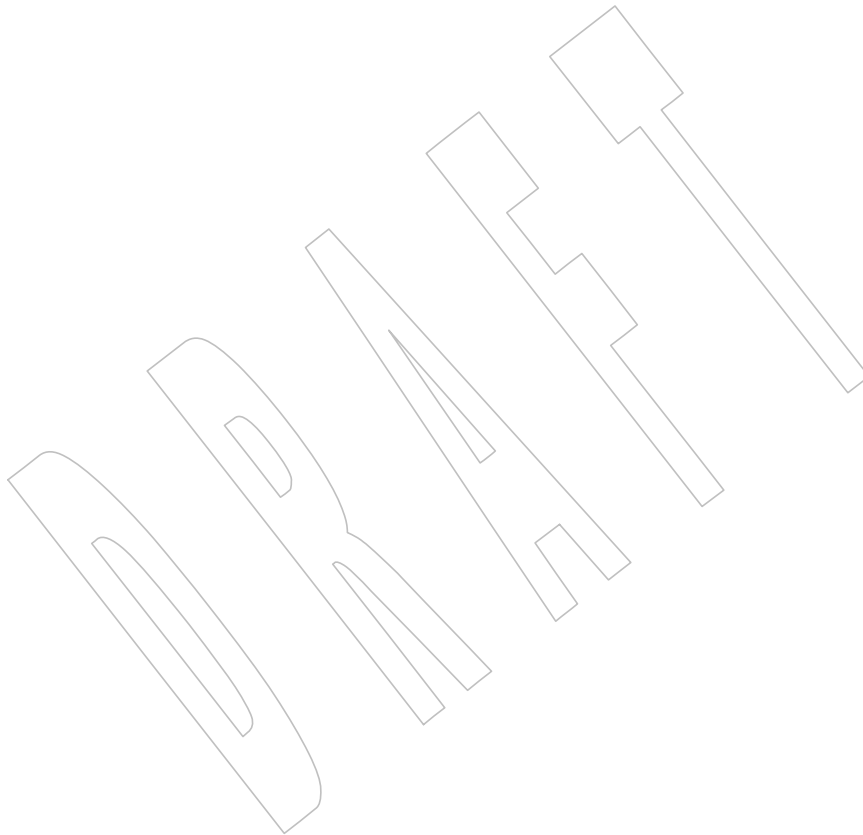
XBD <[sys/utsname.h](#)>

66464

CHANGE HISTORY

66465

First released in Issue 1. Derived from Issue 1 of the SVID.



66466 **NAME**
 66467 `ungetc` — push byte back into input stream

66468 **SYNOPSIS**
 66469 `#include <stdio.h>`
 66470 `int ungetc(int c, FILE *stream);`

66471 **DESCRIPTION**
 66472 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 66473 conflict between the requirements described here and the ISO C standard is unintentional. This
 66474 volume of POSIX.1-200x defers to the ISO C standard.

66475 The `ungetc()` function shall push the byte specified by `c` (converted to an **unsigned char**) back
 66476 onto the input stream pointed to by `stream`. The pushed-back bytes shall be returned by
 66477 subsequent reads on that stream in the reverse order of their pushing. A successful intervening
 66478 call (with the stream pointed to by `stream`) to a file-positioning function (`fseek()`, `fsetpos()`, or
 66479 `rewind()`) shall discard any pushed-back bytes for the stream. The external storage
 66480 corresponding to the stream shall be unchanged.

66481 One byte of push-back shall be provided. If `ungetc()` is called too many times on the same stream
 66482 without an intervening read or file-positioning operation on that stream, the operation may fail.

66483 If the value of `c` equals that of the macro `EOF`, the operation shall fail and the input stream shall
 66484 be left unchanged.

66485 A successful call to `ungetc()` shall clear the end-of-file indicator for the stream. The value of the
 66486 file-position indicator for the stream after reading or discarding all pushed-back bytes shall be
 66487 the same as it was before the bytes were pushed back. The file-position indicator is decremented
 66488 by each successful call to `ungetc()`; if its value was 0 before a call, its value is unspecified after
 66489 the call.

66490 **RETURN VALUE**
 66491 Upon successful completion, `ungetc()` shall return the byte pushed back after conversion.
 66492 Otherwise, it shall return `EOF`.

66493 **ERRORS**
 66494 No errors are defined.

66495 **EXAMPLES**
 66496 None.

66497 **APPLICATION USAGE**
 66498 None.

66499 **RATIONALE**
 66500 None.

66501 **FUTURE DIRECTIONS**
 66502 None.

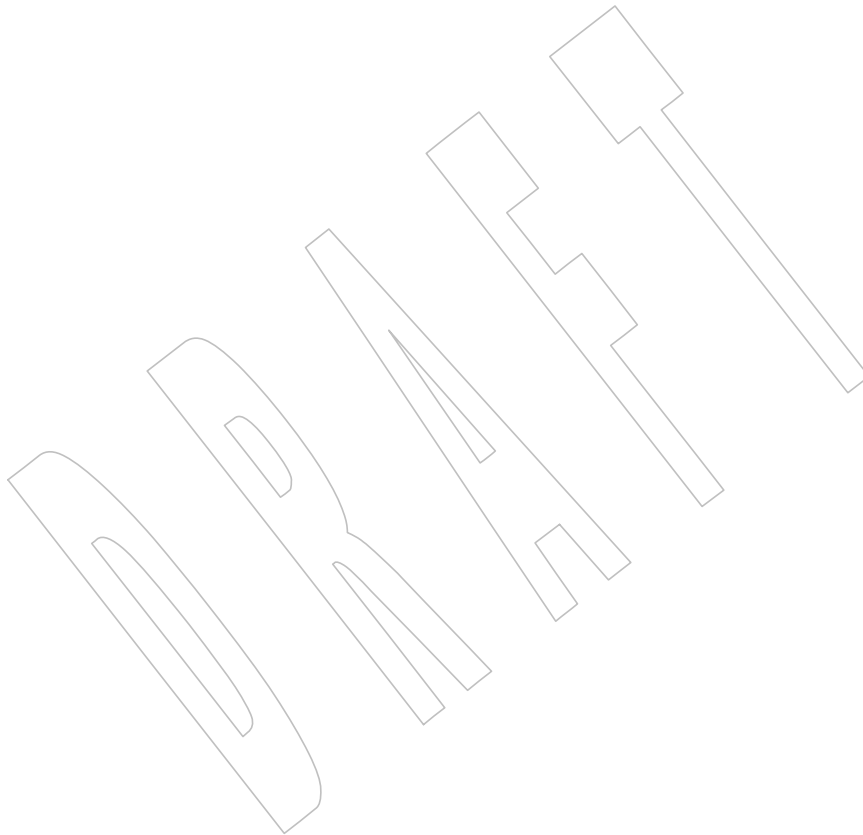
66503 **SEE ALSO**
 66504 [fseek\(\)](#), [getc\(\)](#), [fsetpos\(\)](#), [read](#), [rewind\(\)](#), [setbuf\(\)](#)
 66505 XBD [<stdio.h>](#)

66506

66507

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.



66508 **NAME**
 66509 ungetwc — push wide-character code back into the input stream

66510 **SYNOPSIS**
 66511 #include <stdio.h>
 66512 #include <wchar.h>
 66513 wint_t ungetwc(wint_t wc, FILE *stream);

66514 **DESCRIPTION**
 66515 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 66516 conflict between the requirements described here and the ISO C standard is unintentional. This
 66517 volume of POSIX.1-200x defers to the ISO C standard.

66518 The *ungetwc()* function shall push the character corresponding to the wide-character code
 66519 specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters
 66520 shall be returned by subsequent reads on that stream in the reverse order of their pushing. A
 66521 successful intervening call (with the stream pointed to by *stream*) to a file-positioning function
 66522 (*fseek()*, *fsetpos()*, or *rewind()*) discards any pushed-back characters for the stream. The external
 66523 storage corresponding to the stream is unchanged.

66524 At least one character of push-back shall be provided. If *ungetwc()* is called too many times on
 66525 the same stream without an intervening read or file-positioning operation on that stream, the
 66526 operation may fail.

66527 If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream
 66528 shall be left unchanged.

66529 A successful call to *ungetwc()* shall clear the end-of-file indicator for the stream. The value of the
 66530 file-position indicator for the stream after reading or discarding all pushed-back characters shall
 66531 be the same as it was before the characters were pushed back. The file-position indicator is
 66532 decremented (by one or more) by each successful call to *ungetwc()*; if its value was 0 before a
 66533 call, its value is unspecified after the call.

66534 **RETURN VALUE**
 66535 Upon successful completion, *ungetwc()* shall return the wide-character code corresponding to
 66536 the pushed-back character. Otherwise, it shall return WEOF.

66537 **ERRORS**
 66538 The *ungetwc()* function may fail if:

66539 CX [EILSEQ] An invalid character sequence is detected, or a wide-character code does not
 66540 correspond to a valid character.

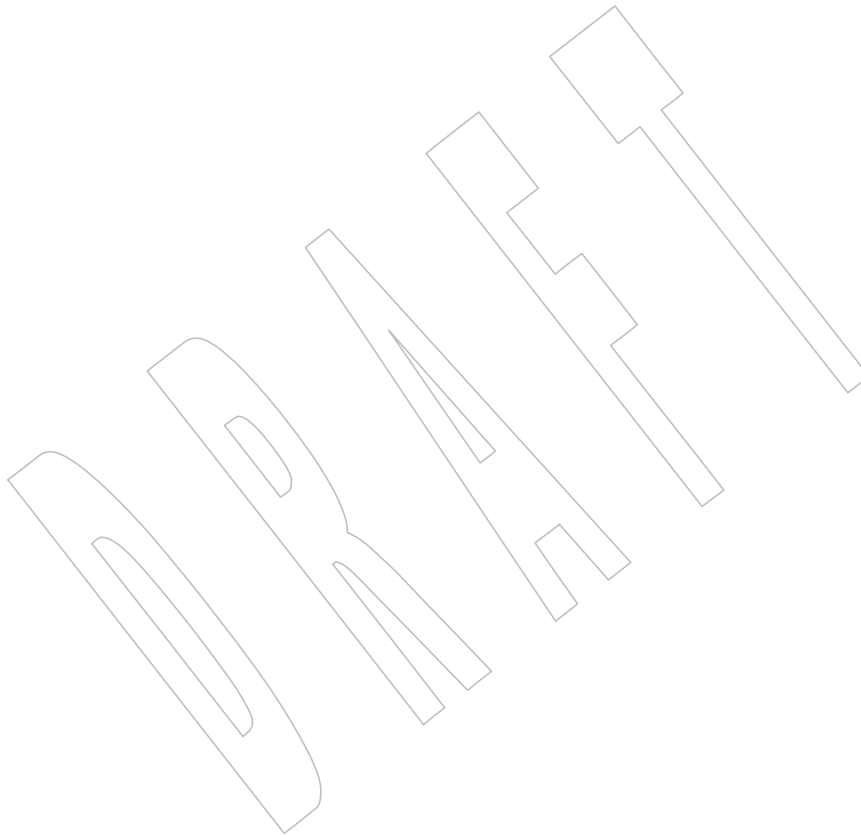
66541 **EXAMPLES**
 66542 None.

66543 **APPLICATION USAGE**
 66544 None.

66545 **RATIONALE**
 66546 None.

66547 **FUTURE DIRECTIONS**
 66548 None.

- 66549 **SEE ALSO**
- 66550 *fseek()*, *fsetpos()*, *read*, *rewind()*, *setbuf()*
- 66551 XBD `<stdio.h>`, `<wchar.h>`
- 66552 **CHANGE HISTORY**
- 66553 First released in Issue 4. Derived from the MSE working draft.
- 66554 **Issue 5**
- 66555 The Optional Header (OH) marking is removed from `<stdio.h>`.
- 66556 **Issue 6**
- 66557 The [EILSEQ] optional error condition is marked CX.



66558 **NAME**66559 `unlink, unlinkat` — remove a directory entry relative to directory file descriptor66560 **SYNOPSIS**66561 `#include <unistd.h>`66562 `int unlink(const char *path);`66563 `int unlinkat(int fd, const char *path, int flag);`66564 **DESCRIPTION**

66565 The `unlink()` function shall remove a link to a file. If `path` names a symbolic link, `unlink()` shall
 66566 remove the symbolic link named by `path` and shall not affect any file or directory named by the
 66567 contents of the symbolic link. Otherwise, `unlink()` shall remove the link named by the pathname
 66568 pointed to by `path` and shall decrement the link count of the file referenced by the link.

66569 When the file's link count becomes 0 and no process has the file open, the space occupied by the
 66570 file shall be freed and the file shall no longer be accessible. If one or more processes have the file
 66571 open when the last link is removed, the link shall be removed before `unlink()` returns, but the
 66572 removal of the file contents shall be postponed until all references to the file are closed.

66573 The `path` argument shall not name a directory unless the process has appropriate privileges and
 66574 the implementation supports using `unlink()` on directories.

66575 Upon successful completion, `unlink()` shall mark for update the last data modification and last
 66576 file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last
 66577 file status change timestamp of the file shall be marked for update.

66578 The `unlinkat()` function shall be equivalent to the `unlink()` or `rmdir()` function except in the case
 66579 where `path` specifies a relative path. In this case the directory entry to be removed is determined
 66580 relative to the directory associated with the file descriptor `fd` instead of the current working
 66581 directory. It is unspecified whether directory searches are permitted based on whether the file
 66582 was opened with search permission or on the current permissions of the directory underlying
 66583 the file descriptor.

66584 Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined
 66585 in `<fcntl.h>`:

66586 `AT_REMOVEDIR` Remove the directory entry specified by `fd` and `path` as a directory, not a
 66587 normal file.

66588 If `unlinkat()` is passed the special value `AT_FDCWD` in the `fd` parameter, the current working
 66589 directory is used and the behavior shall be identical to a call to `unlink()` or `rmdir()` respectively,
 66590 depending on whether or not the `AT_REMOVEDIR` bit is set in `flag`.

66591 **RETURN VALUE**

66592 Upon successful completion, these functions shall return 0. Otherwise, these functions shall
 66593 return `-1` and set `errno` to indicate the error. If `-1` is returned, the named file shall not be changed.

66594 **ERRORS**

66595 These functions shall fail and shall not unlink the file if:

66596 `[EACCES]` Search permission is denied for a component of the path prefix, or write
 66597 permission is denied on the directory containing the directory entry to be
 66598 removed.

66599 `[EBUSY]` The file named by the `path` argument cannot be unlinked because it is being
 66600 used by the system or another process and the implementation considers this
 66601 an error.

66602	[ELOOP]	A loop exists in symbolic links encountered during resolution of the <i>path</i> argument.
66603		
66604	[ENAMETOOLONG]	
66605		The length of the <i>path</i> argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}.
66606		
66607	[ENOENT]	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
66608	[ENOTDIR]	A component of the path prefix is not a directory.
66609	[EPERM]	The file named by <i>path</i> is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using <i>unlink()</i> on directories.
66610		
66611		
66612	XSI [EPERM] or [EACCES]	
66613		The S_ISVTX flag is set on the directory containing the file referred to by the <i>path</i> argument and the caller is not the file owner, nor is the caller the directory owner, nor does the caller have appropriate privileges.
66614		
66615		
66616	[EROFS]	The directory entry to be unlinked is part of a read-only file system.
66617		The <i>unlinkat()</i> function shall fail if:
66618	[EBADF]	The <i>path</i> argument does not specify an absolute path and the <i>fd</i> argument is neither AT_FDCWD nor a valid file descriptor open for reading.
66619		
66620	[EEXIST] or [ENOTEMPTY]	
66621		The <i>flag</i> parameter has the AT_REMOVEDIR bit set and the <i>path</i> argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.
66622		
66623		
66624	[ENOTDIR]	The <i>flag</i> parameter has the AT_REMOVEDIR bit set and <i>path</i> does not name a directory.
66625		
66626		These functions may fail and not unlink the file if:
66627	XSI [EBUSY]	The file named by <i>path</i> is a named STREAM.
66628	[ELOOP]	More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the <i>path</i> argument.
66629		
66630	[ENAMETOOLONG]	
66631		As a result of encountering a symbolic link in resolution of the <i>path</i> argument, the length of the substituted pathname string exceeded {PATH_MAX}.
66632		
66633	[ETXTBSY]	The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.
66634		
66635		The <i>unlinkat()</i> function may fail if:
66636	[EINVAL]	The value of the <i>flag</i> argument is not valid.
66637	[ENOTDIR]	The <i>path</i> argument is not an absolute path and <i>fd</i> is neither AT_FDCWD nor a file descriptor associated with a directory.
66638		

66639 EXAMPLES

66640 **Removing a Link to a File**

66641 The following example shows how to remove a link to a file named `/home/cnd/mod1` by
66642 removing the entry named `/modules/pass1`.

```
66643 #include <unistd.h>
66644
66644 char *path = "/modules/pass1";
66645 int status;
66646 ...
66647 status = unlink(path);
```

66648 **Checking for an Error**

66649 The following example fragment creates a temporary password lock file named **LOCKFILE**,
66650 which is defined as `/etc/ptmp`, and gets a file descriptor for it. If the file cannot be opened for
66651 writing, `unlink()` is used to remove the link between the file descriptor and **LOCKFILE**.

```
66652 #include <sys/types.h>
66653 #include <stdio.h>
66654 #include <fcntl.h>
66655 #include <errno.h>
66656 #include <unistd.h>
66657 #include <sys/stat.h>
66658
66658 #define LOCKFILE "/etc/ptmp"
66659
66659 int pfd; /* Integer for file descriptor returned by open call. */
66660 FILE *fpfd; /* File pointer for use in putpwent(). */
66661 ...
66662 /* Open password Lock file. If it exists, this is an error. */
66663 if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR
66664 | S_IWUSR | S_IRGRP | S_IROTH)) == -1) {
66665     fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
66666     exit(1);
66667 }
66668
66668 /* Lock file created; proceed with fdopen of lock file so that
66669 putpwent() can be used.
66670 */
66671 if ((fpfd = fdopen(pfd, "w")) == NULL) {
66672     close(pfd);
66673     unlink(LOCKFILE);
66674     exit(1);
66675 }
```

66676 **Replacing Files**

66677 The following example fragment uses `unlink()` to discard links to files, so that they can be
66678 replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error
66679 occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can
66680 be created, then removes the link to **LOCKFILE** when it is no longer needed.

```
66681 #include <sys/types.h>
66682 #include <stdio.h>
66683 #include <fcntl.h>
```

```

66684     #include <errno.h>
66685     #include <unistd.h>
66686     #include <sys/stat.h>
66687
66687     #define LOCKFILE "/etc/ptmp"
66688     #define PASSWDFILE "/etc/passwd"
66689     #define SAVEFILE "/etc/opasswd"
66690     ...
66691     /* If no change was made, assume error and leave passwd unchanged. */
66692     if (!valid_change) {
66693         fprintf(stderr, "Could not change password for user %s\n", user);
66694         unlink(LOCKFILE);
66695         exit(1);
66696     }
66697
66697     /* Change permissions on new password file. */
66698     chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);
66699
66699     /* Remove saved password file. */
66700     unlink(SAVEFILE);
66701
66701     /* Save current password file. */
66702     link(PASSWDFILE, SAVEFILE);
66703
66703     /* Remove current password file. */
66704     unlink(PASSWDFILE);
66705
66705     /* Save new password file as current password file. */
66706     link(LOCKFILE, PASSWDFILE);
66707
66707     /* Remove lock file. */
66708     unlink(LOCKFILE);
66709
66709     exit(0);

```

APPLICATION USAGE

Applications should use *rmdir()* to remove a directory.

RATIONALE

Unlinking a directory is restricted to the superuser in many historical implementations for reasons given in *link()* (see also *rename()*).

The meaning of [EBUSY] in historical implementations is “mount point busy”. Since this volume of POSIX.1-200x does not cover the system administration concepts of mounting and unmounting, the description of the error was changed to “resource busy”. (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working directory of any process.

The standard developers reviewed TR 24715-2006 and noted that LSB-conforming implementations may return [EISDIR] instead of [EPERM] when unlinking a directory. A change to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was considered, but decided against since it would break existing strictly conforming and conforming applications. Applications written for portability to both POSIX.1-200x and the LSB should be prepared to handle either error code.

The purpose of the *unlinkat()* function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *unlink()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *unlinkat()* function it can be guaranteed that the removed directory entry is located relative to the desired directory.

66732 **FUTURE DIRECTIONS**

66733 None.

66734 **SEE ALSO**66735 *close(), link, remove(), rename(), rmdir, symlink()*66736 XBD *<fcntl.h>*, *<unistd.h>*66737 **CHANGE HISTORY**

66738 First released in Issue 1. Derived from Issue 1 of the SVID.

66739 **Issue 5**

66740 The [EBUSY] error is added to the optional part of the ERRORS section.

66741 **Issue 6**66742 The following new requirements on POSIX implementations derive from alignment with the
66743 Single UNIX Specification:

- 66744 • In the DESCRIPTION, the effect is specified if *path* specifies a symbolic link.
- 66745 • The [ELOOP] mandatory error condition is added.
- 66746 • A second [ENAMETOOLONG] is added as an optional error condition.
- 66747 • The [ETXTBSY] optional error condition is added.

66748 The following changes were made to align with the IEEE P1003.1a draft standard:

- 66749 • The [ELOOP] optional error condition is added.

66750 The normative text is updated to avoid use of the term “must” for application requirements.

66751 **Issue 7**66752 Text arising from the LSB Conflicts TR is added to the RATIONALE about the use of [EPERM]
66753 and [EISDIR].66754 The *unlinkat()* function is added from The Open Group Technical Standard, 2006, Extended API
66755 Set Part 2.

66756 Changes are made related to support for finegrained timestamps. +

66757 **NAME**
66758 unlinkat — remove a directory entry relative to directory file descriptor

66759 **SYNOPSIS**
66760 #include <unistd.h>
66761 int unlinkat(int *fd*, const char **path*, int *flag*);

66762 **DESCRIPTION**
66763 Refer to *unlink*.



66764 **NAME**
 66765 unlockpt — unlock a pseudo-terminal master/slave pair

66766 **SYNOPSIS**
 66767 XSI `#include <stdlib.h>`
 66768 `int unlockpt(int fildev);`

66769 **DESCRIPTION**
 66770 The *unlockpt()* function shall unlock the slave pseudo-terminal device associated with the master
 66771 to which *fildev* refers.

66772 Conforming applications shall ensure that they call *unlockpt()* before opening the slave side of a
 66773 pseudo-terminal device.

66774 **RETURN VALUE**
 66775 Upon successful completion, *unlockpt()* shall return 0. Otherwise, it shall return -1 and set *errno*
 66776 to indicate the error.

66777 **ERRORS**
 66778 The *unlockpt()* function may fail if:
 66779 [EBADF] The *fildev* argument is not a file descriptor open for writing.
 66780 [EINVAL] The *fildev* argument is not associated with a master pseudo-terminal device.

66781 **EXAMPLES**
 66782 None.

66783 **APPLICATION USAGE**
 66784 None.

66785 **RATIONALE**
 66786 None.

66787 **FUTURE DIRECTIONS**
 66788 None.

66789 **SEE ALSO**
 66790 *grantpt()*, *open()*, *ptsname()*
 66791 XBD `<stdlib.h>`

66792 **CHANGE HISTORY**
 66793 First released in Issue 4, Version 2.

66794 **Issue 5**
 66795 Moved from X/OPEN UNIX extension to BASE.

66796 **Issue 6**
 66797 The normative text is updated to avoid use of the term “must” for application requirements.

66798 **NAME**
 66799 unsetenv — remove an environment variable

66800 **SYNOPSIS**

66801 CX `#include <stdlib.h>`
 66802 `int unsetenv(const char *name);`

66803 **DESCRIPTION**

66804 The *unsetenv()* function shall remove an environment variable from the environment of the
 66805 calling process. The *name* argument points to a string, which is the name of the variable to be
 66806 removed. The named argument shall not contain an '=' character. If the named variable does
 66807 not exist in the current environment, the environment shall be unchanged and the function is
 66808 considered to have completed successfully.

66809 If the application modifies *environ* or the pointers to which it points, the behavior of *unsetenv()* is
 66810 undefined. The *unsetenv()* function shall update the list of pointers to which *environ* points.

66811 The *unsetenv()* function need not be thread-safe. A function that is not required to be thread-safe
 66812 is not required to be reentrant.

66813 **RETURN VALUE**

66814 Upon successful completion, zero shall be returned. Otherwise, -1 shall be returned, *errno* set to
 66815 indicate the error, and the environment shall be unchanged.

66816 **ERRORS**

66817 The *unsetenv()* function shall fail if:

66818 [EINVAL] The *name* argument is a null pointer, points to an empty string, or points to a
 66819 string containing an '=' character.

66820 **EXAMPLES**

66821 None.

66822 **APPLICATION USAGE**

66823 None.

66824 **RATIONALE**

66825 Refer to the RATIONALE section in *setenv()*.

66826 **FUTURE DIRECTIONS**

66827 None.

66828 **SEE ALSO**

66829 *getenv()*, *setenv()*

66830 XBD [<stdlib.h>](#), [<sys/types.h>](#), [<unistd.h>](#)

66831 **CHANGE HISTORY**

66832 First released in Issue 6. Derived from the IEEE P1003.1a draft standard.

66833 **NAME**

66834 uselocale — use locale in current thread

66835 **SYNOPSIS**

```
66836 CX      #include <locale.h>
66837      locale_t uselocale(locale_t newloc);
```

66838 **DESCRIPTION**

66839 The *uselocale()* function shall set the current locale for the current thread to the locale
66840 represented by *newloc*.

66841 The value for the *newloc* argument shall be one of the following:

- 66842 1. A value returned by the *newlocale()* or *duplocale()* functions
- 66843 2. The special locale object descriptor *LC_GLOBAL_LOCALE*
- 66844 3. **(locale_t)0**

66845 Once the *uselocale()* function has been called to install a thread-local locale, the behavior of every
66846 interface using data from the current locale shall be affected for the calling thread. The current
66847 locale for other threads shall remain unchanged.

66848 If the *newloc* argument is a null pointer, the object returned is the current locale or
66849 *LC_GLOBAL_LOCALE* if there has been no previous call to *uselocale()* for the current thread.

66850 If the *newloc* argument is *LC_GLOBAL_LOCALE*, the thread shall use the global locale
66851 determined by the *setlocale()* function.

66852 **RETURN VALUE**

66853 The *uselocale()* function returns the locale handle from the previous call for the current thread. If
66854 there was no such previous call, the function shall return the value *LC_GLOBAL_LOCALE*.

66855 **ERRORS**

66856 The *uselocale()* function may fail if:

66857 [EINVAL] *locale* is not a valid locale object.

66858 **EXAMPLES**

66859 None.

66860 **APPLICATION USAGE**

66861 Unlike the *setlocale()* function, the *uselocale()* function does not allow replacing some locale
66862 categories only. Applications that need to install a locale which differs only in a few categories
66863 must use *newlocale()* to change a locale object equivalent to the currently used locale and install
66864 it.

66865 **RATIONALE**

66866 None.

66867 **FUTURE DIRECTIONS**

66868 None.

66869 **SEE ALSO**

66870 *duplocale()*, *freelocale()*, *newlocale()*, *setlocale()*

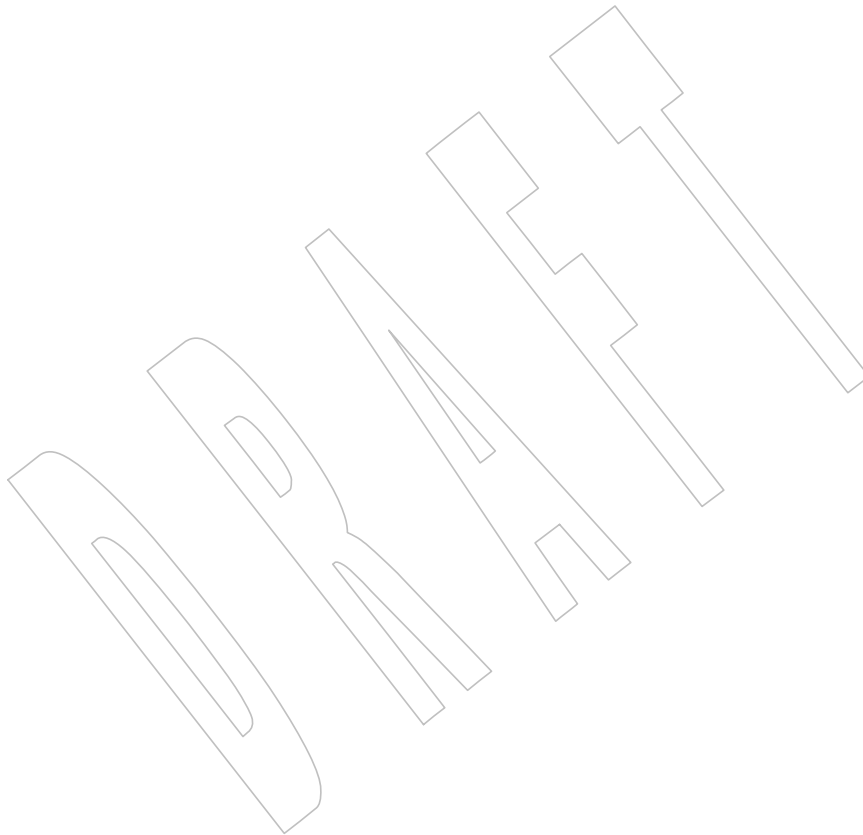
66871 XBD **<locale.h>**

66872

66873

CHANGE HISTORY

First released in Issue 7.



66874 **NAME**
 66875 `utime` — set file access and modification times

66876 **SYNOPSIS**

66877 OB

```
#include <utime.h>
```


 66878

```
int utime(const char *path, const struct utimbuf *times);
```

66879 **DESCRIPTION**

66880 The `utime()` function shall set the access and modification times of the file named by the `path`
 66881 argument.

66882 If `times` is a null pointer, the access and modification times of the file shall be set to the current
 66883 time. The effective user ID of the process shall match the owner of the file, or the process has
 66884 write permission to the file or has appropriate privileges, to use `utime()` in this manner.

66885 If `times` is not a null pointer, `times` shall be interpreted as a pointer to a **utimbuf** structure and the
 66886 access and modification times shall be set to the values contained in the designated structure.
 66887 Only a process with the effective user ID equal to the user ID of the file or a process with
 66888 appropriate privileges may use `utime()` this way.

66889 The **utimbuf** structure is defined in the `<utime.h>` header. The times in the structure **utimbuf**
 66890 are measured in seconds since the Epoch.

66891 Upon successful completion, the `utime()` function shall mark the last file status change
 66892 timestamp for update; see `<sys/stat.h>`.

66893 **RETURN VALUE**

66894 Upon successful completion, 0 shall be returned. Otherwise, `-1` shall be returned and `errno` shall
 66895 be set to indicate the error, and the file times shall not be affected.

66896 **ERRORS**

66897 The `utime()` function shall fail if:

66898 [EACCES] Search permission is denied by a component of the path prefix; or the `times`
 66899 argument is a null pointer and the effective user ID of the process does not
 66900 match the owner of the file, the process does not have write permission for the
 66901 file, and the process does not have appropriate privileges.

66902 [ELOOP] A loop exists in symbolic links encountered during resolution of the `path`
 66903 argument.

66904 [ENAMETOOLONG] The length of the `path` argument exceeds `{PATH_MAX}` or a pathname
 66905 component is longer than `{NAME_MAX}`.
 66906

66907 [ENOENT] A component of `path` does not name an existing file or `path` is an empty string.

66908 [ENOTDIR] A component of the path prefix is not a directory.

66909 [EPERM] The `times` argument is not a null pointer and the effective user ID of the calling
 66910 process does not match the owner of the file and the calling process does not
 66911 have the appropriate privileges.

66912 [EROFS] The file system containing the file is read-only.

66913 The `utime()` function may fail if:

66914 [ELOOP] More than {SYMLOOP_MAX} symbolic links were encountered during
66915 resolution of the *path* argument.

66916 [ENAMETOOLONG]
66917 As a result of encountering a symbolic link in resolution of the *path* argument,
66918 the length of the substituted pathname string exceeded {PATH_MAX}.

EXAMPLES

66919 None.
66920

APPLICATION USAGE

66921 Since the **utimbuf** structure only contains **time_t** variables and is not accurate to fractions of a
66922 second, applications should use the *utimensat()* function instead of the obsolescent *utime()*
66923 function.
66924

RATIONALE

66925 The *actime* structure member must be present so that an application may set it, even though an
66926 implementation may ignore it and not change the last data access timestamp on the file. If an
66927 application intends to leave one of the times of a file unchanged while changing the other, it
66928 should use *stat()* or *fstat()* to retrieve the file's *st_atim* and *st_mtim* parameters, set *actime* and
66929 *modtime* in the buffer, and change one of them before making the *utime()* call. +
66930

FUTURE DIRECTIONS

66931 The *utime()* function may be removed in a future version. |
66932

SEE ALSO

66933 *fstat()*, *fstatat()*, *futimens()*

66934 XBD <sys/stat.h>, <utime.h> |
66935

CHANGE HISTORY

66936 First released in Issue 1. Derived from Issue 1 of the SVID.
66937

Issue 6

66938 The following new requirements on POSIX implementations derive from alignment with the
66939 Single UNIX Specification:
66940

- 66941 • The requirement to include <sys/types.h> has been removed. Although <sys/types.h> was
66942 required for conforming implementations of previous POSIX specifications, it was not
66943 required for UNIX applications.
- 66944 • The [ELOOP] mandatory error condition is added.
- 66945 • A second [ENAMETOOLONG] is added as an optional error condition.

66946 The following changes were made to align with the IEEE P1003.1a draft standard:

- 66947 • The [ELOOP] optional error condition is added.

66948 The normative text is updated to avoid use of the term “must” for application requirements.

Issue 7

66949 The *utime()* function is marked obsolescent. +
66950

66951 Changes are made related to support for finegrained timestamps.

66952 **NAME**
 66953 utimensat, utimes — set file access and modification times relative to directory file descriptor |

66954 **SYNOPSIS**
 66955 #include <sys/stat.h> +
 66956 int utimensat(int *fd*, const char **path*, const struct timespec *times*[2], +
 66957 int *flag*); +

66958 XSI #include <sys/time.h>
 66959 int utimes(const char **path*, const struct timeval *times*[2]); |

66960 **DESCRIPTION**
 66961 Refer to *futimens()*. |

66962

66963 **NAME**

66964 `va_arg, va_copy, va_end, va_start` — handle variable argument list

66965 **SYNOPSIS**

66966 `#include <stdarg.h>`

66967 `type va_arg(va_list ap, type);`

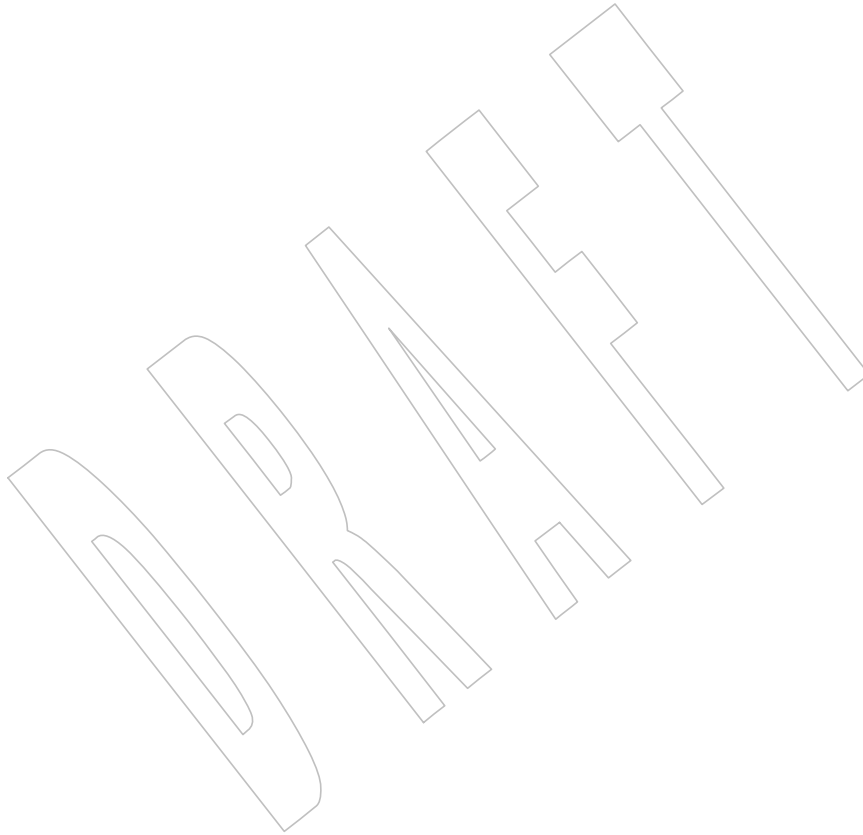
66968 `void va_copy(va_list dest, va_list src);`

66969 `void va_end(va_list ap);`

66970 `void va_start(va_list ap, argN);`

66971 **DESCRIPTION**

66972 Refer to the Base Definitions volume of POSIX.1-200x, `<stdarg.h>`.



66973 **NAME**

66974 vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

66975 **SYNOPSIS**

66976 #include <stdarg.h>

66977 #include <stdio.h>

66978 int vfprintf(FILE *restrict stream, const char *restrict format,
66979 va_list ap);

66980 int vprintf(const char *restrict format, va_list ap);

66981 int vsnprintf(char *restrict s, size_t n, const char *restrict format,
66982 va_list ap);

66983 int vsprintf(char *restrict s, const char *restrict format, va_list ap);

66984 **DESCRIPTION**66985 CX The functionality described on this reference page is aligned with the ISO C standard. Any
66986 conflict between the requirements described here and the ISO C standard is unintentional. This
66987 volume of POSIX.1-200x defers to the ISO C standard.66988 The *vprintf()*, *vfprintf()*, *vsnprintf()*, and *vsprintf()* functions shall be equivalent to *printf()*,
66989 *fprintf()*, *snprintf()*, and *sprintf()* respectively, except that instead of being called with a variable
66990 number of arguments, they are called with an argument list as defined by <stdarg.h>.66991 These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro,
66992 the value of *ap* after the return is unspecified.66993 **RETURN VALUE**66994 Refer to *fprintf()*.66995 **ERRORS**66996 Refer to *fprintf()*.66997 **EXAMPLES**

66998 None.

66999 **APPLICATION USAGE**67000 Applications using these functions should call *va_end(ap)* afterwards to clean up.67001 **RATIONALE**

67002 None.

67003 **FUTURE DIRECTIONS**

67004 None.

67005 **SEE ALSO**67006 *fprintf()*

67007 XBD <stdarg.h>, <stdio.h>

67008 **CHANGE HISTORY**

67009 First released in Issue 1. Derived from Issue 1 of the SVID.

67010 **Issue 5**67011 The *vsnprintf()* function is added.67012 **Issue 6**67013 The *vfprintf()*, *vprintf()*, *vsnprintf()*, and *vsprintf()* functions are updated for alignment with the
67014 ISO/IEC 9899:1999 standard.

67015 **NAME**67016 `vfscanf`, `vscanf`, `vsscanf` — format input of a `stdarg` argument list67017 **SYNOPSIS**67018 `#include <stdarg.h>`67019 `#include <stdio.h>`67020 `int vfscanf(FILE *restrict stream, const char *restrict format,`
67021 `va_list arg);`67022 `int vscanf(const char *restrict format, va_list arg);`67023 `int vsscanf(const char *restrict s, const char *restrict format,`
67024 `va_list arg);`67025 **DESCRIPTION**67026 CX The functionality described on this reference page is aligned with the ISO C standard. Any
67027 conflict between the requirements described here and the ISO C standard is unintentional. This
67028 volume of POSIX.1-200x defers to the ISO C standard.67029 The `vscanf()`, `vfscanf()`, and `vsscanf()` functions shall be equivalent to the `scanf()`, `fscanf()`, and
67030 `sscanf()` functions, respectively, except that instead of being called with a variable number of
67031 arguments, they are called with an argument list as defined in the `<stdarg.h>` header. These
67032 functions shall not invoke the `va_end` macro. As these functions invoke the `va_arg` macro, the
67033 value of `ap` after the return is unspecified.67034 **RETURN VALUE**67035 Refer to `fscanf()`.67036 **ERRORS**67037 Refer to `fscanf()`.67038 **EXAMPLES**

67039 None.

67040 **APPLICATION USAGE**67041 Applications using these functions should call `va_end(ap)` afterwards to clean up.67042 **RATIONALE**

67043 None.

67044 **FUTURE DIRECTIONS**

67045 None.

67046 **SEE ALSO**67047 `fscanf()`67048 XBD `<stdarg.h>`, `<stdio.h>`67049 **CHANGE HISTORY**

67050 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

67051 **NAME**
 67052 vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

67053 SYNOPSIS

```
67054 #include <stdarg.h>
67055 #include <stdio.h>
67056 #include <wchar.h>

67057 int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
67058             va_list arg);
67059 int vswprintf(wchar_t *restrict ws, size_t n,
67060             const wchar_t *restrict format, va_list arg);
67061 int vwprintf(const wchar_t *restrict format, va_list arg);
```

67062 DESCRIPTION

67063 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67064 conflict between the requirements described here and the ISO C standard is unintentional. This
 67065 volume of POSIX.1-200x defers to the ISO C standard.

67066 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* functions shall be equivalent to *fwprintf()*, *swprintf()*,
 67067 and *wprintf()* respectively, except that instead of being called with a variable number of
 67068 arguments, they are called with an argument list as defined by **<stdarg.h>**.

67069 These functions shall not invoke the *va_end* macro. However, as these functions do invoke the
 67070 *va_arg* macro, the value of *ap* after the return is unspecified.

67071 RETURN VALUE

67072 Refer to *fwprintf()*.

67073 ERRORS

67074 Refer to *fwprintf()*.

67075 EXAMPLES

67076 None.

67077 APPLICATION USAGE

67078 Applications using these functions should call *va_end(ap)* afterwards to clean up.

67079 RATIONALE

67080 None.

67081 FUTURE DIRECTIONS

67082 None.

67083 SEE ALSO

67084 *fwprintf()*
 67085 XBD **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

67086 CHANGE HISTORY

67087 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 67088 (E).

67089 Issue 6

67090 The *vfwprintf()*, *vswprintf()*, and *vwprintf()* prototypes are updated for alignment with the
 67091 ISO/IEC 9899:1999 standard. ()

67092 **NAME**

67093 vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

67094 **SYNOPSIS**

67095 #include <stdarg.h>

67096 #include <stdio.h>

67097 #include <wchar.h>

67098 int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
67099 va_list arg);67100 int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
67101 va_list arg);

67102 int vwscanf(const wchar_t *restrict format, va_list arg);

67103 **DESCRIPTION**67104 CX The functionality described on this reference page is aligned with the ISO C standard. Any
67105 conflict between the requirements described here and the ISO C standard is unintentional. This
67106 volume of POSIX.1-200x defers to the ISO C standard.67107 The *vfwscanf()*, *vswscanf()*, and *vwscanf()* functions shall be equivalent to the *fscanf()*,
67108 *swscanf()*, and *wscanf()* functions, respectively, except that instead of being called with a variable
67109 number of arguments, they are called with an argument list as defined in the <stdarg.h> header.
67110 These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro,
67111 the value of *ap* after the return is unspecified.67112 **RETURN VALUE**67113 Refer to *fscanf()*.67114 **ERRORS**67115 Refer to *fscanf()*.67116 **EXAMPLES**

67117 None.

67118 **APPLICATION USAGE**67119 Applications using these functions should call *va_end(ap)* afterwards to clean up.67120 **RATIONALE**

67121 None.

67122 **FUTURE DIRECTIONS**

67123 None.

67124 **SEE ALSO**67125 *fscanf()*

67126 XBD <stdarg.h>, <stdio.h>, <wchar.h>

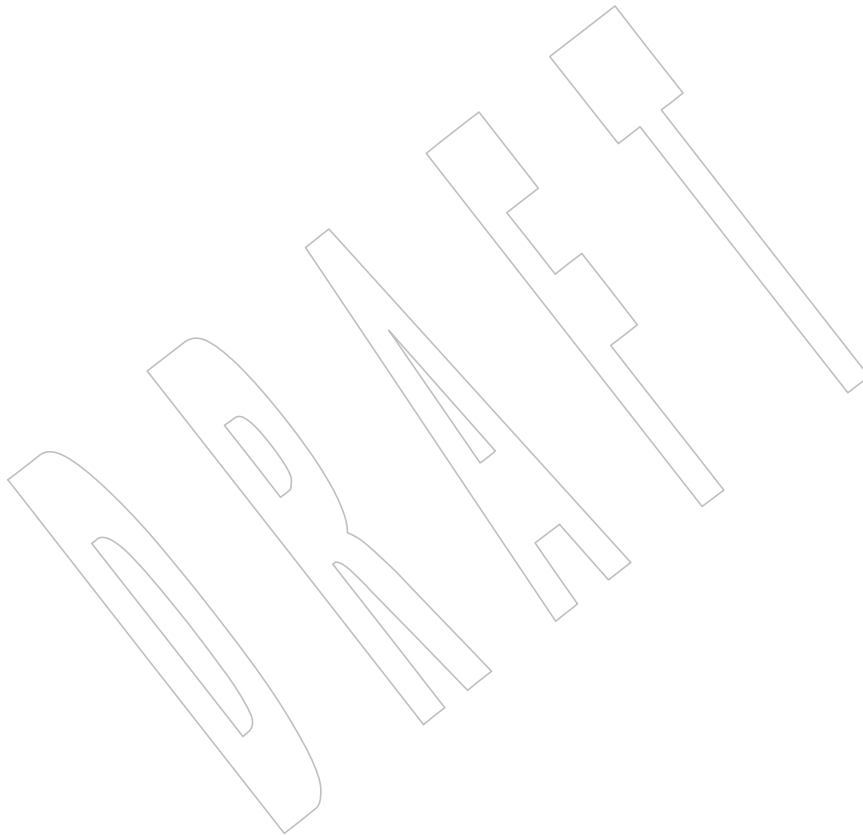
67127 **CHANGE HISTORY**

67128 First released in Issue 6. Derived from the ISO/IEC 9899:1999 standard.

67129 **NAME**
67130 `vprintf` — format the output of a `stdarg` argument list

67131 **SYNOPSIS**
67132 `#include <stdarg.h>`
67133 `#include <stdio.h>`
67134 `int vprintf(const char *restrict format, va_list ap);`

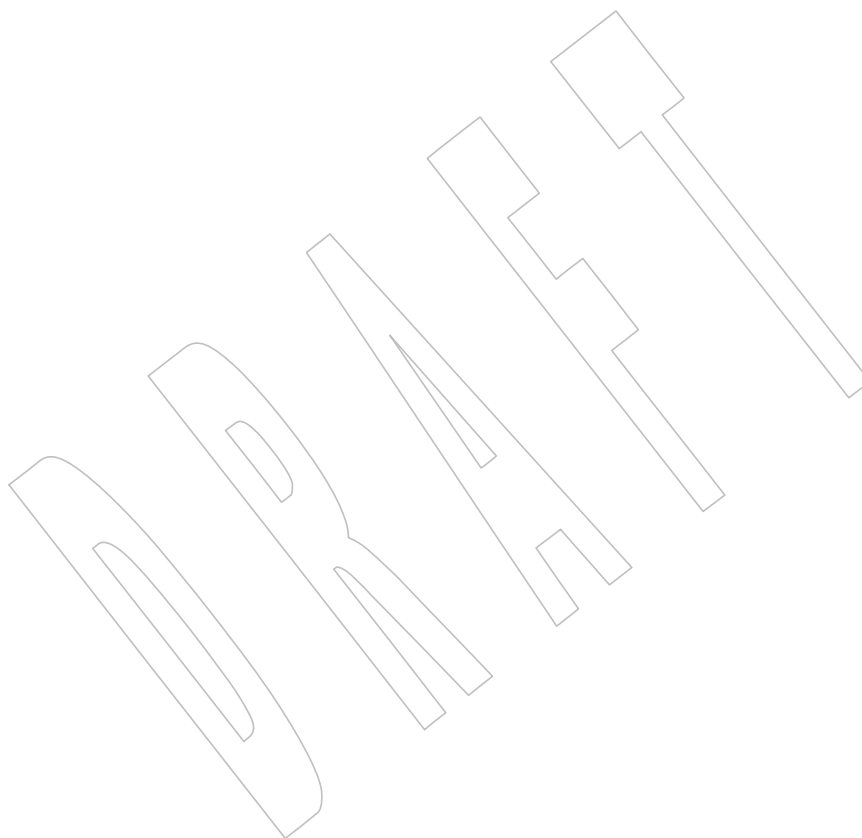
67135 **DESCRIPTION**
67136 Refer to *vfprintf()*.



67137 **NAME**
67138 vscanf — format input of a stdarg argument list

67139 **SYNOPSIS**
67140 #include <stdarg.h>
67141 #include <stdio.h>
67142 int vscanf(const char *restrict format, va_list arg);

67143 **DESCRIPTION**
67144 Refer to *vfscanf()*.



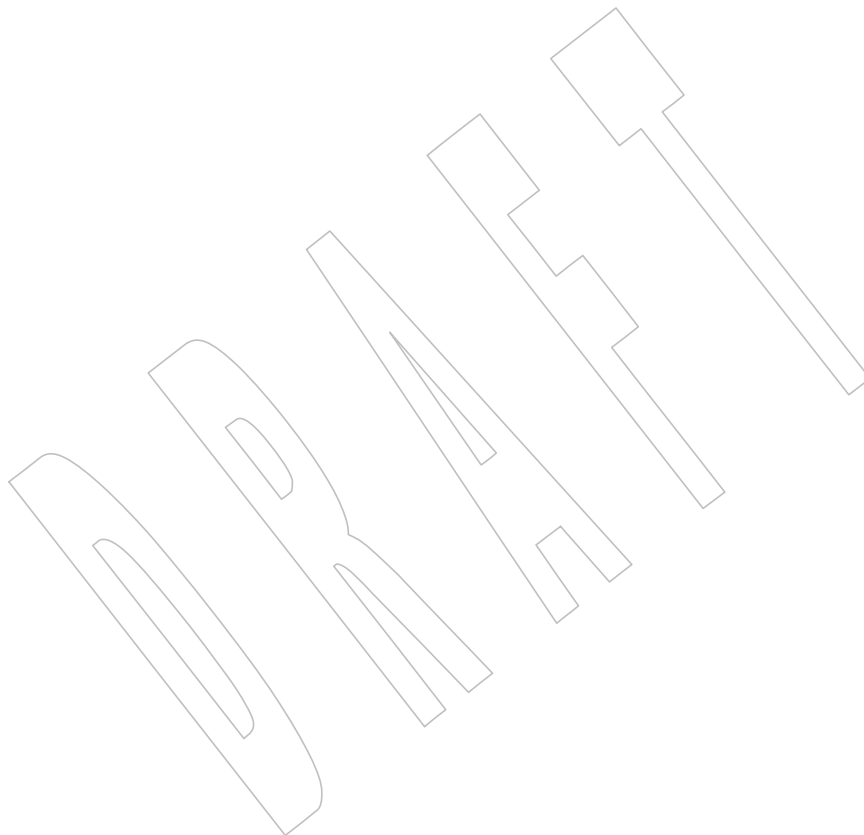
67145 **NAME**
67146 `vsnprintf`, `vsprintf` — format output of a stdarg argument list

67147 **SYNOPSIS**

```
67148 #include <stdarg.h>  
67149 #include <stdio.h>  
  
67150 int vsnprintf(char *restrict s, size_t n,  
67151             const char *restrict format, va_list ap);  
67152 int vsprintf(char *restrict s, const char *restrict format,  
67153            va_list ap);
```

67154 **DESCRIPTION**

67155 Refer to *fprintf()*.



67156 **NAME**
67157 vsscanf — format input of a stdarg argument list

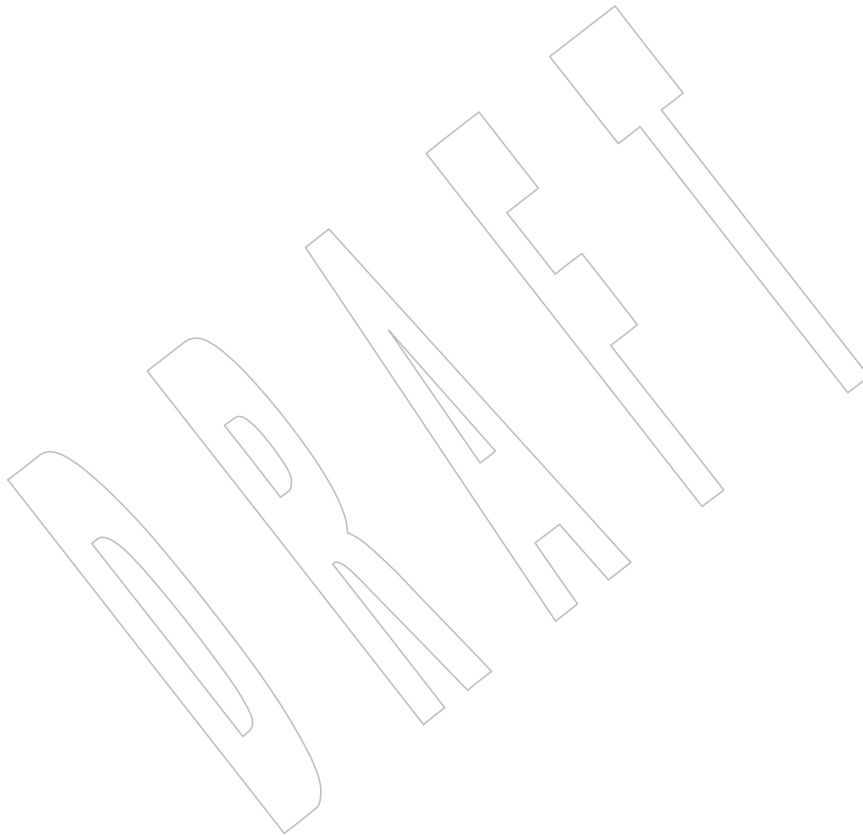
67158 **SYNOPSIS**

67159 #include <stdarg.h>
67160 #include <stdio.h>

67161 int vsscanf(const char *restrict *s*, const char *restrict *format*,
67162 va_list *arg*);

67163 **DESCRIPTION**

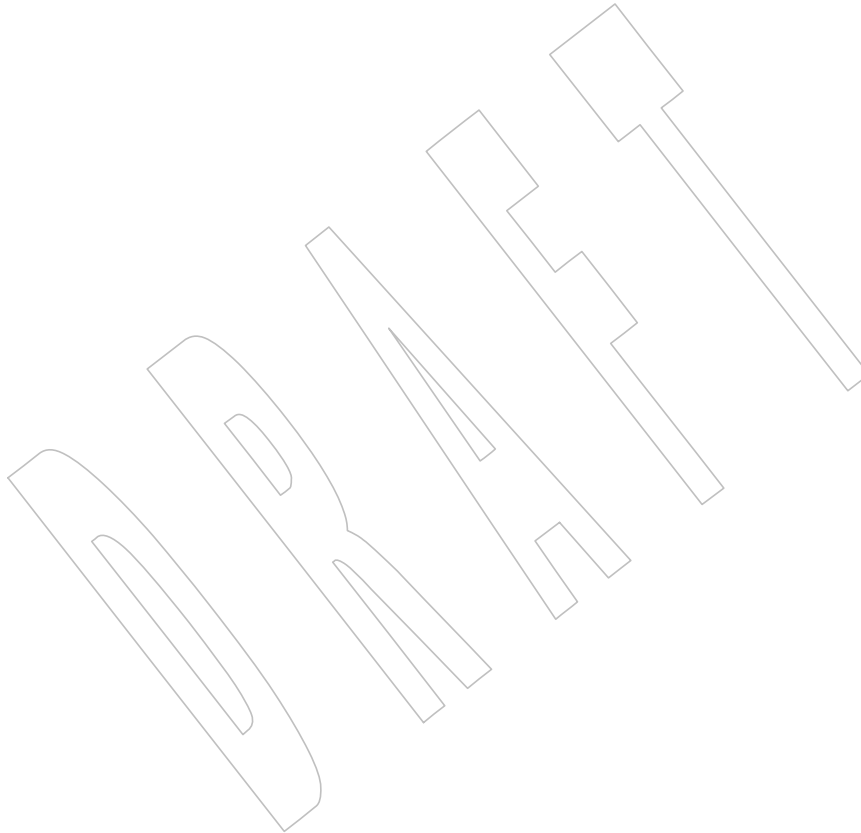
67164 Refer to *vfscanf()*.



67165 **NAME**
67166 `vswprintf` — wide-character formatted output of a `stdarg` argument list

67167 **SYNOPSIS**
67168 `#include <stdarg.h>`
67169 `#include <stdio.h>`
67170 `#include <wchar.h>`
67171 `int vswprintf(wchar_t *restrict ws, size_t n,`
67172 `const wchar_t *restrict format, va_list arg);`

67173 **DESCRIPTION**
67174 Refer to *[vfwprintf\(\)](#)*.



67175 **NAME**
67176 vswscanf — wide-character formatted input of a stdarg argument list

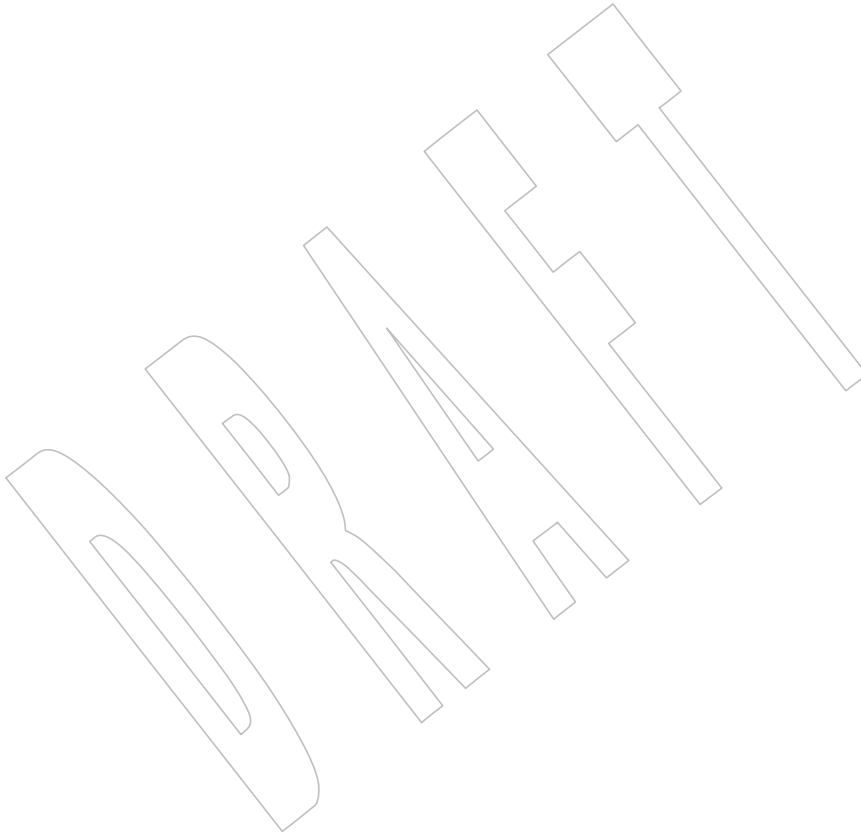
67177 **SYNOPSIS**

67178 #include <stdarg.h>
67179 #include <stdio.h>
67180 #include <wchar.h>

67181 int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
67182 va_list arg);

67183 **DESCRIPTION**

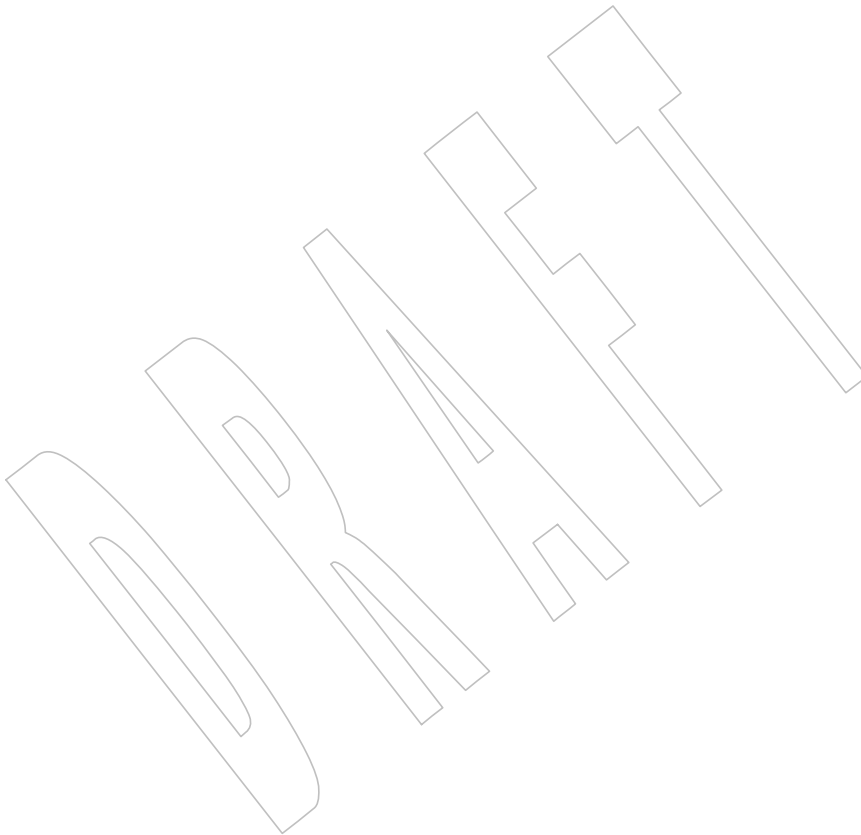
67184 Refer to *vfwscanf()*.



67185 **NAME**
67186 `vwprintf` — wide-character formatted output of a `stdarg` argument list

67187 **SYNOPSIS**
67188 `#include <stdarg.h>`
67189 `#include <stdio.h>`
67190 `#include <wchar.h>`
67191 `int vwprintf(const wchar_t *restrict format, va_list arg);`

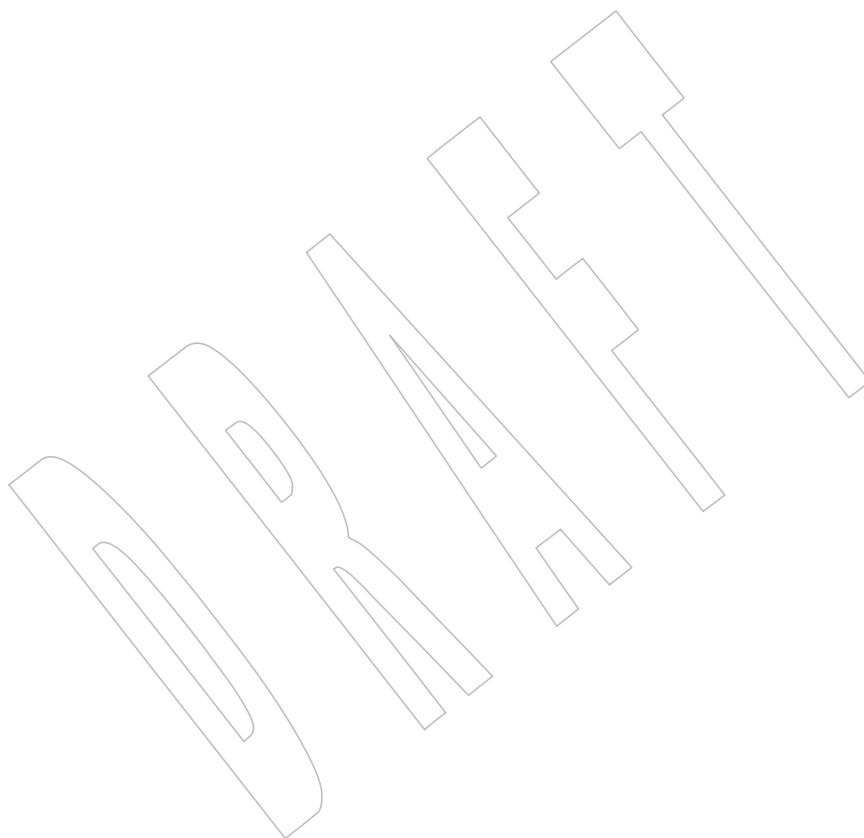
67192 **DESCRIPTION**
67193 Refer to *[vfwprintf\(\)](#)*.



67194 **NAME**
67195 vwscanf — wide-character formatted input of a stdarg argument list

67196 **SYNOPSIS**
67197 #include <stdarg.h>
67198 #include <stdio.h>
67199 #include <wchar.h>
67200 int vwscanf(const wchar_t *restrict format, va_list arg);

67201 **DESCRIPTION**
67202 Refer to *vwscanf()*.



67203 **NAME**
 67204 wait, waitpid — wait for a child process to stop or terminate

67205 **SYNOPSIS**
 67206 #include <sys/wait.h>
 67207 pid_t wait(int *stat_loc);
 67208 pid_t waitpid(pid_t pid, int *stat_loc, int options);

67209 **DESCRIPTION**
 67210 The *wait()* and *waitpid()* functions shall obtain status information pertaining to one of the
 67211 caller's child processes. Various options permit status information to be obtained for child
 67212 processes that have terminated or stopped. If status information is available for two or more
 67213 child processes, the order in which their status is reported is unspecified.

67214 The *wait()* function shall suspend execution of the calling thread until status information for one
 67215 of the terminated child processes of the calling process is available, or until delivery of a signal
 67216 whose action is either to execute a signal-catching function or to terminate the process. If more
 67217 than one thread is suspended in *wait()* or *waitpid()* awaiting termination of the same process,
 67218 exactly one thread shall return the process status at the time of the target process termination. If
 67219 status information is available prior to the call to *wait()*, return shall be immediate.

67220 The *waitpid()* function shall be equivalent to *wait()* if the *pid* argument is **(pid_t)-1** and the
 67221 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and
 67222 *options* arguments.

67223 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid()*
 67224 function shall only return the status of a child process from this set:

- 67225 • If *pid* is equal to **(pid_t)-1**, *status* is requested for any child process. In this respect,
 67226 *waitpid()* is then equivalent to *wait()*.
- 67227 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is
 67228 requested.
- 67229 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that
 67230 of the calling process.
- 67231 • If *pid* is less than **(pid_t)-1**, *status* is requested for any child process whose process group
 67232 ID is equal to the absolute value of *pid*.

67233 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the
 67234 following flags, defined in the **<sys/wait.h>** header:

67235 XSI **WCONTINUED** The *waitpid()* function shall report the status of any continued child process
 67236 specified by *pid* whose status has not been reported since it continued from a
 67237 job control stop.

67238 **WNOHANG** The *waitpid()* function shall not suspend execution of the calling thread if
 67239 *status* is not immediately available for one of the child processes specified by
 67240 *pid*.

67241 **WUNTRACED** The status of any child processes specified by *pid* that are stopped, and whose
 67242 status has not yet been reported since they stopped, shall also be reported to
 67243 the requesting process.

67244 XSI If the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, and the process
 67245 has no unwaited-for children that were transformed into zombie processes, the calling thread
 67246 shall block until all of the children of the process containing the calling thread terminate, and

67247		<code>wait()</code> and <code>waitpid()</code> shall fail and set <code>errno</code> to [ECHILD].
67248 67249 67250 67251 67252		If <code>wait()</code> or <code>waitpid()</code> return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument <code>stat_loc</code> is not a null pointer, information shall be stored in the location pointed to by <code>stat_loc</code> . The value stored at the location pointed to by <code>stat_loc</code> shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:
67253 67254 67255		<ol style="list-style-type: none"> 1. The process returned 0 from <code>main()</code>. 2. The process called <code>_exit()</code> or <code>exit()</code> with a <code>status</code> argument of 0. 3. The process was terminated because the last thread in the process terminated.
67256 67257 67258		Regardless of its value, this information may be interpreted using the following macros, which are defined in <code><sys/wait.h></code> and evaluate to integral expressions; the <code>stat_val</code> argument is the integer value pointed to by <code>stat_loc</code> .
67259 67260 67261		WIFEXITED(<i>stat_val</i>) Evaluates to a non-zero value if <code>status</code> was returned for a child process that terminated normally.
67262 67263 67264 67265		WEXITSTATUS(<i>stat_val</i>) If the value of WIFEXITED(<i>stat_val</i>) is non-zero, this macro evaluates to the low-order 8 bits of the <code>status</code> argument that the child process passed to <code>_exit()</code> or <code>exit()</code> , or the value the child process returned from <code>main()</code> .
67266 67267 67268		WIFSIGNALED(<i>stat_val</i>) Evaluates to a non-zero value if <code>status</code> was returned for a child process that terminated due to the receipt of a signal that was not caught (see <code><signal.h></code>).
67269 67270 67271		WTERMSIG(<i>stat_val</i>) If the value of WIFSIGNALED(<i>stat_val</i>) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.
67272 67273 67274		WIFSTOPPED(<i>stat_val</i>) Evaluates to a non-zero value if <code>status</code> was returned for a child process that is currently stopped.
67275 67276 67277		WSTOPSIG(<i>stat_val</i>) If the value of WIFSTOPPED(<i>stat_val</i>) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.
67278 67279 67280	XSI	WIFCONTINUED(<i>stat_val</i>) Evaluates to a non-zero value if <code>status</code> was returned for a child process that has continued from a job control stop.
67281 67282 67283 67284	SPN	It is unspecified whether the <code>status</code> value returned by calls to <code>wait()</code> or <code>waitpid()</code> for processes created by <code>posix_spawn()</code> or <code>posix_spawnnp()</code> can indicate a WIFSTOPPED(<i>stat_val</i>) before subsequent calls to <code>wait()</code> or <code>waitpid()</code> indicate WIFEXITED(<i>stat_val</i>) as the result of an error detected before the new process image starts executing.
67285 67286 67287		It is unspecified whether the <code>status</code> value returned by calls to <code>wait()</code> or <code>waitpid()</code> for processes created by <code>posix_spawn()</code> or <code>posix_spawnnp()</code> can indicate a WIFSIGNALED(<i>stat_val</i>) if a signal is sent to the parent's process group after <code>posix_spawn()</code> or <code>posix_spawnnp()</code> is called.
67288 67289 67290 67291	XSI	If the information pointed to by <code>stat_loc</code> was stored by a call to <code>waitpid()</code> that specified the WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(*<i>stat_loc</i>) , WIFSIGNALED(*<i>stat_loc</i>) , and WIFSTOPPED(*<i>stat_loc</i>) shall evaluate to a non-zero value.
67292		If the information pointed to by <code>stat_loc</code> was stored by a call to <code>waitpid()</code> that specified the

wait()

67293 XSI WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(*stat_loc),
 67294 XSI WIFSIGNALED(*stat_loc), WIFSTOPPED(*stat_loc), and WIFCONTINUED(*stat_loc) shall
 67295 evaluate to a non-zero value.

67296 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 67297 XSI WUNTRACED or WCONTINUED flags, or by a call to the *wait()* function, exactly one of the
 67298 macros WIFEXITED(*stat_loc) and WIFSIGNALED(*stat_loc) shall evaluate to a non-zero value.

67299 If the information pointed to by *stat_loc* was stored by a call to *waitpid()* that did not specify the
 67300 XSI WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait()* function,
 67301 XSI exactly one of the macros WIFEXITED(*stat_loc), WIFSIGNALED(*stat_loc), and
 67302 WIFCONTINUED(*stat_loc) shall evaluate to a non-zero value.

67303 If `_POSIX_REALTIME_SIGNALS` is defined, and the implementation queues the SIGCHLD
 67304 signal, then if *wait()* or *waitpid()* returns because the status of a child process is available, any
 67305 pending SIGCHLD signal associated with the process ID of the child process shall be discarded.
 67306 Any other pending SIGCHLD signals shall remain pending.

67307 Otherwise, if SIGCHLD is blocked, if *wait()* or *waitpid()* return because the status of a child
 67308 process is available, any pending SIGCHLD signal shall be cleared unless the status of another
 67309 child process is available.

67310 For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD
 67311 signal is delivered.

67312 There may be additional implementation-defined circumstances under which *wait()* or *waitpid()*
 67313 report *status*. This shall not occur unless the calling process or one of its child processes
 67314 explicitly makes use of a non-standard extension. In these cases the interpretation of the
 67315 reported *status* is implementation-defined.

67316 If a parent process terminates without waiting for all of its child processes to terminate, the
 67317 remaining child processes shall be assigned a new parent process ID corresponding to an
 67318 implementation-defined system process.

RETURN VALUE

67319 If *wait()* or *waitpid()* returns because the status of a child process is available, these functions
 67320 shall return a value equal to the process ID of the child process for which *status* is reported. If
 67321 *wait()* or *waitpid()* returns due to the delivery of a signal to the calling process, `-1` shall be
 67322 returned and *errno* set to [EINTR]. If *waitpid()* was invoked with WNOHANG set in *options*, it
 67323 has at least one child process specified by *pid* for which *status* is not available, and *status* is not
 67324 available for any process specified by *pid*, `0` is returned. Otherwise, `(pid_t)-1` shall be returned,
 67325 and *errno* set to indicate the error.
 67326

ERRORS

67327 The *wait()* function shall fail if:

67328 [ECHILD] The calling process has no existing unwaited-for child processes.

67330 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 67331 by *stat_loc* is undefined.

67332 The *waitpid()* function shall fail if:

67333 [ECHILD] The process specified by *pid* does not exist or is not a child of the calling
 67334 process, or the process group specified by *pid* does not exist or does not have
 67335 any member process that is a child of the calling process.

67336 [EINTR] The function was interrupted by a signal. The value of the location pointed to
 67337 by *stat_loc* is undefined.

67338 [EINVAL] The *options* argument is not valid.

67339 EXAMPLES

67340 Waiting for a Child Process and then Checking its Status

67341 The following example demonstrates the use of *waitpid()*, *fork()*, and the macros used to
 67342 interpret the status value returned by *waitpid()* (and *wait()*). The code segment creates a child
 67343 process which does some unspecified work. Meanwhile the parent loops performing calls to
 67344 *waitpid()* to monitor the status of the child. The loop terminates when child termination is
 67345 detected.

```

67346 #include <stdio.h>
67347 #include <stdlib.h>
67348 #include <unistd.h>
67349 #include <sys/wait.h>
67350 ...
67351 pid_t child_pid, wpid;
67352 int status;
67353
67354 child_pid = fork();
67355 if (child_pid == -1) {          /* fork() failed */
67356     perror("fork");
67357     exit(EXIT_FAILURE);
67358 }
67359 if (child_pid == 0) {          /* This is the child */
67360     /* Child does some work and then terminates */
67361     ...
67362 } else {                        /* This is the parent */
67363     do {
67364         wpid = waitpid(child_pid, &status, WUNTRACED
67365 #ifdef WCONTINUED          /* Not all implementations support this */
67366         | WCONTINUED
67367 #endif
67368         );
67369         if (wpid == -1) {
67370             perror("waitpid");
67371             exit(EXIT_FAILURE);
67372         }
67373         if (WIFEXITED(status)) {
67374             printf("child exited, status=%d\n", WEXITSTATUS(status));
67375         } else if (WIFSIGNALED(status)) {
67376             printf("child killed (signal %d)\n", WTERMSIG(status));
67377         } else if (WIFSTOPPED(status)) {
67378             printf("child stopped (signal %d)\n", WSTOPSIG(status));
67379         }
67380 #ifdef WIFCONTINUED          /* Not all implementations support this */
67381         } else if (WIFCONTINUED(status)) {
67382             printf("child continued\n");
67383         }
67384 #endif
67385     } else {                    /* Non-standard case -- may never happen */
67386         printf("Unexpected status (0x%x)\n", status);
67387     }
67388     } while (!WIFEXITED(status) && !WIFSIGNALED(status));

```

```

67386     }
67387
67387     Waiting for a Child Process in a Signal Handler for SIGCHLD
67388     The following example demonstrates how to use waitpid() in a signal handler for SIGCHLD +
67389     without passing -1 as the pid argument. (See the APPLICATION USAGE section below for the +
67390     reasons why passing a pid of -1 is not recommended.) The method used here relies on the +
67391     standard behavior of waitpid() when SIGCHLD is blocked. On historical non-conforming +
67392     systems, the status of some child processes might not be reported. +
67393
67393     #include <stdlib.h> +
67394     #include <stdio.h> +
67395     #include <signal.h> +
67396     #include <sys/types.h> +
67397     #include <sys/wait.h> +
67398     #include <unistd.h> +
67399
67399     #define CHILDREN 10 +
67400
67400     static void +
67401     handle_sigchld(int signum, siginfo_t *sinfo, void *unused) +
67402     { +
67403         int status; +
67404
67404         /* +
67405          * Obtain status information for the child which +
67406          * caused the SIGCHLD signal and write its exit code +
67407          * to stdout. +
67408          */ +
67409         if (sinfo->si_code != CLD_EXITED) +
67410         { +
67411             static char msg[] = "wrong si_code\n"; +
67412             write(2, msg, sizeof msg - 1); +
67413         } +
67414         else if (waitpid(sinfo->si_pid, &status, 0) == -1) +
67415         { +
67416             static char msg[] = "waitpid() failed\n"; +
67417             write(2, msg, sizeof msg - 1); +
67418         } +
67419         else if (!WIFEXITED(status)) +
67420         { +
67421             static char msg[] = "WIFEXITED was false\n"; +
67422             write(2, msg, sizeof msg - 1); +
67423         } +
67424         else +
67425         { +
67426             int code = WEXITSTATUS(status); +
67427             char buf[2]; +
67428             buf[0] = '0' + code; +
67429             buf[1] = '\n'; +
67430             write(1, buf, 2); +
67431         } +
67432     } +
67433
67433     int +
67434     main(void) +
67435     { +

```

```

67436     int i;                                     +
67437     pid_t pid;                                 +
67438     struct sigaction sa;                       +
67439         sa.sa_flags = SA_SIGINFO;             +
67440         sa.sa_sigaction = handle_sigchld;     +
67441         sigemptyset(&sa.sa_mask);            +
67442         if (sigaction(SIGCHLD, &sa, NULL) == -1) +
67443     {                                           +
67444         perror("sigaction");                 +
67445         exit(EXIT_FAILURE);                  +
67446     }                                           +
67447     for (i = 0; i < CHILDREN; i++)            +
67448     {                                           +
67449         switch (pid = fork())                 +
67450         {                                       +
67451             case -1:                           +
67452                 perror("fork");               +
67453                 exit(EXIT_FAILURE);           +
67454             case 0:                             +
67455                 sleep(2);                      +
67456                 _exit(i);                     +
67457         }                                       +
67458     }                                           +
67459     /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */ +
67460     alarm(3);                                   +
67461     for (;;)                                    +
67462         pause();                               +
67463 }                                               +

```

APPLICATION USAGE

Calls to *wait()* will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system()*). For this and other reasons it is recommended that portable applications not use *wait()*, but instead use *waitpid()*. For these same reasons, the use of *waitpid()* with a *pid* argument of *-1*, and the use of *waitid()* with the *idtype* argument set to *P_ALL*, are also not recommended for portable applications.

RATIONALE

A call to the *wait()* or *waitpid()* function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork()* call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork()*, none of those grandchildren nor any of their descendants affect the behavior of a *wait()* from the original parent process. Nothing in this volume of POSIX.1-200x prevents an implementation from providing extensions that permit a process to get *status* from a grandchild or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid()* function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait()* function
3. To permit a library routine, such as *system()* or *pclose()*, to wait for its children without interfering with other terminated children for which the process has not waited

67485 The first two of these facilities are based on the *wait3()* function provided by 4.3 BSD. The
 67486 function uses the *options* argument, which is equivalent to an argument to *wait3()*. The
 67487 WUNTRACED flag is used only in conjunction with job control on systems supporting job
 67488 control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped
 67489 processes in that implementation: processes being traced via the *ptrace()* debugging facility and
 67490 (untraced) processes stopped by job control signals. Since *ptrace()* is not part of this volume of
 67491 POSIX.1-200x, only the second type is relevant. The name WUNTRACED was retained because
 67492 its usage is the same, even though the name is not intuitively meaningful in this context.

67493 The third reason for the *waitpid()* function is to permit independent sections of a process to
 67494 spawn and wait for children without interfering with each other. For example, the following
 67495 problem occurs in developing a portable shell, or command interpreter:

```
67496 stream = popen("/bin/true");
67497 (void) system("sleep 100");
67498 (void) pclose(stream);
```

67499 On all historical implementations, the final *pclose()* fails to reap the *wait()* status of the *popen()*.

67500 The status values are retrieved by macros, rather than given as specific bit encodings as they are
 67501 in most historical implementations (and thus expected by existing programs). This was
 67502 necessary to eliminate a limitation on the number of signals an implementation can support that
 67503 was inherent in the traditional encodings. This volume of POSIX.1-200x does require that a *status*
 67504 value of zero corresponds to a process calling *_exit(0)*, as this is the most common encoding
 67505 expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

67506 These macros syntactically operate on an arbitrary integer value. The behavior is undefined
 67507 unless that value is one stored by a successful call to *wait()* or *waitpid()* in the location pointed to
 67508 by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each
 67509 argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other
 67510 specifications in this volume of POSIX.1-200x or traditional usage. It also could have implied
 67511 that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or
 67512 passed as an argument to other functions before being interpreted by these macros.

67513 The extension that affects *wait()* and *waitpid()* and is common in historical implementations is
 67514 the *ptrace()* function. It is called by a child process and causes that child to stop and return a
 67515 *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace()*
 67516 children is traditionally returned regardless of the WUNTRACED flag (or by the *wait()*
 67517 function). Most applications do not need to concern themselves with such extensions because
 67518 they have control over what extensions they or their children use. However, applications, such
 67519 as command interpreters, that invoke arbitrary processes may see this behavior when those
 67520 arbitrary processes misuse such extensions.

67521 Implementations that support **core** file creation or other implementation-defined actions on
 67522 termination of some processes traditionally provide a bit in the *status* returned by *wait()* to
 67523 indicate that such actions have occurred.

67524 Allowing the *wait()* family of functions to discard a pending SIGCHLD signal that is associated
 67525 with a successfully waited-for child process puts them into the *sigwait()* and *sigwaitinfo()*
 67526 category with respect to SIGCHLD.

67527 This definition allows implementations to treat a pending SIGCHLD signal as accepted by the
 67528 process in *wait()*, with the same meaning of “accepted” as when that word is applied to the
 67529 *sigwait()* family of functions.

67530 Allowing the *wait()* family of functions to behave this way permits an implementation to be able
 67531 to deal precisely with SIGCHLD signals.

67532 In particular, an implementation that does accept (discard) the SIGCHLD signal can make the
 67533 following guarantees regardless of the queuing depth of signals in general (the list of waitable

67534

children can hold the SIGCHLD queue):

67535

1. If a SIGCHLD signal handler is established via *sigaction()* without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.

67536

67537

67538

67539

2. A single *wait()* issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.

67540

67541

67542

3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-NULL pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid()* or *waitid()* will not block or fail.

67543

67544

67545

67546

67547

4. The *system()* function will not cause the SIGCHLD handler of a process to be called as a result of the *fork()/exec* executed within *system()* because *system()* will accept the SIGCHLD signal when it performs a *waitpid()* for its child process. This is a desirable behavior of *system()* so that it can be used in a library without causing side effects to the application linked with the library.

67548

67549

67550

An implementation that does not permit the *wait()* family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

67551

67552

67553

67554

67555

67556

67557

Guarantee #1

Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork()/wait()* operations. If the implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

67558

67559

67560

67561

67562

Guarantees #2 and #3

These cannot be guaranteed unless the *wait()* family of functions accepts the SIGCHLD signal. Otherwise, a *fork()/wait()* executed while SIGCHLD is blocked (as in the *system()* function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

67563

67564

67565

67566

67567

67568

Guarantee #4

Although possible to make this guarantee, *system()* would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork()* would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side effect of discarding all SIGCHLD signals pending to the process.

67569

FUTURE DIRECTIONS

67570

None.

67571

SEE ALSO

67572

exec, *exit()*, *fork()*, *system()*, *waitid()*

67573

XBD Section 4.11 (on page 98), [<signal.h>](#), [<sys/wait.h>](#)

67574

CHANGE HISTORY

67575

First released in Issue 1. Derived from Issue 1 of the SVID.

67576

Issue 5

67577

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

67578

Issue 6

67579

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

67580

- The requirement to include `<sys/types.h>` has been removed. Although `<sys/types.h>` was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

67581

67582

67583

67584

The following changes were made to align with the IEEE P1003.1a draft standard:

67585

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

67586

The semantics of `WIFSTOPPED(stat_val)`, `WIFEXITED(stat_val)`, and `WIFSIGNALED(stat_val)` are defined with respect to `posix_spawn()` or `posix_spawnnp()` for alignment with IEEE Std 1003.1d-1999.

67587

67588

67589

The DESCRIPTION is updated for alignment with the ISO/IEC 9899:1999 standard.

67590

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the EXAMPLES section.

67591

67592

Issue 7

67593

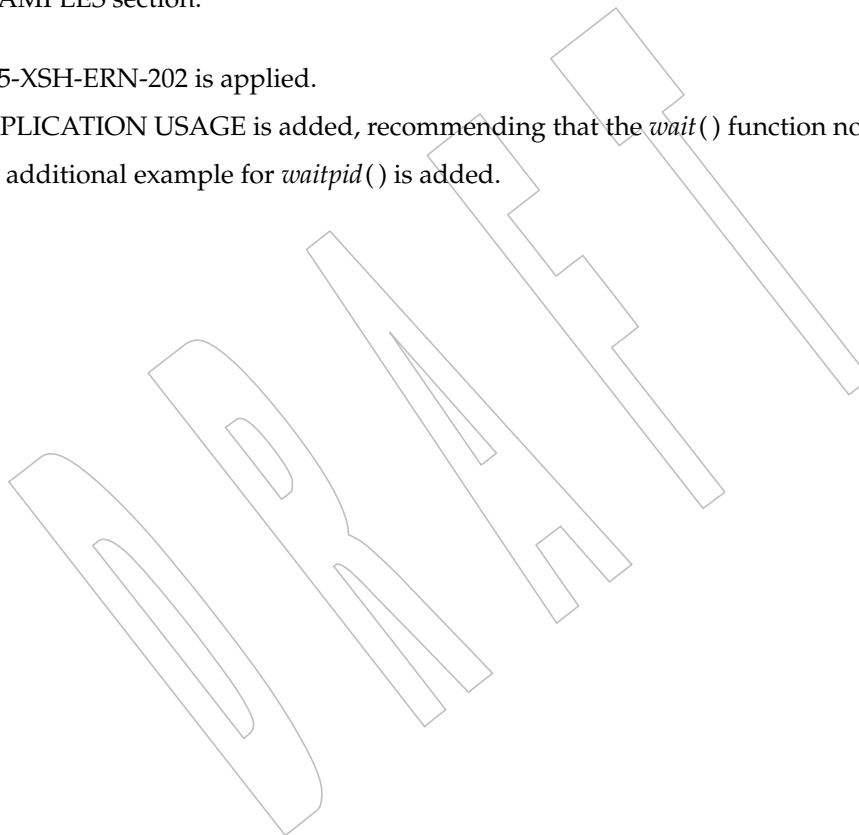
SD5-XSH-ERN-202 is applied. +

67594

APPLICATION USAGE is added, recommending that the `wait()` function not be used. +

67595

An additional example for `waitpid()` is added.



67596 **NAME**
 67597 waitid — wait for a child process to change state

67598 **SYNOPSIS**
 67599 #include <sys/wait.h>

67600 int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);

67601 **DESCRIPTION**

67602 The *waitid()* function shall suspend the calling thread until one child of the process containing
 67603 the calling thread changes state. It records the current state of a child in the structure pointed to
 67604 by *infop*. The fields of the structure pointed to by *infop* are filled in as described for the
 67605 SIGCHLD signal in <signal.h>. If a child process changed state prior to the call to *waitid()*,
 67606 *waitid()* shall return immediately. If more than one thread is suspended in *wait()*, *waitid()*, or
 67607 *waitpid()* waiting for termination of the same process, exactly one thread shall return the process
 67608 status at the time of the target process termination.

67609 The *idtype* and *id* arguments are used to specify which children *waitid()* waits for.

67610 If *idtype* is P_PID, *waitid()* shall wait for the child with a process ID equal to (**pid_t**)*id*.

67611 If *idtype* is P_PGID, *waitid()* shall wait for any child with a process group ID equal to (**pid_t**)*id*.

67612 If *idtype* is P_ALL, *waitid()* shall wait for any children and *id* is ignored.

67613 The *options* argument is used to specify which state changes *waitid()* shall wait for. It is formed
 67614 by OR'ing together the following flags:

67615 WEXITED Wait for processes that have exited.

67616 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

67617 WCONTINUED Status shall be returned for any child that was stopped and has been
 67618 continued.

67619 WNOHANG Return immediately if there are no children to wait for.

67620 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This
 67621 shall not affect the state of the process; the process may be waited for again
 67622 after this call completes.

67623 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to
 67624 be OR'd in with the *options* argument.

67625 The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid()*
 67626 returns because a child process was found that satisfied the conditions indicated by the
 67627 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the
 67628 system with the status of the process. The *si_signo* member shall always be equal to SIGCHLD.

67629 **RETURN VALUE**

67630 If WNOHANG was specified and there are no children to wait for, 0 shall be returned. If *waitid()*
 67631 returns due to the change of state of one of its children, 0 shall be returned. Otherwise, -1 shall
 67632 be returned and *errno* set to indicate the error.

67633 **ERRORS**

67634 The *waitid()* function shall fail if:

67635 [ECHILD] The calling process has no existing unwaited-for child processes.

waitid()

67636	[EINTR]	The <i>waitid()</i> function was interrupted by a signal.
67637	[EINVAL]	An invalid value was specified for <i>options</i> , or <i>idtype</i> and <i>id</i> specify an invalid set of processes.
67638		

EXAMPLES

67639 None.
67640

APPLICATION USAGE

67641 Calls to *waitid()* with *idtype* equal to P_ALL will collect information about any child process. |
67642 This may result in interactions with other interfaces that may be waiting for their own children |
67643 (such as by use of *system()*). For this reason it is recommended that portable applications not |
67644 use *waitid()* with *idtype* of P_ALL. See also APPLICATION USAGE for *wait()*. |
67645

RATIONALE

67646 None.
67647

FUTURE DIRECTIONS

67648 None.
67649

SEE ALSO

67650 *exec*, *exit()*, *wait*
67651

67652 XBD [<signal.h>](#), [<sys/wait.h>](#) |

CHANGE HISTORY

67653 First released in Issue 4, Version 2.
67654

Issue 5

67655 Moved from X/OPEN UNIX extension to BASE.
67656

67657 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

67658 The normative text is updated to avoid use of the term “must” for application requirements.
67659

Issue 7

67660 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.
67661

67662 The *waitid()* function is moved from the XSI option to the Base.

67663 APPLICATION USAGE is added, recommending that the *waitid()* function not be used. +

67664 **NAME**
67665 waitpid — wait for a child process to stop or terminate

67666 **SYNOPSIS**
67667 #include <sys/wait.h>
67668 pid_t waitpid(pid_t pid, int *stat_loc, int options);

67669 **DESCRIPTION**
67670 Refer to *wait*.



wcpcpy()

67671 **NAME**
67672 `wcpcpy` — copy a wide-character string, returning a pointer to its end

SYNOPSIS

67673 CX `#include <wchar.h>`
67674 `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`

DESCRIPTION

67676 Refer to *wcscopy()*.
67677

67678 **NAME**67679 `wcpncpy` — copy a fixed-size wide-character string, returning a pointer to its end67680 **SYNOPSIS**

```
67681 CX #include <wchar.h>
67682     wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
67683                     size_t n);
```

67684 **DESCRIPTION**67685 Refer to *wcsncpy()*.

67686 **NAME**67687 `wcrtomb` — convert a wide-character code to a character (restartable)67688 **SYNOPSIS**67689 `#include <stdio.h>`67690 `size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);`67691 **DESCRIPTION**67692 CX The functionality described on this reference page is aligned with the ISO C standard. Any
67693 conflict between the requirements described here and the ISO C standard is unintentional. This
67694 volume of POSIX.1-200x defers to the ISO C standard.67695 If *s* is a null pointer, the `wcrtomb()` function shall be equivalent to the call:67696 `wcrtomb(buf, L'\0', ps)`67697 where *buf* is an internal buffer.67698 If *s* is not a null pointer, the `wcrtomb()` function shall determine the number of bytes needed to
67699 represent the character that corresponds to the wide character given by *wc* (including any shift
67700 sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At
67701 most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored,
67702 preceded by any shift sequence needed to restore the initial shift state. The resulting state
67703 described shall be the initial conversion state.67704 If *ps* is a null pointer, the `wcrtomb()` function shall use its own internal **mbstate_t** object, which is
67705 initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
67706 pointed to by *ps* shall be used to completely describe the current conversion state of the
67707 associated character sequence. The implementation shall behave as if no function defined in this
67708 volume of POSIX.1-200x calls `wcrtomb()`.67709 CX If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
67710 functions, the application shall ensure that the `wcrtomb()` function is called with a non-NULL *ps*
67711 argument.67712 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.67713 **RETURN VALUE**67714 The `wcrtomb()` function shall return the number of bytes stored in the array object (including any
67715 shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this
67716 case, the function shall store the value of the macro [EILSEQ] in *errno* and shall return (**size_t**)-1;
67717 the conversion state shall be undefined.67718 **ERRORS**67719 The `wcrtomb()` function may fail if:67720 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

67721 [EILSEQ] Invalid wide-character code is detected.

67722
67723
67724
67725
67726
67727
67728
67729
67730
67731
67732
67733
67734
67735
67736
67737
67738
67739
67740

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit(), *wcsrtombs()*

XBD <[wchar.h](#)>

CHANGE HISTORY

First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E).

Issue 6

In the DESCRIPTION, a note on using this function in a threaded application is added.

Extensions beyond the ISO C standard are marked.

The normative text is updated to avoid use of the term “must” for application requirements.

The *wrtomb()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

67741 **NAME**

67742 wscasecmp, wscasecmp_l, wcsncasecmp, wcsncasecmp_l — case-insensitive wide-character
67743 string comparison

67744 **SYNOPSIS**

```
67745 CX #include <wchar.h>
67746
67746 int wscasecmp(const wchar_t *ws1, const wchar_t *ws2);
67747 int wscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
67748 locale_t locale);
67749 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
67750 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
67751 size_t n, locale_t locale);
```

67752 **DESCRIPTION**

67753 The *wscasecmp()* and *wcsncasecmp()* functions are the wide-character equivalent of the
67754 *strcasecmp()* and *strncasecmp()* functions, respectively.

67755 The *wscasecmp()* and *wscasecmp_l()* functions shall compare, while ignoring differences in case,
67756 the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

67757 The *wcsncasecmp()* and *wcsncasecmp_l()* functions shall compare, while ignoring differences in
67758 case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the
67759 wide-character string pointed to by *ws2*.

67760 When the *LC_CTIME* category of the current locale is from the POSIX locale, these functions
67761 shall behave as if the strings had been converted to lowercase and then a byte comparison
67762 performed. Otherwise, the results are unspecified.

67763 The information for *wscasecmp_l()* and *wcsncasecmp_l()* about the case of the characters comes
67764 from the locale represented by *locale*.

67765 **RETURN VALUE**

67766 Upon completion, the *wscasecmp()* and *wscasecmp_l()* functions shall return an integer greater
67767 than, equal to, or less than 0 if the wide-character string pointed to by *ws1* is, ignoring case,
67768 greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

67769 Upon completion, the *wcsncasecmp()* and *wcsncasecmp_l()* functions shall return an integer
67770 greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed
67771 to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character
67772 terminated string pointed to by *ws2*, respectively.

67773 No return values are reserved to indicate an error.

67774 **ERRORS**

67775 The *wscasecmp_l()* and *wcsncasecmp_l()* functions may fail if:

67776 [EINVAL] *locale* is not a valid locale object handle.

67777 **EXAMPLES**

67778 None.

67779 **APPLICATION USAGE**

67780 None.

67781 **RATIONALE**

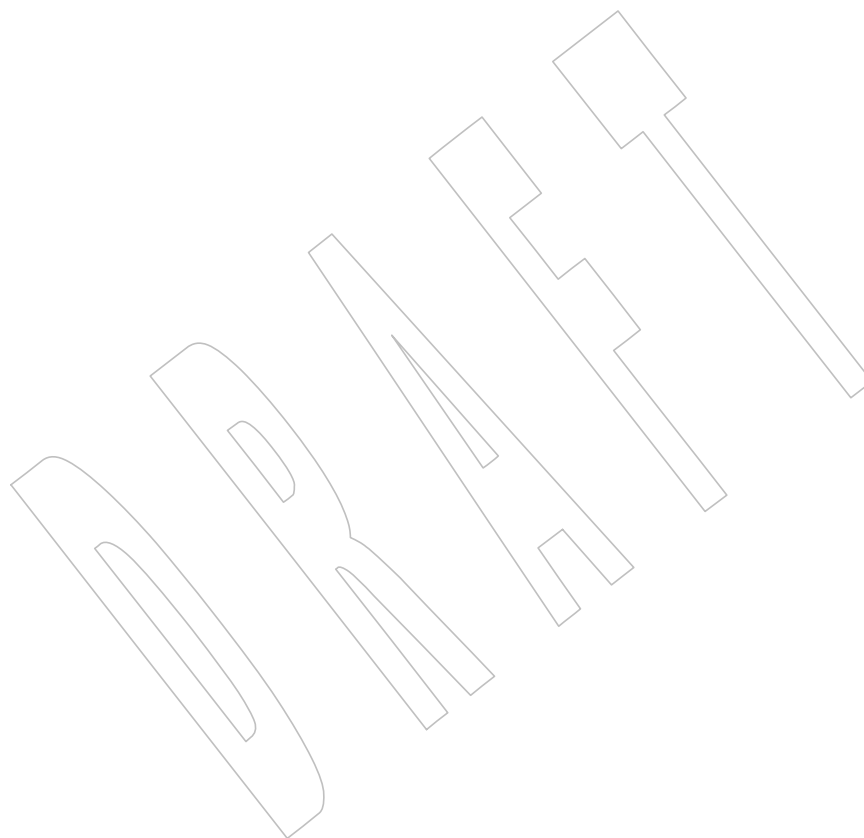
67782 None.

67783 **FUTURE DIRECTIONS**

67784 None.

67785 **SEE ALSO**67786 *strcasecmp()*, *wscmp()*, *wcsncmp()*67787 XBD <[wchar.h](#)>67788 **CHANGE HISTORY**

67789 First released in Issue 7.



67790 **NAME**67791 `wcscat` — concatenate two wide-character strings67792 **SYNOPSIS**67793 `#include <wchar.h>`67794 `wchar_t *wcscat(wchar_t *restrict ws1, const wchar_t *restrict ws2);`67795 **DESCRIPTION**67796 CX The functionality described on this reference page is aligned with the ISO C standard. Any
67797 conflict between the requirements described here and the ISO C standard is unintentional. This
67798 volume of POSIX.1-200x defers to the ISO C standard.67799 The `wcscat()` function shall append a copy of the wide-character string pointed to by `ws2`
67800 (including the terminating null wide-character code) to the end of the wide-character string
67801 pointed to by `ws1`. The initial wide-character code of `ws2` shall overwrite the null wide-character
67802 code at the end of `ws1`. If copying takes place between objects that overlap, the behavior is
67803 undefined.67804 **RETURN VALUE**67805 The `wcscat()` function shall return `ws1`; no return value is reserved to indicate an error.67806 **ERRORS**

67807 No errors are defined.

67808 **EXAMPLES**

67809 None.

67810 **APPLICATION USAGE**

67811 None.

67812 **RATIONALE**

67813 None.

67814 **FUTURE DIRECTIONS**

67815 None.

67816 **SEE ALSO**67817 [`wcsncat\(\)`](#)67818 XBD [`<wchar.h>`](#)67819 **CHANGE HISTORY**

67820 First released in Issue 4. Derived from the MSE working draft.

67821 **Issue 6**67822 The Open Group Corrigendum U040/2 is applied. In the RETURN VALUE section, `s1` is
67823 changed to `ws1`.67824 The `wcscat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

67825 **NAME**
 67826 `wcschr` — wide-character string scanning operation

67827 **SYNOPSIS**
 67828 `#include <wchar.h>`

67829 `wchar_t *wcschr(const wchar_t *ws, wchar_t wc);`

67830 **DESCRIPTION**

67831 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67832 conflict between the requirements described here and the ISO C standard is unintentional. This
 67833 volume of POSIX.1-200x defers to the ISO C standard.

67834 The `wcschr()` function shall locate the first occurrence of `wc` in the wide-character string pointed
 67835 to by `ws`. The application shall ensure that the value of `wc` is a character representable as a type
 67836 `wchar_t` and a wide-character code corresponding to a valid character in the current locale. The
 67837 terminating null wide-character code is considered to be part of the wide-character string.

67838 **RETURN VALUE**

67839 Upon completion, `wcschr()` shall return a pointer to the wide-character code, or a null pointer if
 67840 the wide-character code is not found.

67841 **ERRORS**

67842 No errors are defined.

67843 **EXAMPLES**

67844 None.

67845 **APPLICATION USAGE**

67846 None.

67847 **RATIONALE**

67848 None.

67849 **FUTURE DIRECTIONS**

67850 None.

67851 **SEE ALSO**

67852 [wcsrchr\(\)](#)

67853 XBD [<wchar.h>](#)

67854 **CHANGE HISTORY**

67855 First released in Issue 4. Derived from the MSE working draft.

67856 **Issue 6**

67857 The normative text is updated to avoid use of the term “must” for application requirements.

67858 **NAME**
 67859 `wcscmp` — compare two wide-character strings

67860 **SYNOPSIS**

67861 `#include <wchar.h>`

67862 `int wcscmp(const wchar_t *ws1, const wchar_t *ws2);`

67863 **DESCRIPTION**

67864 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 67865 conflict between the requirements described here and the ISO C standard is unintentional. This
 67866 volume of POSIX.1-200x defers to the ISO C standard.

67867 The `wcscmp()` function shall compare the wide-character string pointed to by `ws1` to the wide-
 67868 character string pointed to by `ws2`.

67869 The sign of a non-zero return value shall be determined by the sign of the difference between the
 67870 values of the first pair of wide-character codes that differ in the objects being compared.

67871 **RETURN VALUE**

67872 Upon completion, `wcscmp()` shall return an integer greater than, equal to, or less than 0, if the
 67873 wide-character string pointed to by `ws1` is greater than, equal to, or less than the wide-character
 67874 string pointed to by `ws2`, respectively.

67875 **ERRORS**

67876 No errors are defined.

67877 **EXAMPLES**

67878 None.

67879 **APPLICATION USAGE**

67880 None.

67881 **RATIONALE**

67882 None.

67883 **FUTURE DIRECTIONS**

67884 None.

67885 **SEE ALSO**

67886 [wcscasecmp\(\)](#), [wcsncmp\(\)](#)

67887 XBD [<wchar.h>](#)

67888 **CHANGE HISTORY**

67889 First released in Issue 4. Derived from the MSE working draft.

67890 **NAME**
 67891 `wscoll`, `wscoll_l` — wide-character string comparison using collating information

67892 **SYNOPSIS**

```
67893 #include <wchar.h>
67894
67894 int wscoll(const wchar_t *ws1, const wchar_t *ws2);
67895 CX int wscoll_l(const wchar_t *ws1, const wchar_t *ws2,
67896 locale_t locale);
```

67897 **DESCRIPTION**

67898 CX For `wscoll()`: The functionality described on this reference page is aligned with the ISO C
 67899 standard. Any conflict between the requirements described here and the ISO C standard is
 67900 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

67901 CX The `wscoll()` and `wscoll_l()` functions shall compare the wide-character string pointed to by
 67902 `ws1` to the wide-character string pointed to by `ws2`, both interpreted as appropriate to the
 67903 CX `LC_COLLATE` category of the current locale of the process, or the locale represented by `locale`,
 67904 respectively.

67905 CX The `wscoll()` and `wscoll_l()` functions shall not change the setting of `errno` if successful.

67906 CX An application wishing to check for error situations should set `errno` to 0 before calling `wscoll()`
 67907 or `wscoll_l()`. If `errno` is non-zero on return, an error has occurred.

67908 **RETURN VALUE**

67909 CX Upon successful completion, `wscoll()` and `wscoll_l()` shall return an integer greater than, equal
 67910 to, or less than 0, according to whether the wide-character string pointed to by `ws1` is greater
 67911 than, equal to, or less than the wide-character string pointed to by `ws2`, when both are
 67912 CX interpreted as appropriate to the current locale, or to the locale represented by `locale`,
 67913 CX respectively. On error, `wscoll()` and `wscoll_l()` shall set `errno`, but no return value is reserved
 67914 to indicate an error.

67915 **ERRORS**

67916 These functions may fail if:

67917 CX [EINVAL] The `ws1` or `ws2` arguments contain wide-character codes outside the domain of
 67918 the collating sequence.

67919 The `wscoll_l()` function may fail if:

67920 CX [EINVAL] `locale` is not a valid locale object handle.

67921 **EXAMPLES**

67922 None.

67923 **APPLICATION USAGE**

67924 The `wcsxfrm()` and `wscmp()` functions should be used for sorting large lists.

67925 **RATIONALE**

67926 None.

67927 **FUTURE DIRECTIONS**

67928 None.

67929

SEE ALSO

67930

wscmp(), *wcsxfrm()*

67931

XBD <[wchar.h](#)>

67932

CHANGE HISTORY

67933

First released in Issue 4. Derived from the MSE working draft.

67934

Issue 5

67935

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

67936

The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

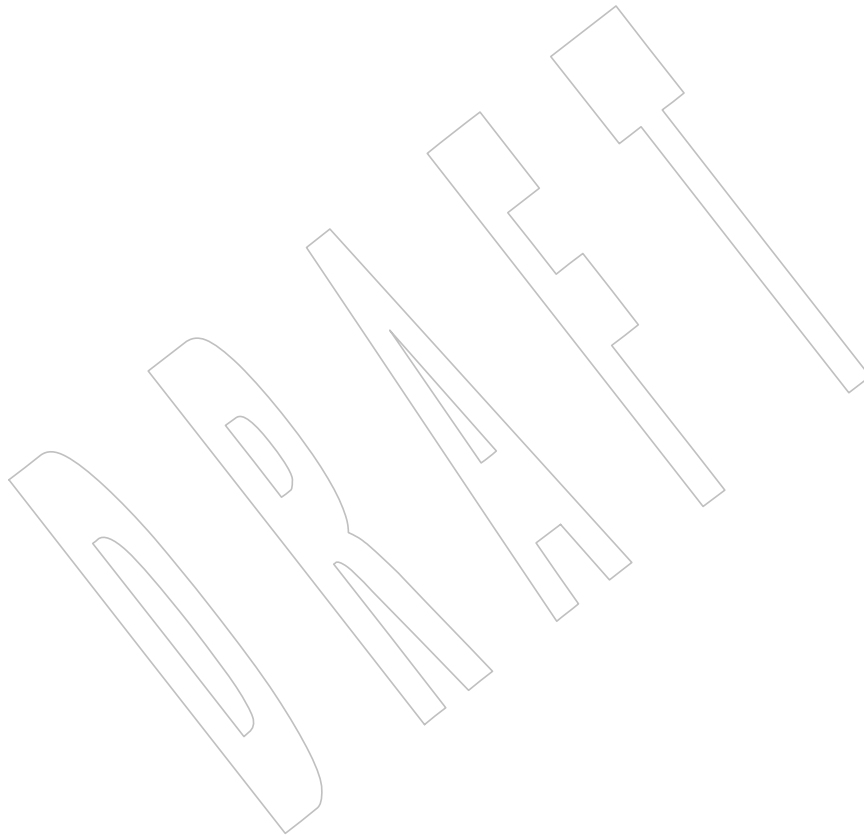
67937

Issue 7

67938

The *wscoll_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

67939



67940 **NAME**67941 `wcpcpy`, `wcscopy` — copy a wide-character string, returning a pointer to its end67942 **SYNOPSIS**67943 `#include <wchar.h>`67944 CX `wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`67945 `wchar_t *wcscopy(wchar_t *restrict ws1, const wchar_t *restrict ws2);`67946 **DESCRIPTION**67947 CX For `wcscopy()`: The functionality described on this reference page is aligned with the ISO C
67948 standard. Any conflict between the requirements described here and the ISO C standard is
67949 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.67950 CX The `wcpcpy()` and `wcscopy()` functions shall copy the wide-character string pointed to by `ws2`
67951 (including the terminating null wide-character code) into the array pointed to by `ws1`.67952 The application shall ensure that there is room for at least `wcslen(ws2)+1` wide characters in the
67953 `ws1` array, and that the `ws2` and `ws1` arrays do not overlap.

67954 If copying takes place between objects that overlap, the behavior is undefined.

67955 **RETURN VALUE**67956 CX The `wcpcpy()` function shall return a pointer to the terminating null wide-character code copied
67957 into the `ws1` buffer.67958 The `wcscopy()` function shall return `ws1`.

67959 No return values are reserved to indicate an error.

67960 **ERRORS**

67961 No errors are defined.

67962 **EXAMPLES**

67963 None.

67964 **APPLICATION USAGE**

67965 None.

67966 **RATIONALE**

67967 None.

67968 **FUTURE DIRECTIONS**

67969 None.

67970 **SEE ALSO**67971 [*strcpy\(\)*](#), [*wcsdup\(\)*](#), [*wscncpy\(\)*](#)67972 XBD [*<wchar.h>*](#)67973 **CHANGE HISTORY**

67974 First released in Issue 4. Derived from the MSE working draft.

67975 **Issue 6**67976 The `wcscopy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.67977 **Issue 7**67978 The `wcpcpy()` function is added from The Open Group Technical Standard, 2006, Extended API
67979 Set Part 1.

67980 **NAME**67981 `wcscspn` — get the length of a complementary wide substring67982 **SYNOPSIS**67983 `#include <wchar.h>`67984 `size_t wcscspn(const wchar_t *ws1, const wchar_t *ws2);`67985 **DESCRIPTION**67986 CX The functionality described on this reference page is aligned with the ISO C standard. Any
67987 conflict between the requirements described here and the ISO C standard is unintentional. This
67988 volume of POSIX.1-200x defers to the ISO C standard.67989 The `wcscspn()` function shall compute the length (in wide characters) of the maximum initial
67990 segment of the wide-character string pointed to by `ws1` which consists entirely of wide-character
67991 codes *not* from the wide-character string pointed to by `ws2`.67992 **RETURN VALUE**67993 The `wcscspn()` function shall return the length of the initial substring of `ws1`; no return value is
67994 reserved to indicate an error.67995 **ERRORS**

67996 No errors are defined.

67997 **EXAMPLES**

67998 None.

67999 **APPLICATION USAGE**

68000 None.

68001 **RATIONALE**

68002 None.

68003 **FUTURE DIRECTIONS**

68004 None.

68005 **SEE ALSO**68006 [*wcsspn\(\)*](#)68007 XBD [*<wchar.h>*](#)68008 **CHANGE HISTORY**

68009 First released in Issue 4. Derived from the MSE working draft.

68010 **Issue 5**68011 The RETURN VALUE section is updated to indicate that `wcscspn()` returns the length of `ws1`,
68012 rather than `ws1` itself.

68013 **NAME**

68014 wcsdup — duplicate a wide-character string

68015 **SYNOPSIS**

```
68016 CX      #include <wchar.h>
68017      wchar_t *wcsdup(const wchar_t *string);
```

68018 **DESCRIPTION**68019 The *wcsdup()* function is the wide-character equivalent of the *strdup()* function.

68020 The *wcsdup()* function shall return a pointer to a new wide-character string, which is the
 68021 duplicate of the wide-character string *string*. The returned pointer can be passed to *free()*. A
 68022 null pointer is returned if the new wide-character string cannot be created.

68023 **RETURN VALUE**

68024 Upon successful completion, the *wcsdup()* function shall return a pointer to the newly allocated
 68025 wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

68026 **ERRORS**68027 The *wcsdup()* function shall fail if:

68028 [ENOMEM] Memory large enough for the duplicate string could not be allocated.

68029 **EXAMPLES**

68030 None.

68031 **APPLICATION USAGE**

68032 None.

68033 **RATIONALE**

68034 None.

68035 **FUTURE DIRECTIONS**

68036 None.

68037 **SEE ALSO**68038 *free()*, *strdup()*, *wcscpy()*

68039 XBD <wchar.h>

68040 **CHANGE HISTORY**

68041 First released in Issue 7.

68042 **NAME**68043 `wcsftime` — convert date and time to a wide-character string68044 **SYNOPSIS**68045 `#include <wchar.h>`68046 `size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,`
68047 `const wchar_t *restrict format, const struct tm *restrict timeptr);`68048 **DESCRIPTION**68049 CX The functionality described on this reference page is aligned with the ISO C standard. Any
68050 conflict between the requirements described here and the ISO C standard is unintentional. This
68051 volume of POSIX.1-200x defers to the ISO C standard.68052 The `wcsftime()` function shall be equivalent to the `strftime()` function, except that:

- 68053 • The argument `wcs` points to the initial element of an array of wide characters into which
68054 the generated output is to be placed.
- 68055 • The argument `maxsize` indicates the maximum number of wide characters to be placed in
68056 the output array.
- 68057 • The argument `format` is a wide-character string and the conversion specifications are
68058 replaced by corresponding sequences of wide characters.
- 68059 • The return value indicates the number of wide characters placed in the output array.

68060 If copying takes place between objects that overlap, the behavior is undefined.

68061 **RETURN VALUE**68062 If the total number of resulting wide-character codes including the terminating null wide-
68063 character code is no more than `maxsize`, `wcsftime()` shall return the number of wide-character
68064 codes placed into the array pointed to by `wcs`, not including the terminating null wide-character
68065 code. Otherwise, zero is returned and the contents of the array are unspecified.68066 **ERRORS**

68067 No errors are defined.

68068 **EXAMPLES**

68069 None.

68070 **APPLICATION USAGE**

68071 None.

68072 **RATIONALE**

68073 None.

68074 **FUTURE DIRECTIONS**

68075 None.

68076 **SEE ALSO**68077 [*strftime\(\)*](#)68078 XBD [**<wchar.h>**](#)68079 **CHANGE HISTORY**

68080 First released in Issue 4.

68081

Issue 5

68082

Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

68083

68084

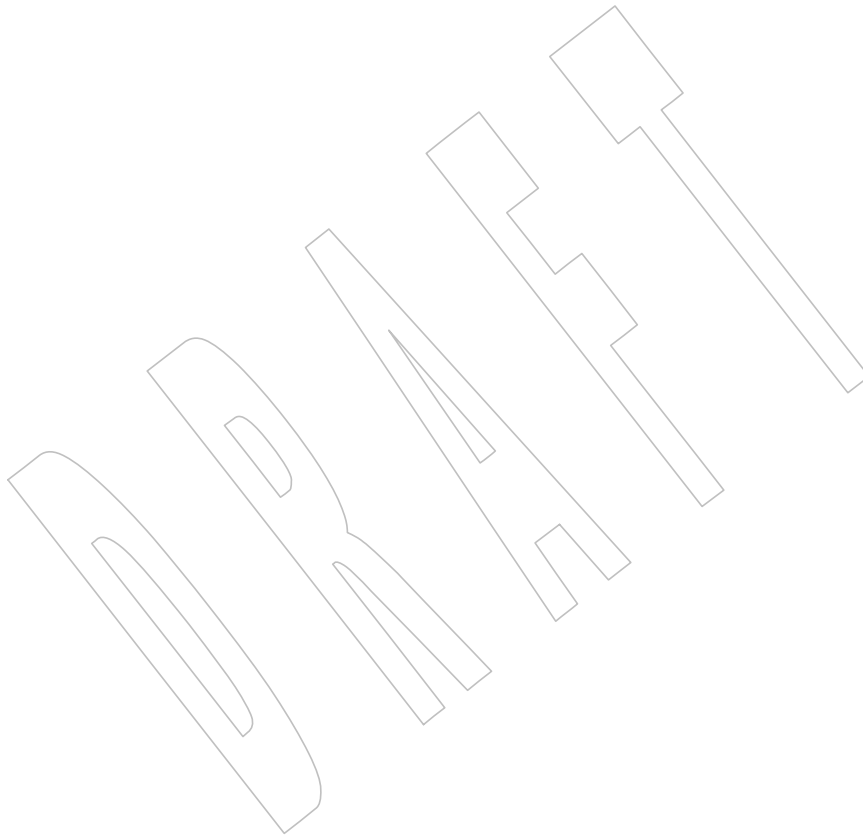
Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, the type of the *format* argument is changed from **const char *** to **const wchar_t ***.

68085

Issue 6

68086

The *wcsftime()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



68087 **NAME**68088 `wcslen`, `wcsnlen` — get length of a fixed-sized wide-character string68089 **SYNOPSIS**68090 `#include <wchar.h>`68091 `size_t wcslen(const wchar_t *ws);`68092 CX `size_t wcsnlen(const wchar_t *ws, size_t maxlen);`68093 **DESCRIPTION**68094 CX For `wcslen()`: The functionality described on this reference page is aligned with the ISO C
68095 standard. Any conflict between the requirements described here and the ISO C standard is
68096 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.68097 The `wcslen()` function shall compute the number of wide-character codes in the wide-character
68098 string to which `ws` points, not including the terminating null wide-character code.68099 CX The `wcsnlen()` function shall compute the smaller of the number of wide characters in the string
68100 to which `ws` points, not including the terminating null wide-character code, and the value of
68101 `maxlen`. The `wcsnlen()` function shall never examine more than the first `maxlen` characters of the
68102 wide-character string pointed to by `ws`.68103 **RETURN VALUE**68104 The `wcslen()` function shall return the length of `ws`.68105 CX The `wcsnlen()` function shall return an integer containing the smaller of either the length of the
68106 wide-character string pointed to by `ws` or `maxlen`.

68107 No return values are reserved to indicate an error.

68108 **ERRORS**

68109 No errors are defined.

68110 **EXAMPLES**

68111 None.

68112 **APPLICATION USAGE**

68113 None.

68114 **RATIONALE**

68115 None.

68116 **FUTURE DIRECTIONS**

68117 None.

68118 **SEE ALSO**68119 [*strlen\(\)*](#)68120 XBD [*<wchar.h>*](#)68121 **CHANGE HISTORY**

68122 First released in Issue 4. Derived from the MSE working draft.

68123 **Issue 7**68124 The `wcsnlen()` function is added from The Open Group Technical Standard, 2006, Extended API
68125 Set Part 1.

68126 **NAME**68127 `wcsncasecmp`, `wcsncasecmp_l` — case-insensitive wide-character string comparison68128 **SYNOPSIS**

```
68129 CX #include <wchar.h>
68130 int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
68131 int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
68132 size_t n, locale_t locale);
```

68133 **DESCRIPTION**68134 Refer to [wcscasecmp\(\)](#).

68135 **NAME**68136 `wcsncat` — concatenate a wide-character string with part of another68137 **SYNOPSIS**68138 `#include <wchar.h>`68139 `wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
68140 `size_t n);`68141 **DESCRIPTION**68142 CX The functionality described on this reference page is aligned with the ISO C standard. Any
68143 conflict between the requirements described here and the ISO C standard is unintentional. This
68144 volume of POSIX.1-200x defers to the ISO C standard.68145 The `wcsncat()` function shall append not more than *n* wide-character codes (a null wide-
68146 character code and wide-character codes that follow it are not appended) from the array pointed
68147 to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character
68148 code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null
68149 wide-character code shall always be appended to the result. If copying takes place between
68150 objects that overlap, the behavior is undefined.68151 **RETURN VALUE**68152 The `wcsncat()` function shall return *ws1*; no return value is reserved to indicate an error.68153 **ERRORS**

68154 No errors are defined.

68155 **EXAMPLES**

68156 None.

68157 **APPLICATION USAGE**

68158 None.

68159 **RATIONALE**

68160 None.

68161 **FUTURE DIRECTIONS**

68162 None.

68163 **SEE ALSO**68164 [wcscat\(\)](#)68165 XBD [<wchar.h>](#)68166 **CHANGE HISTORY**

68167 First released in Issue 4. Derived from the MSE working draft.

68168 **Issue 6**68169 The `wcsncat()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

68170 **NAME**
 68171 `wcsncmp` — compare part of two wide-character strings

68172 **SYNOPSIS**
 68173 `#include <wchar.h>`
 68174 `int wcsncmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);`

68175 **DESCRIPTION**
 68176 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68177 conflict between the requirements described here and the ISO C standard is unintentional. This
 68178 volume of POSIX.1-200x defers to the ISO C standard.

68179 The `wcsncmp()` function shall compare not more than *n* wide-character codes (wide-character
 68180 codes that follow a null wide-character code are not compared) from the array pointed to by *ws1*
 68181 to the array pointed to by *ws2*.

68182 The sign of a non-zero return value shall be determined by the sign of the difference between the
 68183 values of the first pair of wide-character codes that differ in the objects being compared.

68184 **RETURN VALUE**
 68185 Upon successful completion, `wcsncmp()` shall return an integer greater than, equal to, or less
 68186 than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less
 68187 than the possibly null-terminated array pointed to by *ws2*, respectively.

68188 **ERRORS**
 68189 No errors are defined.

68190 **EXAMPLES**
 68191 None.

68192 **APPLICATION USAGE**
 68193 None.

68194 **RATIONALE**
 68195 None.

68196 **FUTURE DIRECTIONS**
 68197 None.

68198 **SEE ALSO**
 68199 [wscasecmp\(\)](#), [wscmp\(\)](#)

68200 XBD [<wchar.h>](#)

68201 **CHANGE HISTORY**
 68202 First released in Issue 4. Derived from the MSE working draft.

68203 **NAME**68204 `wcpncpy`, `wcsncpy` — copy a fixed-size wide-character string, returning a pointer to its end68205 **SYNOPSIS**68206 `#include <wchar.h>`68207 CX `wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,`
68208 `size_t n);`68209 `wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
68210 `size_t n);`68211 **DESCRIPTION**68212 CX For `wcsncpy()`: The functionality described on this reference page is aligned with the ISO C
68213 standard. Any conflict between the requirements described here and the ISO C standard is
68214 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.68215 CX The `wcpncpy()` and `wcsncpy()` functions shall copy not more than *n* wide-character codes (wide-
68216 character codes that follow a null wide-character code are not copied) from the array pointed to
68217 by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the
68218 behavior is undefined.68219 If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character
68220 codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*,
68221 until *n* wide-character codes in all are written.68222 **RETURN VALUE**68223 CX If any null wide-character codes were written into the destination, the `wcpncpy()` function shall
68224 return the address of the first such null wide-character code. Otherwise, it shall return `&ws1[n]`.68225 The `wcsncpy()` function shall return *ws1*.

68226 No return values are reserved to indicate an error.

68227 **ERRORS**

68228 No errors are defined.

68229 **EXAMPLES**

68230 None.

68231 **APPLICATION USAGE**68232 If there is no null wide-character code in the first *n* wide-character codes of the array pointed to
68233 by *ws2*, the result is not null-terminated.68234 **RATIONALE**

68235 None.

68236 **FUTURE DIRECTIONS**

68237 None.

68238 **SEE ALSO**68239 [strncpy\(\)](#), [wcsncpy\(\)](#)68240 XBD [<wchar.h>](#)68241 **CHANGE HISTORY**

68242 First released in Issue 4. Derived from the MSE working draft.

68243

Issue 6

68244

The *wcsncpy()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

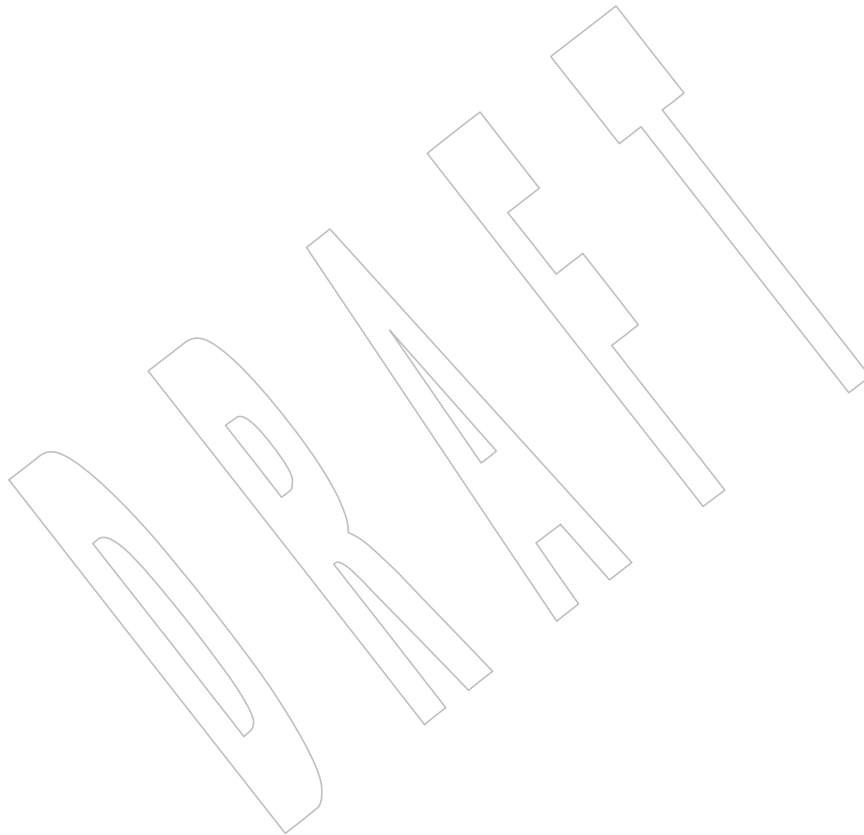
68245

Issue 7

68246

The *wcpncpy()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

68247



wcsnlen()

68248 **NAME**
68249 `wcsnlen` — get length of a fixed-sized wide-character string

SYNOPSIS

68251 CX `#include <wchar.h>`
68252 `size_t wcsnlen(const wchar_t *ws, size_t maxlen);`

DESCRIPTION

68253 Refer to *wcslen()*.
68254

68255 **NAME**68256 `wcsnrtombs` — convert wide-character string to multi-byte string68257 **SYNOPSIS**

```
68258 CX #include <wchar.h>  
68259 size_t wcsnrtombs(char *dst, const wchar_t **src, size_t nwc,  
68260 size_t len, mbstate_t *ps);
```

68261 **DESCRIPTION**68262 Refer to [wcsrtombs\(\)](#).

68263 **NAME**68264 `wcspbrk` — scan a wide-character string for a wide-character code68265 **SYNOPSIS**68266 `#include <wchar.h>`68267 `wchar_t *wcspbrk(const wchar_t *ws1, const wchar_t *ws2);`68268 **DESCRIPTION**68269 CX The functionality described on this reference page is aligned with the ISO C standard. Any
68270 conflict between the requirements described here and the ISO C standard is unintentional. This
68271 volume of POSIX.1-200x defers to the ISO C standard.68272 The `wcspbrk()` function shall locate the first occurrence in the wide-character string pointed to by
68273 `ws1` of any wide-character code from the wide-character string pointed to by `ws2`.68274 **RETURN VALUE**68275 Upon successful completion, `wcspbrk()` shall return a pointer to the wide-character code or a null
68276 pointer if no wide-character code from `ws2` occurs in `ws1`.68277 **ERRORS**

68278 No errors are defined.

68279 **EXAMPLES**

68280 None.

68281 **APPLICATION USAGE**

68282 None.

68283 **RATIONALE**

68284 None.

68285 **FUTURE DIRECTIONS**

68286 None.

68287 **SEE ALSO**68288 [*wcchr\(\)*](#), [*wcsrchr\(\)*](#)68289 XBD [*<wchar.h>*](#)68290 **CHANGE HISTORY**

68291 First released in Issue 4. Derived from the MSE working draft.

68292 **NAME**
 68293 wcsrchr — wide-character string scanning operation

68294 **SYNOPSIS**
 68295 #include <wchar.h>
 68296 wchar_t *wcsrchr(const wchar_t *ws, wchar_t wc);

68297 **DESCRIPTION**
 68298 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68299 conflict between the requirements described here and the ISO C standard is unintentional. This
 68300 volume of POSIX.1-200x defers to the ISO C standard.

68301 The *wcsrchr()* function shall locate the last occurrence of *wc* in the wide-character string pointed
 68302 to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type
 68303 **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The
 68304 terminating null wide-character code shall be considered to be part of the wide-character string.

68305 **RETURN VALUE**
 68306 Upon successful completion, *wcsrchr()* shall return a pointer to the wide-character code or a null
 68307 pointer if *wc* does not occur in the wide-character string.

68308 **ERRORS**
 68309 No errors are defined.

68310 **EXAMPLES**
 68311 None.

68312 **APPLICATION USAGE**
 68313 None.

68314 **RATIONALE**
 68315 None.

68316 **FUTURE DIRECTIONS**
 68317 None.

68318 **SEE ALSO**
 68319 *wcschr()*
 68320 XBD <wchar.h>

68321 **CHANGE HISTORY**
 68322 First released in Issue 4. Derived from the MSE working draft.

68323 **Issue 6**
 68324 The normative text is updated to avoid use of the term “must” for application requirements.

68325 **NAME**

68326 wcsnrtombs, wcsrtombs — convert a wide-character string to a character string (restartable)

68327 **SYNOPSIS**

68328 #include <wchar.h>

68329 CX

```
size_t wcsnrtombs(char *dst, const wchar_t **src, size_t nwc,  
size_t len, mbstate_t *ps);
```

68330

```
size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,  
size_t len, mbstate_t *restrict ps);
```

68333 **DESCRIPTION**68334 CX For *wcsrtombs()*: The functionality described on this reference page is aligned with the ISO C
68335 standard. Any conflict between the requirements described here and the ISO C standard is
68336 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.68337 The *wcsrtombs()* function shall convert a sequence of wide characters from the array indirectly
68338 pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state
68339 described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters
68340 shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a
68341 terminating null wide character, which shall also be stored. Conversion shall stop earlier in the
68342 following cases:

- 68343
- When a code is reached that does not correspond to a valid character
 - When the next character would exceed the limit of *len* total bytes to be stored in the array
68344 pointed to by *dst* (and *dst* is not a null pointer)

68346 Each conversion shall take place as if by a call to the *wcrtomb()* function.68347 If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null
68348 pointer (if conversion stopped due to reaching a terminating null wide character) or the address
68349 just past the last wide character converted (if any). If conversion stopped due to reaching a
68350 terminating null wide character, the resulting state described shall be the initial conversion state.68351 If *ps* is a null pointer, the *wcsrtombs()* function shall use its own internal **mbstate_t** object, which
68352 is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object
68353 pointed to by *ps* shall be used to completely describe the current conversion state of the
68354 associated character sequence.68355 CX If the application uses any of the `_POSIX_THREAD_SAFE_FUNCTIONS` or `_POSIX_THREADS`
68356 functions, the application shall ensure that the *wcsrtombs()* function is called with a non-NULL
68357 *ps* argument.68358 The *wcsnrtombs()* function shall be equivalent to the *wcsrtombs()* function, except that the
68359 conversion is limited to the first *nwc* wide characters.68360 The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.68361 The implementation shall behave as if no function defined in System Interfaces volume of
68362 POSIX.1-200x calls these functions.68363 **RETURN VALUE**68364 If conversion stops because a code is reached that does not correspond to a valid character, an
68365 encoding error occurs. In this case, these functions shall store the value of the macro `[EILSEQ]` in
68366 *errno* and return (**size_t**)-1; the conversion state is undefined. Otherwise, these functions shall
68367 return the number of bytes in the resulting character sequence, not including the terminating
68368 null (if any).

68369 **ERRORS**

68370 These functions may fail if:

68371 CX [EINVAL] *ps* points to an object that contains an invalid conversion state.

68372 [EILSEQ] A wide-character code does not correspond to a valid character.

68373 **EXAMPLES**

68374 None.

68375 **APPLICATION USAGE**

68376 None.

68377 **RATIONALE**

68378 None.

68379 **FUTURE DIRECTIONS**

68380 None.

68381 **SEE ALSO**68382 *mbsinit()*, *wcrtomb()*68383 XBD <[wchar.h](#)>68384 **CHANGE HISTORY**68385 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
68386 (E).68387 **Issue 6**

68388 In the DESCRIPTION, a note on using this function in a threaded application is added.

68389 Extensions beyond the ISO C standard are marked.

68390 The normative text is updated to avoid use of the term “must” for application requirements.

68391 The *wcsrtombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.68392 **Issue 7**68393 The *wcnsrtombs()* function is added from The Open Group Technical Standard, 2006, Extended
68394 API Set Part 1.

68395 **NAME**
 68396 `wcsspn` — get the length of a wide substring

68397 **SYNOPSIS**
 68398 `#include <wchar.h>`

68399 `size_t wcsspn(const wchar_t *ws1, const wchar_t *ws2);`

68400 **DESCRIPTION**

68401 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68402 conflict between the requirements described here and the ISO C standard is unintentional. This
 68403 volume of POSIX.1-200x defers to the ISO C standard.

68404 The `wcsspn()` function shall compute the length (in wide characters) of the maximum initial
 68405 segment of the wide-character string pointed to by `ws1` which consists entirely of wide-character
 68406 codes from the wide-character string pointed to by `ws2`.

68407 **RETURN VALUE**

68408 The `wcsspn()` function shall return the length of the initial substring of `ws1`; no return value is
 68409 reserved to indicate an error.

68410 **ERRORS**

68411 No errors are defined.

68412 **EXAMPLES**

68413 None.

68414 **APPLICATION USAGE**

68415 None.

68416 **RATIONALE**

68417 None.

68418 **FUTURE DIRECTIONS**

68419 None.

68420 **SEE ALSO**

68421 [*wcscspn\(\)*](#)

68422 XBD [*<wchar.h>*](#)

68423 **CHANGE HISTORY**

68424 First released in Issue 4. Derived from the MSE working draft.

68425 **Issue 5**

68426 The RETURN VALUE section is updated to indicate that `wcsspn()` returns the length of `ws1`
 68427 rather than `ws1` itself.

68428 **NAME**68429 `wcsstr` — find a wide-character substring68430 **SYNOPSIS**68431 `#include <wchar.h>`68432 `wchar_t *wcsstr(const wchar_t *restrict ws1,`
68433 `const wchar_t *restrict ws2);`68434 **DESCRIPTION**68435 **CX** The functionality described on this reference page is aligned with the ISO C standard. Any
68436 conflict between the requirements described here and the ISO C standard is unintentional. This
68437 volume of POSIX.1-200x defers to the ISO C standard.68438 The `wcsstr()` function shall locate the first occurrence in the wide-character string pointed to by
68439 `ws1` of the sequence of wide characters (excluding the terminating null wide character) in the
68440 wide-character string pointed to by `ws2`.68441 **RETURN VALUE**68442 Upon successful completion, `wcsstr()` shall return a pointer to the located wide-character string,
68443 or a null pointer if the wide-character string is not found.68444 If `ws2` points to a wide-character string with zero length, the function shall return `ws1`.68445 **ERRORS**

68446 No errors are defined.

68447 **EXAMPLES**

68448 None.

68449 **APPLICATION USAGE**

68450 None.

68451 **RATIONALE**

68452 None.

68453 **FUTURE DIRECTIONS**

68454 None.

68455 **SEE ALSO**68456 [*wcschr\(\)*](#)68457 XBD [*<wchar.h>*](#)68458 **CHANGE HISTORY**68459 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
68460 (E).68461 **Issue 6**68462 The `wcsstr()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

68463 **NAME**
 68464 `wcstod`, `wcstof`, `wcstold` — convert a wide-character string to a double-precision number

68465 **SYNOPSIS**
 68466 `#include <wchar.h>`
 68467 `double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);`
 68468 `float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);`
 68469 `long double wcstold(const wchar_t *restrict nptr,`
 68470 `wchar_t **restrict endptr);`

68471 DESCRIPTION

68472 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68473 conflict between the requirements described here and the ISO C standard is unintentional. This
 68474 volume of POSIX.1-200x defers to the ISO C standard.

68475 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
 68476 **double**, **float**, and **long double** representation, respectively. First, they shall decompose the
 68477 input wide-character string into three parts:

- 68478 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
 68479 `iswspace()`)
- 68480 2. A subject sequence interpreted as a floating-point constant or representing infinity or
 68481 NaN
- 68482 3. A final wide-character string of one or more unrecognized wide-character codes,
 68483 including the terminating null wide-character code of the input wide-character string

68484 Then they shall attempt to convert the subject sequence to a floating-point number, and return
 68485 the result.

68486 The expected form of the subject sequence is an optional plus or minus sign, then one of the
 68487 following:

- 68488 • A non-empty sequence of decimal digits optionally containing a radix character; then an
 68489 optional exponent part consisting of the wide character 'e' or the wide character 'E',
 68490 optionally followed by a '+' or '-' wide character, and then followed by one or more
 68491 decimal digits
- 68492 • A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix
 68493 character; then an optional binary exponent part consisting of the wide character 'p' or
 68494 the wide character 'P', optionally followed by a '+' or '-' wide character, and then
 68495 followed by one or more decimal digits
- 68496 • One of INF or INFINITY, or any other wide string equivalent except for case
- 68497 • One of NAN or NAN(*n-wchar-sequence_{opt}*), or any other wide string ignoring case in the
 68498 NAN part, where:

68499 `n-wchar-sequence:`
 68500 `digit`
 68501 `nondigit`
 68502 `n-wchar-sequence digit`
 68503 `n-wchar-sequence nondigit`

68504 The subject sequence is defined as the longest initial subsequence of the input wide string,
 68505 starting with the first non-white-space wide character, that is of the expected form. The subject
 68506 sequence contains no wide characters if the input wide string is not of the expected form.

68507 If the subject sequence has the expected form for a floating-point number, the sequence of wide
 68508 characters starting with the first digit or the radix character (whichever occurs first) shall be
 68509 interpreted as a floating constant according to the rules of the C language, except that the radix
 68510 character shall be used in place of a period, and that if neither an exponent part nor a radix
 68511 character appears in a decimal floating-point number, or if a binary exponent part does not
 68512 appear in a hexadecimal floating-point number, an exponent part of the appropriate type with
 68513 value zero shall be assumed to follow the last digit in the string. If the subject sequence begins
 68514 with a minus sign, the sequence shall be interpreted as negated. A wide-character sequence INF
 68515 or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it
 68516 were a floating constant that is too large for the range of the return type. A wide-character
 68517 sequence NAN or NAN(*n-wchar-sequence_{opt}*) shall be interpreted as a quiet NaN, if supported in
 68518 the return type, else as if it were a subject sequence part that does not have the expected form;
 68519 the meaning of the *n-wchar* sequences is implementation-defined. A pointer to the final wide
 68520 string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

68521 If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the
 68522 conversion shall be rounded in an implementation-defined manner.

68523 CX The radix character shall be as defined in the locale of the process (category *LC_NUMERIC*). In
 68524 the POSIX locale, or in a locale where the radix character is not defined, the radix character shall
 68525 default to a period (' . ').

68526 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
 68527 accepted.

68528 If the subject sequence is empty or does not have the expected form, no conversion shall be
 68529 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
 68530 *endptr* is not a null pointer.

68531 CX The *wcstod()* function shall not change the setting of *errno* if successful.

68532 Since 0 is returned on error and is also a valid return on success, an application wishing to check
 68533 for error situations should set *errno* to 0, then call *wcstod()*, *wcstof()*, or *wcstold()*, then check
 68534 *errno*.

68535 RETURN VALUE

68536 Upon successful completion, these functions shall return the converted value. If no conversion
 68537 CX could be performed, 0 shall be returned and *errno* may be set to [EINVAL].

68538 If the correct value is outside the range of representable values, \pm HUGE_VAL, \pm HUGE_VALF, or
 68539 \pm HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to
 68540 [ERANGE].

68541 If the correct value would cause underflow, a value whose magnitude is no greater than the
 68542 smallest normalized positive number in the return type shall be returned and *errno* set to
 68543 [ERANGE].

68544 ERRORS

68545 The *wcstod()* function shall fail if:

68546 [ERANGE] The value to be returned would cause overflow or underflow.

68547 The *wcstod()* function may fail if:

68548 CX [EINVAL] No conversion could be performed.

EXAMPLES

68549 None.
68550

APPLICATION USAGE

68551 If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the
68552 result is not exactly representable, the result should be one of the two numbers in the
68553 appropriate internal format that are adjacent to the hexadecimal floating source value, with the
68554 extra stipulation that the error should have a correct sign for the current rounding direction.
68555

68556 If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in `<float.h>`)
68557 significant digits, the result should be correctly rounded. If the subject sequence *D* has the
68558 decimal form and more than DECIMAL_DIG significant digits, consider the two bounding,
68559 adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the
68560 values of *L*, *D*, and *U* satisfy " $L \leq D \leq U$ ". The result should be one of the (equal or
68561 adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current
68562 rounding direction, with the extra stipulation that the error with respect to *D* should have a
68563 correct sign for the current rounding direction.

RATIONALE

68564 None.
68565

FUTURE DIRECTIONS

68566 None.
68567

SEE ALSO

68568 *scanf()*, *iswspace()*, *localeconv()*, *setlocale()*, *wcstol()*

68569 XBD Chapter 7 (on page 121), `<float.h>`, `<wchar.h>`

CHANGE HISTORY

68571 First released in Issue 4. Derived from the MSE working draft.
68572

Issue 5

68573 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
68574

Issue 6

68575 Extensions beyond the ISO C standard are marked.
68576

68577 The following new requirements on POSIX implementations derive from alignment with the
68578 Single UNIX Specification:

- In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is added if no conversion could be performed.

68581 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

- The *wcstod()* prototype is updated.
- The *wcstof()* and *wcstold()* functions are added.
- If the correct value for *wcstod()* would cause underflow, the return value changed from 0 (as specified in Issue 5) to the smallest normalized positive number.
- The DESCRIPTION, RETURN VALUE, and APPLICATION USAGE sections are extensively updated.

68588 ISO/IEC 9899:1999 standard, Technical Corrigendum 1 is incorporated.

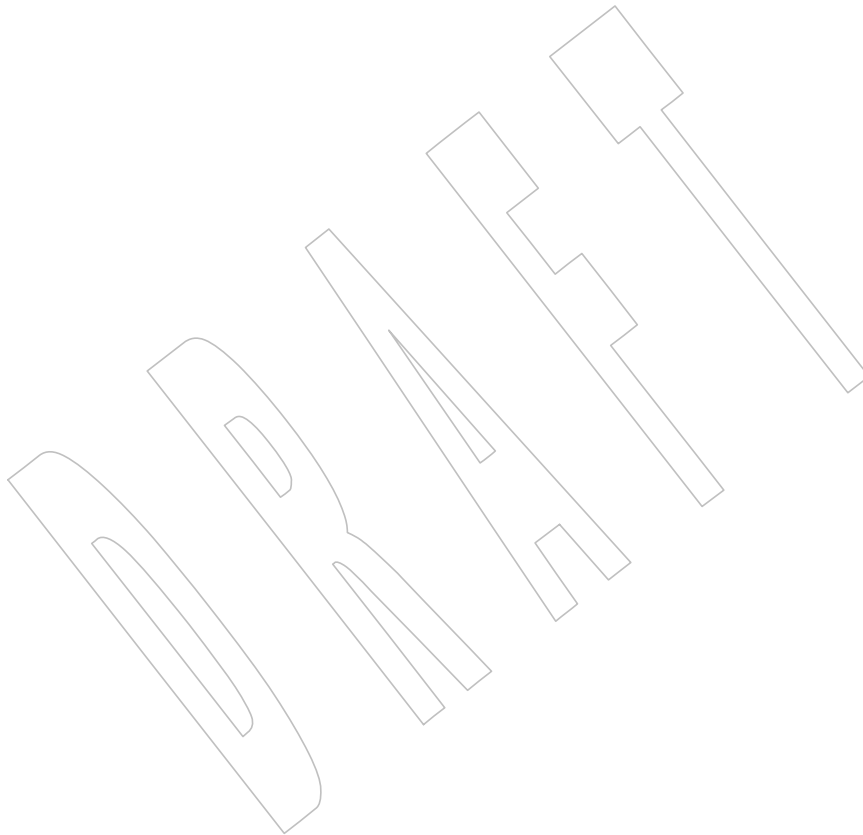
68589 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/66 is applied, correcting the second
68590 paragraph in the RETURN VALUE section.

68591

Issue 7

68592

Austin Group Interpretation 1003.1-2001 #015 is applied.



68593 **NAME**68594 `wcstoimax`, `wcstoumax` — convert a wide-character string to an integer type68595 **SYNOPSIS**68596 `#include <stddef.h>`68597 `#include <inttypes.h>`

```
68598     intmax_t wcstoimax(const wchar_t *restrict nptr,
68599                     wchar_t **restrict endptr, int base);
68600     uintmax_t wcstoumax(const wchar_t *restrict nptr,
68601                       wchar_t **restrict endptr, int base);
```

68602 **DESCRIPTION**

68603 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68604 conflict between the requirements described here and the ISO C standard is unintentional. This
 68605 volume of POSIX.1-200x defers to the ISO C standard.

68606 These functions shall be equivalent to the `wcstol()`, `wcstoll()`, `wcstoul()`, and `wcstoull()` functions,
 68607 respectively, except that the initial portion of the wide string shall be converted to `intmax_t` and
 68608 `uintmax_t` representation, respectively.

68609 **RETURN VALUE**

68610 These functions shall return the converted value, if any.

68611 If no conversion could be performed, zero shall be returned. If the correct value is outside the
 68612 range of representable values, `{INTMAX_MAX}`, `{INTMAX_MIN}`, or `{UINTMAX_MAX}` shall
 68613 be returned (according to the return type and sign of the value, if any), and `errno` shall be set to
 68614 `[ERANGE]`.

68615 **ERRORS**

68616 These functions shall fail if:

68617 `[EINVAL]` The value of `base` is not supported.68618 `[ERANGE]` The value to be returned is not representable.

68619 These functions may fail if:

68620 `[EINVAL]` No conversion could be performed.68621 **EXAMPLES**

68622 None.

68623 **APPLICATION USAGE**

68624 None.

68625 **RATIONALE**

68626 None.

68627 **FUTURE DIRECTIONS**

68628 None.

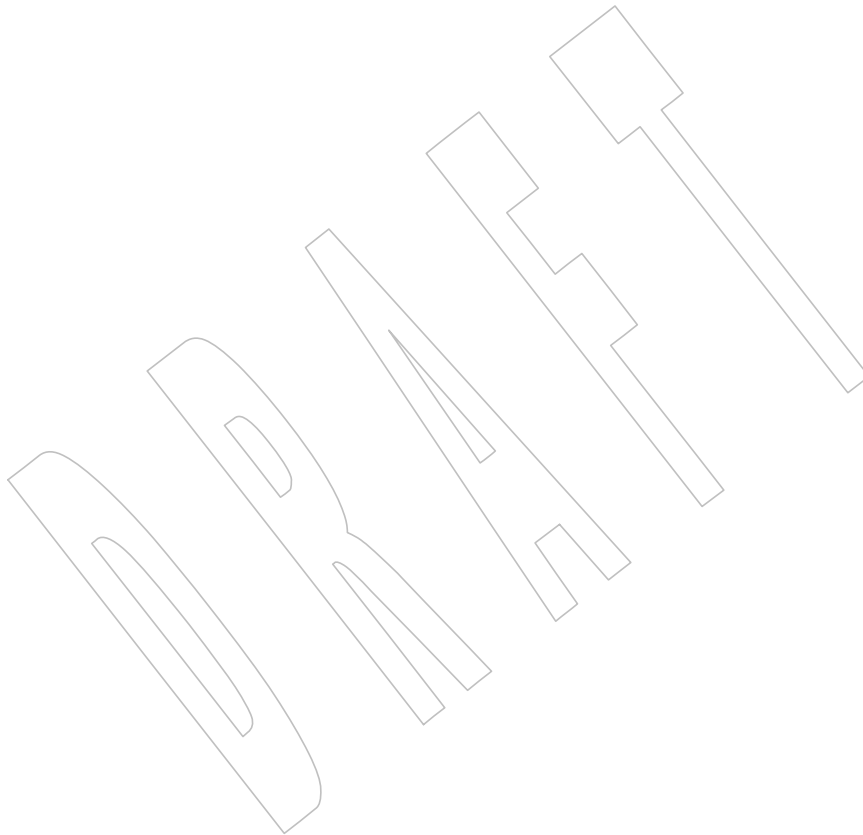
68629 **SEE ALSO**68630 [wcstol\(\)](#), [wcstoul\(\)](#)68631 XBD [<inttypes.h>](#), [<stddef.h>](#)

68632

68633

CHANGE HISTORY

First released in Issue 6. Derived from the ISO/IEC 9899: 1999 standard.



68634 **NAME**
 68635 `wcstok` — split a wide-character string into tokens

68636 **SYNOPSIS**
 68637 `#include <wchar.h>`
 68638 `wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
 68639 `wchar_t **restrict ptr);`

68640 **DESCRIPTION**

68641 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68642 conflict between the requirements described here and the ISO C standard is unintentional. This
 68643 volume of POSIX.1-200x defers to the ISO C standard.

68644 A sequence of calls to `wcstok()` shall break the wide-character string pointed to by `ws1` into a
 68645 sequence of tokens, each of which shall be delimited by a wide-character code from the wide-
 68646 character string pointed to by `ws2`. The `ptr` argument points to a caller-provided `wchar_t` pointer
 68647 into which the `wcstok()` function shall store information necessary for it to continue scanning the
 68648 same wide-character string.

68649 The first call in the sequence has `ws1` as its first argument, and is followed by calls with a null
 68650 pointer as their first argument. The separator string pointed to by `ws2` may be different from call
 68651 to call.

68652 The first call in the sequence shall search the wide-character string pointed to by `ws1` for the first
 68653 wide-character code that is *not* contained in the current separator string pointed to by `ws2`. If no
 68654 such wide-character code is found, then there are no tokens in the wide-character string pointed
 68655 to by `ws1` and `wcstok()` shall return a null pointer. If such a wide-character code is found, it shall
 68656 be the start of the first token.

68657 The `wcstok()` function shall then search from there for a wide-character code that *is* contained in
 68658 the current separator string. If no such wide-character code is found, the current token extends
 68659 to the end of the wide-character string pointed to by `ws1`, and subsequent searches for a token
 68660 shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a
 68661 null wide character, which terminates the current token. The `wcstok()` function shall save a
 68662 pointer to the following wide-character code, from which the next search for a token shall start.

68663 Each subsequent call, with a null pointer as the value of the first argument, shall start searching
 68664 from the saved pointer and behave as described above.

68665 The implementation shall behave as if no function calls `wcstok()`.

68666 **RETURN VALUE**

68667 Upon successful completion, the `wcstok()` function shall return a pointer to the first wide-
 68668 character code of a token. Otherwise, if there is no token, `wcstok()` shall return a null pointer.

68669 **ERRORS**

68670 No errors are defined.

68671

EXAMPLES

68672

None.

68673

APPLICATION USAGE

68674

None.

68675

RATIONALE

68676

None.

68677

FUTURE DIRECTIONS

68678

None.

68679

SEE ALSO

68680

XBD [<wchar.h>](#)

68681

CHANGE HISTORY

68682

First released in Issue 4.

68683

Issue 5

68684

Aligned with ISO/IEC 9899:1990/Amendment 1:1995 (E). Specifically, a third argument is added to the definition of *wcstok()* in the SYNOPSIS.

68685

68686

Issue 6

68687

The *wcstok()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

DRAFT

68688 **NAME**
 68689 `wcstol, wcstoll` — convert a wide-character string to a long integer

68690 **SYNOPSIS**
 68691 `#include <wchar.h>`
 68692 `long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,`
 68693 `int base);`
 68694 `long long wcstoll(const wchar_t *restrict nptr,`
 68695 `wchar_t **restrict endptr, int base);`

68696 **DESCRIPTION**

68697 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68698 conflict between the requirements described here and the ISO C standard is unintentional. This
 68699 volume of POSIX.1-200x defers to the ISO C standard.

68700 These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to
 68701 **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

- 68702 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
 68703 `iswspace()`)
- 68704 2. A subject sequence interpreted as an integer represented in some radix determined by the
 68705 value of *base*
- 68706 3. A final wide-character string of one or more unrecognized wide-character codes,
 68707 including the terminating null wide-character code of the input wide-character string

68708 Then they shall attempt to convert the subject sequence to an integer, and return the result.

68709 If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
 68710 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
 68711 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
 68712 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
 68713 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
 68714 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.

68715 If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence
 68716 of letters and digits representing an integer with the radix specified by *base*, optionally preceded
 68717 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
 68718 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
 68719 than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code
 68720 representations of 0x or 0X may optionally precede the sequence of letters and digits, following
 68721 the sign if present.

68722 The subject sequence is defined as the longest initial subsequence of the input wide-character
 68723 string, starting with the first non-white-space wide-character code that is of the expected form.
 68724 The subject sequence contains no wide-character codes if the input wide-character string is
 68725 empty or consists entirely of white-space wide-character code, or if the first non-white-space
 68726 wide-character code is other than a sign or a permissible letter or digit.

68727 If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes
 68728 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
 68729 the expected form and the value of *base* is between 2 and 36, it shall be used as the base for
 68730 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
 68731 minus sign, the value resulting from the conversion shall be negated. A pointer to the final wide-
 68732 character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a
 68733 null pointer.

68734 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
68735 accepted.

68736 If the subject sequence is empty or does not have the expected form, no conversion shall be
68737 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
68738 *endptr* is not a null pointer.

68739 CX These functions shall not change the setting of *errno* if successful.

68740 Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on
68741 error and are also valid returns on success, an application wishing to check for error situations
68742 should set *errno* to 0, then call *wcstol()* or *wcstoll()*, then check *errno*.

68743 RETURN VALUE

68744 Upon successful completion, these functions shall return the converted value, if any. If no
68745 CX conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If
68746 the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX},
68747 {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and
68748 *errno* set to [ERANGE].

68749 ERRORS

68750 These functions shall fail if:

68751 CX [EINVAL] The value of *base* is not supported.

68752 [ERANGE] The value to be returned is not representable.

68753 These functions may fail if:

68754 CX [EINVAL] No conversion could be performed.

68755 EXAMPLES

68756 None.

68757 APPLICATION USAGE

68758 None.

68759 RATIONALE

68760 None.

68761 FUTURE DIRECTIONS

68762 None.

68763 SEE ALSO

68764 *fscanf()*, *iswalpha()*, *wcstod()*

68765 XBD <wchar.h>

68766 CHANGE HISTORY

68767 First released in Issue 4. Derived from the MSE working draft.

68768 Issue 5

68769 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.

68770 Issue 6

68771 Extensions beyond the ISO C standard are marked.

68772 The following new requirements on POSIX implementations derive from alignment with the
68773 Single UNIX Specification:

- 68774 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
68775 added if no conversion could be performed.

68776 The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

wcstol()

68777

- The *wcstol()* prototype is updated.

68778

- The *wcstoll()* function is added.

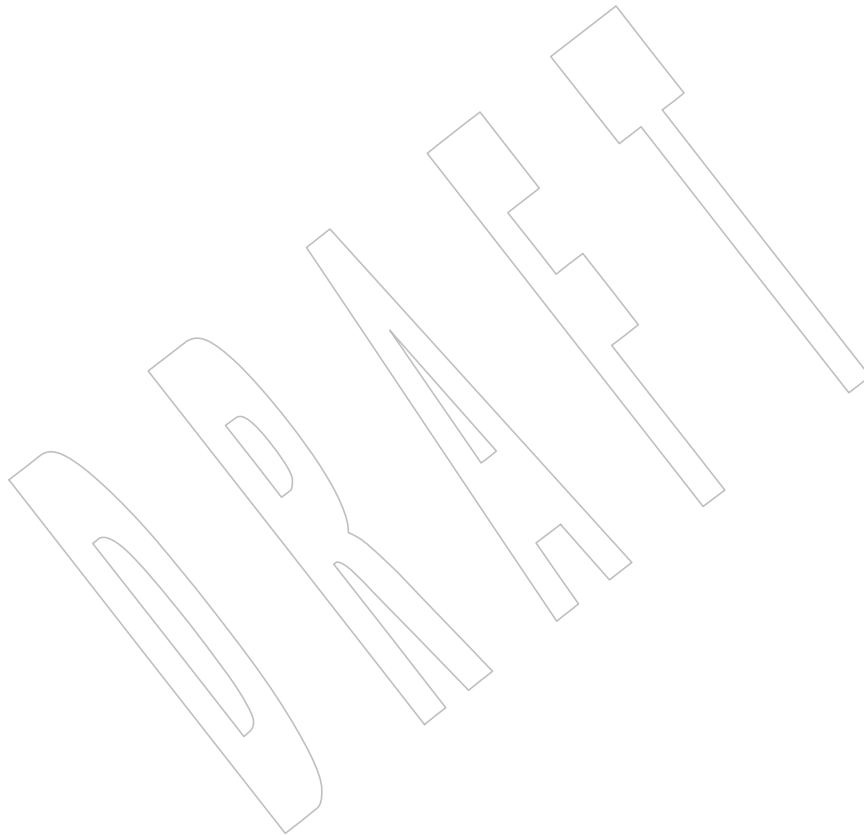
68779

Issue 7

68780

SD5-XSH-ERN-56 is applied, removing the reference to **unsigned long** and **unsigned long long** from the DESCRIPTION.

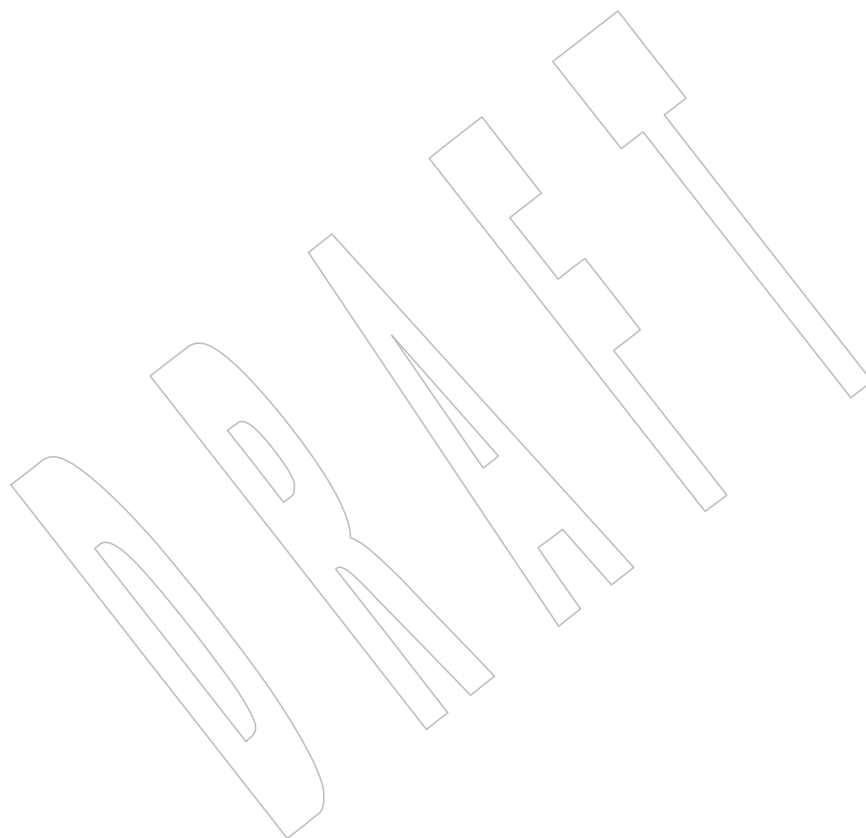
68781



68782 **NAME**
68783 `wcstold` — convert a wide-character string to a double-precision number

68784 **SYNOPSIS**
68785 `#include <wchar.h>`
68786 `long double wcstold(const wchar_t *restrict nptr,`
68787 `wchar_t **restrict endptr);`

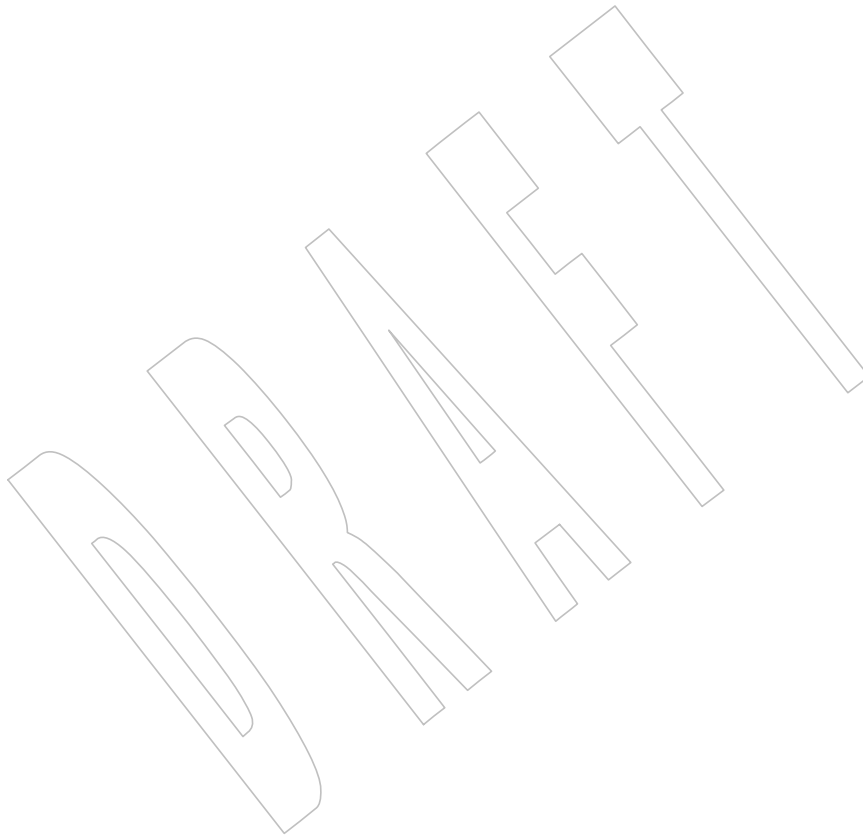
68788 **DESCRIPTION**
68789 Refer to *wcstod()*.



68790 **NAME**
68791 `wcstoll` — convert a wide-character string to a long integer

68792 **SYNOPSIS**
68793 `#include <wchar.h>`
68794 `long long wcstoll(const wchar_t *restrict nptr,`
68795 `wchar_t **restrict endptr, int base);`

68796 **DESCRIPTION**
68797 Refer to *wcstol()*.



68798 **NAME**
 68799 `wcstombs` — convert a wide-character string to a character string

68800 **SYNOPSIS**
 68801 `#include <stdlib.h>`
 68802 `size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,`
 68803 `size_t n);`

68804 **DESCRIPTION**

68805 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 68806 conflict between the requirements described here and the ISO C standard is unintentional. This
 68807 volume of POSIX.1-200x defers to the ISO C standard.

68808 The `wcstombs()` function shall convert the sequence of wide-character codes that are in the array
 68809 pointed to by `pwcs` into a sequence of characters that begins in the initial shift state and store
 68810 these characters into the array pointed to by `s`, stopping if a character would exceed the limit of `n`
 68811 total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to
 68812 `wctomb()`, except that the shift state of `wctomb()` shall not be affected.

68813 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

68814 No more than `n` bytes shall be modified in the array pointed to by `s`. If copying takes place
 68815 CX between objects that overlap, the behavior is undefined. If `s` is a null pointer, `wcstombs()` shall
 68816 return the length required to convert the entire array regardless of the value of `n`, but no values
 68817 are stored.

68818 The `wcstombs()` function need not be thread-safe. A function that is not required to be thread-
 68819 safe is not required to be reentrant.

68820 **RETURN VALUE**

68821 If a wide-character code is encountered that does not correspond to a valid character (of one or
 68822 more bytes each), `wcstombs()` shall return `(size_t)-1`. Otherwise, `wcstombs()` shall return the
 68823 number of bytes stored in the character array, not including any terminating null byte. The array
 68824 shall not be null-terminated if the value returned is `n`.

68825 **ERRORS**

68826 The `wcstombs()` function may fail if:

68827 CX **[EILSEQ]** A wide-character code does not correspond to a valid character.

68828 **EXAMPLES**

68829 None.

68830 **APPLICATION USAGE**

68831 None.

68832 **RATIONALE**

68833 None.

68834 **FUTURE DIRECTIONS**

68835 None.

68836 **SEE ALSO**

68837 [`mblen\(\)`](#), [`mbtowc\(\)`](#), [`mbstowcs\(\)`](#), [`wctomb\(\)`](#)

68838 XBD [`<stdlib.h>`](#)

68839

CHANGE HISTORY

68840

First released in Issue 4. Derived from the ISO C standard.

68841

Issue 6

68842

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

68843

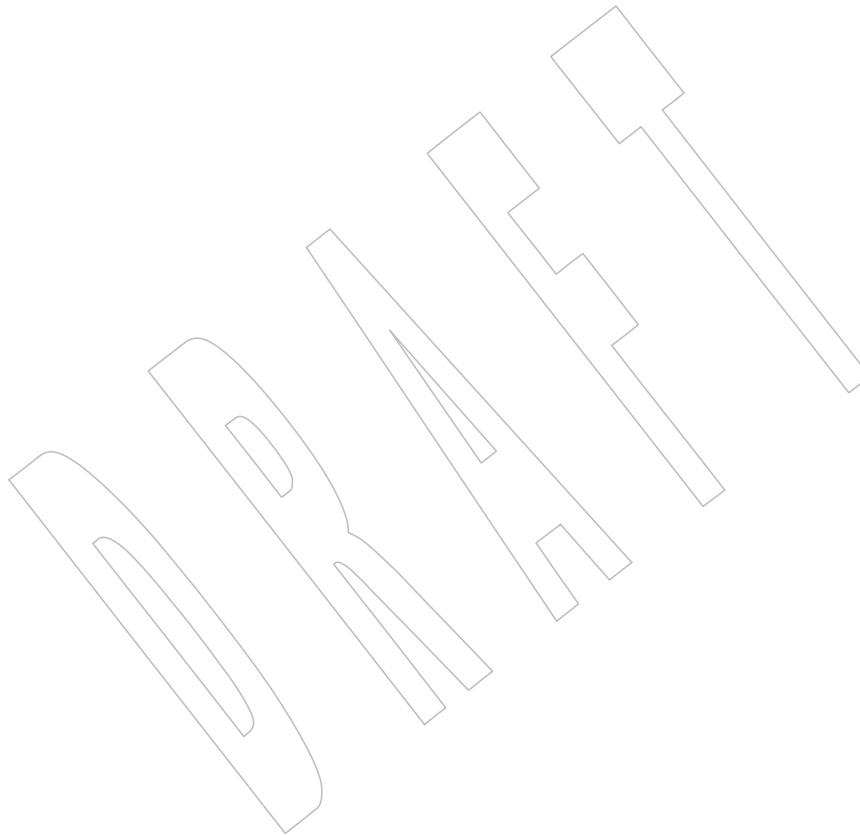
- The DESCRIPTION states the effect of when *s* is a null pointer.
- The [EILSEQ] error condition is added.

68844

68845

68846

The *wcstombs()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.



68847 **NAME**68848 `wcstoul`, `wcstoull` — convert a wide-character string to an unsigned long68849 **SYNOPSIS**68850 `#include <wchar.h>`68851 `unsigned long wcstoul(const wchar_t *restrict nptr,`
68852 `wchar_t **restrict endptr, int base);`68853 `unsigned long long wcstoull(const wchar_t *restrict nptr,`
68854 `wchar_t **restrict endptr, int base);`68855 **DESCRIPTION**68856 CX The functionality described on this reference page is aligned with the ISO C standard. Any
68857 conflict between the requirements described here and the ISO C standard is unintentional. This
68858 volume of POSIX.1-200x defers to the ISO C standard.68859 The `wcstoul()` and `wcstoull()` functions shall convert the initial portion of the wide-character
68860 string pointed to by `nptr` to **unsigned long** and **unsigned long long** representation, respectively.
68861 First, they shall decompose the input wide-character string into three parts:

- 68862 1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by
-
- 68863
- `iswspace()`
-)
-
- 68864 2. A subject sequence interpreted as an integer represented in some radix determined by the
-
- 68865 value of
- `base`
-
- 68866 3. A final wide-character string of one or more unrecognized wide-character codes,
-
- 68867 including the terminating null wide-character code of the input wide-character string

68868 Then they shall attempt to convert the subject sequence to an unsigned integer, and return the
68869 result.68870 If `base` is 0, the expected form of the subject sequence is that of a decimal constant, octal constant,
68871 or hexadecimal constant, any of which may be preceded by a '+' or '-' sign. A decimal
68872 constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal
68873 constant consists of the prefix '0' optionally followed by a sequence of the digits '0' to '7'
68874 only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the
68875 decimal digits and letters 'a' (or 'A') to 'f' (or 'F') with values 10 to 15 respectively.68876 If the value of `base` is between 2 and 36, the expected form of the subject sequence is a sequence
68877 of letters and digits representing an integer with the radix specified by `base`, optionally preceded
68878 by a '+' or '-' sign, but not including an integer suffix. The letters from 'a' (or 'A') to 'z'
68879 (or 'Z') inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less
68880 than that of `base` shall be permitted. If the value of `base` is 16, the wide-character codes 0x or 0X
68881 may optionally precede the sequence of letters and digits, following the sign if present.68882 The subject sequence is defined as the longest initial subsequence of the input wide-character
68883 string, starting with the first wide-character code that is not white space and is of the expected
68884 form. The subject sequence contains no wide-character codes if the input wide-character string is
68885 empty or consists entirely of white-space wide-character codes, or if the first wide-character
68886 code that is not white space is other than a sign or a permissible letter or digit.68887 If the subject sequence has the expected form and `base` is 0, the sequence of wide-character codes
68888 starting with the first digit shall be interpreted as an integer constant. If the subject sequence has
68889 the expected form and the value of `base` is between 2 and 36, it shall be used as the base for
68890 conversion, ascribing to each letter its value as given above. If the subject sequence begins with a
68891 minus sign, the value resulting from the conversion shall be negated. A pointer to the final wide-
68892 character string shall be stored in the object pointed to by `endptr`, provided that `endptr` is not a

wcstoul()

68893 null pointer.

68894 CX In other than the C or POSIX locales, other implementation-defined subject sequences may be
68895 accepted.

68896 If the subject sequence is empty or does not have the expected form, no conversion shall be
68897 performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that
68898 *endptr* is not a null pointer.

68899 CX The *wcstoul()* function shall not change the setting of *errno* if successful.

68900 Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return
68901 on success, an application wishing to check for error situations should set *errno* to 0, then call
68902 *wcstoul()* or *wcstoull()*, then check *errno*.

RETURN VALUE

68903 Upon successful completion, the *wcstoul()* and *wcstoull()* functions shall return the converted
68904 value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to
68905 indicate the error. If the correct value is outside the range of representable values,
68906 {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to [ERANGE].
68907

ERRORS

68908 These functions shall fail if:

68909 CX [EINVAL] The value of *base* is not supported.

68910 [ERANGE] The value to be returned is not representable.

68911 These functions may fail if:

68912 CX [EINVAL] No conversion could be performed.

EXAMPLES

68914 None.
68915

APPLICATION USAGE

68916 None.
68917

RATIONALE

68918 None.
68919

FUTURE DIRECTIONS

68920 None.
68921

SEE ALSO

68922 *fscanf()*, *iswalphalpha()*, *wcstod()*, *wcstol()*

68923 XBD <[wchar.h](#)>

CHANGE HISTORY

68924 First released in Issue 4. Derived from the MSE working draft.

Issue 5

68925 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.
68926

Issue 6

68927 Extensions beyond the ISO C standard are marked.
68928

68929 The following new requirements on POSIX implementations derive from alignment with the
68930 Single UNIX Specification:

- 68931 • The [EINVAL] error condition is added for when the value of *base* is not supported.

68932 In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
68933 added if no conversion could be performed.
68934
68935

68936

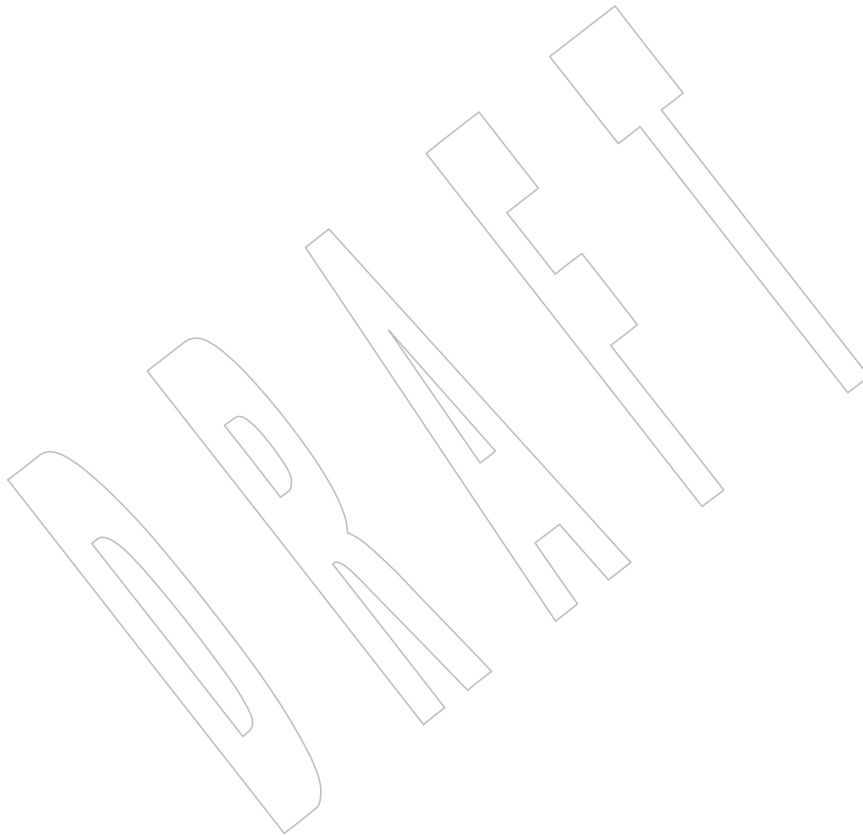
The following changes are made for alignment with the ISO/IEC 9899:1999 standard:

68937

- The *wcstoul()* prototype is updated.

68938

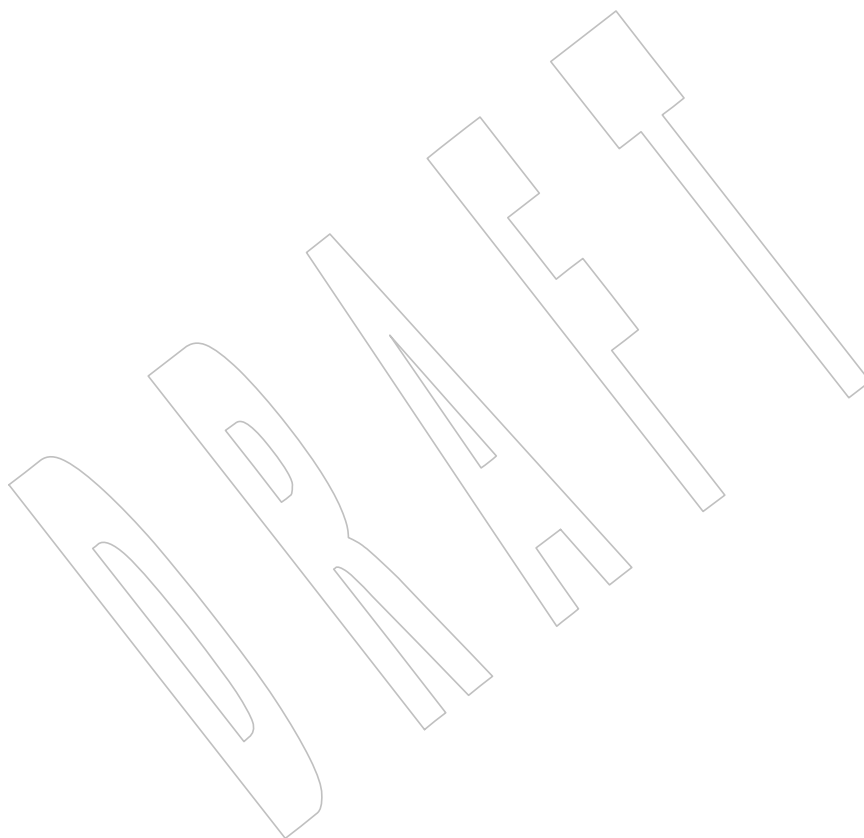
- The *wcstoull()* function is added.



68939 **NAME**
68940 `wcstoumax` — convert a wide-character string to an integer type

68941 **SYNOPSIS**
68942 `#include <stddef.h>`
68943 `#include <inttypes.h>`
68944 `uintmax_t wcstoumax(const wchar_t *restrict nptr,`
68945 `wchar_t **restrict endptr, int base);`

68946 **DESCRIPTION**
68947 Refer to [wcstoimax\(\)](#).



68948 **NAME**

68949 wcswidth — number of column positions of a wide-character string

68950 **SYNOPSIS**

```
68951 XSI #include <wchar.h>
68952 int wcswidth(const wchar_t *pwcs, size_t n);
```

68953 **DESCRIPTION**

68954 The *wcswidth()* function shall determine the number of column positions required for *n* wide-
 68955 character codes (or fewer than *n* wide-character codes if a null wide-character code is
 68956 encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

68957 **RETURN VALUE**

68958 The *wcswidth()* function either shall return 0 (if *pwcs* points to a null wide-character code), or
 68959 return the number of column positions to be occupied by the wide-character string pointed to by
 68960 *pwcs*, or return -1 (if any of the first *n* wide-character codes in the wide-character string pointed
 68961 to by *pwcs* is not a printable wide-character code).

68962 **ERRORS**

68963 No errors are defined.

68964 **EXAMPLES**

68965 None.

68966 **APPLICATION USAGE**

68967 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
 68968 return value for a non-printable wide character is not specified.

68969 **RATIONALE**

68970 None.

68971 **FUTURE DIRECTIONS**

68972 None.

68973 **SEE ALSO**

68974 [wctype\(\)](#)
 68975 XBD [Section 3.102](#) (on page 47), [<wchar.h>](#)

68976 **CHANGE HISTORY**

68977 First released in Issue 4. Derived from the MSE working draft.

68978 **Issue 6**

68979 The Open Group Corrigendum U021/11 is applied. The function is marked as an extension.

68980 **NAME**68981 `wcsxfrm`, `wcsxfrm_l` — wide-character string transformation68982 **SYNOPSIS**68983

```
#include <wchar.h>
```

68984

```
size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,  
size_t n);
```

68985

```
CX size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,  
68987 size_t n, locale_t locale);
```

68988 **DESCRIPTION**68989 CX For `wcsxfrm()`: The functionality described on this reference page is aligned with the ISO C
68990 standard. Any conflict between the requirements described here and the ISO C standard is
68991 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.68992 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall transform the wide-character string pointed to
68993 by `ws2` and place the resulting wide-character string into the array pointed to by `ws1`. The
68994 transformation shall be such that if `wcscmp()` is applied to two transformed wide strings, it shall
68995 return a value greater than, equal to, or less than 0, corresponding to the result of `wscoll()` and
68996 `wscoll_l()` applied to the same two original wide-character strings, and the same `LC_COLLATE`
68997 category of the locale of the process or the locale object `locale`, respectively. No more than `n`
68998 wide-character codes shall be placed into the resulting array pointed to by `ws1`, including the
68999 terminating null wide-character code. If `n` is 0, `ws1` is permitted to be a null pointer. If copying
69000 takes place between objects that overlap, the behavior is undefined.69001 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall not change the setting of `errno` if successful.69002 Since no return value is reserved to indicate an error, an application wishing to check for error
69003 situations should set `errno` to 0, then call `wcsxfrm()` or `wcsxfrm_l()`, then check `errno`.69004 **RETURN VALUE**69005 CX The `wcsxfrm()` and `wcsxfrm_l()` functions shall return the length of the transformed wide-
69006 character string (not including the terminating null wide-character code). If the value returned is
69007 `n` or more, the contents of the array pointed to by `ws1` are unspecified.69008 CX On error, the `wcsxfrm()` and `wcsxfrm_l()` functions may set `errno`, but no return value is reserved
69009 to indicate an error.69010 **ERRORS**

69011 These functions may fail if:

69012 CX [EINVAL] The wide-character string pointed to by `ws2` contains wide-character codes
69013 outside the domain of the collating sequence.69014 The `wcsxfrm_l()` function may fail if:69015 CX [EINVAL] `locale` is not a valid locale object handle.

69016 **EXAMPLES**

69017 None.

69018 **APPLICATION USAGE**

69019 The transformation function is such that two transformed wide-character strings can be ordered
 69020 by *wscmp()* as appropriate to collating sequence information in the locale of the process
 69021 (category *LC_COLLATE*).

69022 The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of
 69023 the *ws1* array prior to making the transformation.

69024 **RATIONALE**

69025 None.

69026 **FUTURE DIRECTIONS**

69027 None.

69028 **SEE ALSO**69029 *wscmp()*, *wscoll()*69030 XBD <*wchar.h*>69031 **CHANGE HISTORY**

69032 First released in Issue 4. Derived from the MSE working draft.

69033 **Issue 5**

69034 Moved from ENHANCED I18N to BASE and the [ENOSYS] error is removed.

69035 The DESCRIPTION is updated to indicate that *errno* is not changed if the function is successful.69036 **Issue 6**69037 In earlier versions, this function was required to return *-1* on error.

69038 Extensions beyond the ISO C standard are marked.

69039 The following new requirements on POSIX implementations derive from alignment with the
 69040 Single UNIX Specification:

- 69041 • In the RETURN VALUE and ERRORS sections, the [EINVAL] optional error condition is
 69042 added if no conversion could be performed.

69043 The *wcsxfrm()* prototype is updated for alignment with the ISO/IEC 9899:1999 standard.69044 **Issue 7**

69045 The *wcsxfrm_l()* function is added from The Open Group Technical Standard, 2006, Extended
 69046 API Set Part 4.

69047 **NAME**
 69048 `wctob` — wide-character to single-byte conversion

69049 **SYNOPSIS**
 69050 `#include <stdio.h>`
 69051 `#include <wchar.h>`
 69052 `int wctob(wint_t c);`

69053 **DESCRIPTION**
 69054 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69055 conflict between the requirements described here and the ISO C standard is unintentional. This
 69056 volume of POSIX.1-200x defers to the ISO C standard.

69057 The `wctob()` function shall determine whether `c` corresponds to a member of the extended
 69058 character set whose character representation is a single byte when in the initial shift state.

69059 The behavior of this function shall be affected by the `LC_CTYPE` category of the current locale.

69060 **RETURN VALUE**
 69061 The `wctob()` function shall return EOF if `c` does not correspond to a character with length one in
 69062 the initial shift state. Otherwise, it shall return the single-byte representation of that character as
 69063 an **unsigned char** converted to **int**.

69064 **ERRORS**
 69065 No errors are defined.

69066 **EXAMPLES**
 69067 None.

69068 **APPLICATION USAGE**
 69069 None.

69070 **RATIONALE**
 69071 None.

69072 **FUTURE DIRECTIONS**
 69073 None.

69074 **SEE ALSO**
 69075 [*btowc\(\)*](#)
 69076 XBD [*<wchar.h>*](#)

69077 **CHANGE HISTORY**
 69078 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69079 (E).

69080 **NAME**
 69081 wctomb — convert a wide-character code to a character

69082 **SYNOPSIS**
 69083 #include <stdlib.h>

69084 int wctomb(char *s, wchar_t wchar);

69085 DESCRIPTION

69086 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69087 conflict between the requirements described here and the ISO C standard is unintentional. This
 69088 volume of POSIX.1-200x defers to the ISO C standard.

69089 The *wctomb()* function shall determine the number of bytes needed to represent the character
 69090 corresponding to the wide-character code whose value is *wchar* (including any change in the
 69091 shift state). It shall store the character representation (possibly multiple bytes and any special
 69092 bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most
 69093 {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any
 69094 shift sequence needed to restore the initial shift state, and *wctomb()* shall be left in the initial shift
 69095 state.

69096 CX The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a
 69097 state-dependent encoding, this function shall be placed into its initial state by a call for which its
 69098 character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null
 69099 pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a
 69100 null pointer shall cause this function to return a non-zero value if encodings have state
 69101 dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this
 69102 function to be unspecified.

69103 The *wctomb()* function need not be thread-safe. A function that is not required to be thread-safe
 69104 is not required to be reentrant.

69105 The implementation shall behave as if no function defined in this volume of POSIX.1-200x calls
 69106 *wctomb()*.

69107 RETURN VALUE

69108 If *s* is a null pointer, *wctomb()* shall return a non-zero or 0 value, if character encodings,
 69109 respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb()*
 69110 shall return -1 if the value of *wchar* does not correspond to a valid character, or return the
 69111 number of bytes that constitute the character corresponding to the value of *wchar*.

69112 In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

69113 ERRORS

69114 No errors are defined.

69115 EXAMPLES

69116 None.

69117 APPLICATION USAGE

69118 None.

69119 RATIONALE

69120 None.

wctomb()

69121

FUTURE DIRECTIONS

69122

None.

69123

SEE ALSO

69124

mblen(), *mbtowc()*, *mbstowcs()*, *wcstombs()*

69125

XBD <stdlib.h>

69126

CHANGE HISTORY

69127

First released in Issue 4. Derived from the ANSI C standard.

69128

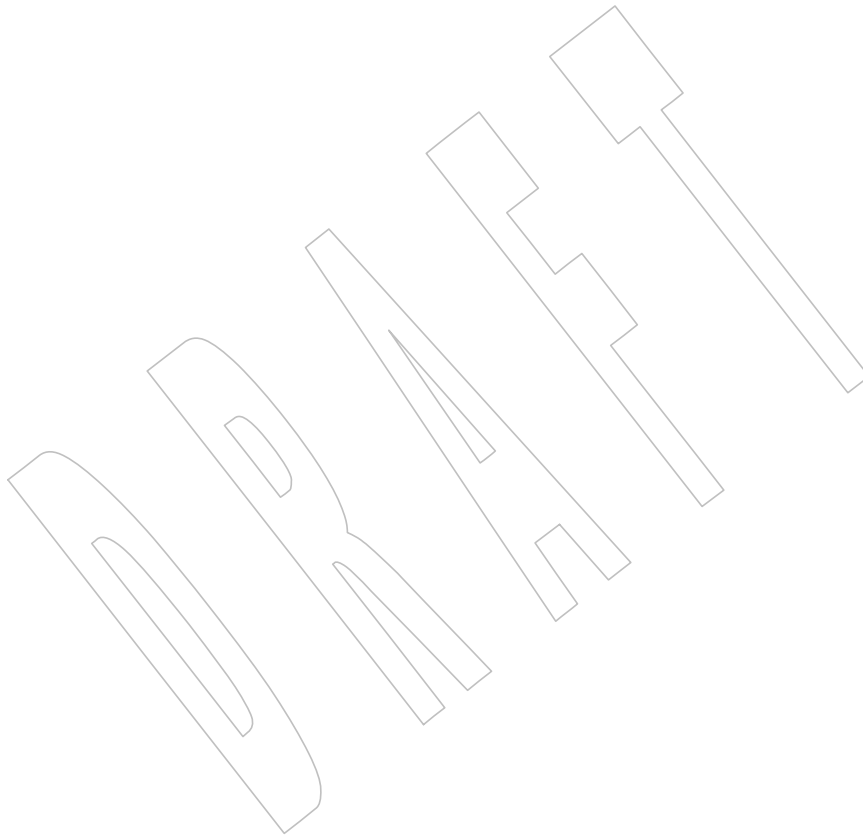
Issue 6

69129

Extensions beyond the ISO C standard are marked.

69130

In the DESCRIPTION, a note about reentrancy and thread-safety is added.



69131 **NAME**

69132 wctrans, wctrans_l — define character mapping

69133 **SYNOPSIS**

69134 #include <wctype.h>

69135 wctrans_t wctrans(const char *charclass);

69136 CX wctrans_t wctrans_l(const char *charclass, locale_t locale);

69137 **DESCRIPTION**69138 CX For *wctrans()*: The functionality described on this reference page is aligned with the ISO C
69139 standard. Any conflict between the requirements described here and the ISO C standard is
69140 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.69141 CX The *wctrans()* and *wctrans_l()* functions are defined for valid character mapping names
69142 identified in the current locale. The *charclass* is a string identifying a generic character mapping
69143 name for which codeset-specific information is required. The following character mapping
69144 names are defined in all locales: **tolower** and **toupper**.69145 These functions shall return a value of type **wctrans_t**, which can be used as the second
69146 argument to subsequent calls of *towctrans()* and *towctrans_l()*.69147 CX The *wctrans()* and *wctrans_l()* functions shall determine values of **wctrans_t** according to the
69148 rules of the coded character set defined by character mapping information in the locale of the
69149 process or in the locale represented by *locale*, respectively (category *LC_CTYPE*).69150 The values returned by *wctrans()* shall be valid until a call to *setlocale()* that modifies the
69151 category *LC_CTYPE*.69152 CX The values returned by *wctrans_l()* shall be valid only in calls to *wctrans_l()* with a locale
69153 represented by *locale* with the same *LC_CTYPE* category value.69154 **RETURN VALUE**69155 CX The *wctrans()* and *wctrans_l()* functions shall return 0 and may set *errno* to indicate the error if
69156 the given character mapping name is not valid for the current locale (category *LC_CTYPE*);
69157 otherwise, they shall return a non-zero object of type **wctrans_t** that can be used in calls to
69158 *towctrans()* and *towctrans_l()*.69159 **ERRORS**

69160 These functions may fail if:

69161 CX [EINVAL] The character mapping name pointed to by *charclass* is not valid in the current
69162 locale.69163 The *wctrans_l()* function may fail if:69164 CX [EINVAL] *locale* is not a valid locale object handle.69165 **EXAMPLES**

69166 None.

69167 **APPLICATION USAGE**

69168 None.

69169 **RATIONALE**

69170 None.

69171

FUTURE DIRECTIONS

69172

None.

69173

SEE ALSO

69174

[towctrans\(\)](#)

69175

XBD [<wctype.h>](#)

69176

CHANGE HISTORY

69177

First released in Issue 5. Derived from ISO/IEC 9899:1990/Amendment 1:1995 (E).

69178

Issue 7

69179

The *wctrans_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

69180



69181 **NAME**
 69182 `wctype, wctype_l` — define character class

69183 **SYNOPSIS**

69184 `#include <wctype.h>`
 69185 `wctype_t wctype(const char *property);`
 69186 CX `wctype_t wctype_l(const char *property, locale_t locale);`

69187 **DESCRIPTION**

69188 CX For `wctype()`: The functionality described on this reference page is aligned with the ISO C
 69189 standard. Any conflict between the requirements described here and the ISO C standard is
 69190 unintentional. This volume of POSIX.1-200x defers to the ISO C standard.

69191 CX The `wctype()` and `wctype_l()` functions are defined for valid character class names as defined in
 69192 CX the current locale or in the locale represented by `locale`, respectively.

69193 The `property` argument is a string identifying a generic character class for which codeset-specific
 69194 type information is required. The following character class names shall be defined in all locales:

69195	alnum	digit	punct
69196	alpha	graph	space
69197	blank	lower	upper
69198	cntrl	print	xdigit

69199 Additional character class names defined in the locale definition file (category `LC_CTYPE`) can
 69200 also be specified.

69201 These functions shall return a value of type `wctype_t`, which can be used as the second
 69202 CX argument to subsequent calls of `iswctype()` and `iswctype_l()`.

69203 CX The `wctype()` and `wctype_l()` functions shall determine values of `wctype_t` according to the
 69204 rules of the coded character set defined by character type information in the locale of the process
 69205 CX or in the locale represented by `locale`, respectively (category `LC_CTYPE`).

69206 The values returned by `wctype()` shall be valid until a call to `setlocale()` that modifies the category
 69207 `LC_CTYPE`.

69208 CX The values returned by `wctype_l()` shall be valid only in calls to `iswctype_l()` with a locale
 69209 represented by `locale` with the same `LC_CTYPE` category value.

69210 **RETURN VALUE**

69211 CX The `wctype()` and `wctype_l()` functions shall return 0 if the given character class name is not
 69212 valid for the current locale (category `LC_CTYPE`); otherwise, they shall return an object of type
 69213 CX `wctype_t` that can be used in calls to `iswctype()` and `iswctype_l()`.

69214 **ERRORS**

69215 The `wctype_l()` function may fail if:

69216 CX [EINVAL] `locale` is not a valid locale object handle.

wctype()

69217

EXAMPLES

69218

None.

69219

APPLICATION USAGE

69220

None.

69221

RATIONALE

69222

None.

69223

FUTURE DIRECTIONS

69224

None.

69225

SEE ALSO

69226

iswctype()

69227

XBD **<wctype.h>**

69228

CHANGE HISTORY

69229

First released in Issue 4.

69230

Issue 5

69231

The following change has been made in this version for alignment with ISO/IEC 9899:1990/Amendment 1:1995 (E):

69232

- The SYNOPSIS has been changed to indicate that this function and associated data types are now made visible by inclusion of the **<wctype.h>** header rather than **<wchar.h>**.

69233

69234

69235

Issue 7

69236

The *wctype_l()* function is added from The Open Group Technical Standard, 2006, Extended API Set Part 4.

69237

69238 **NAME**
 69239 `wcwidth` — number of column positions of a wide-character code

69240 **SYNOPSIS**

```
69241 XSI #include <wchar.h>
69242 int wcwidth(wchar_t wc);
```

69243 **DESCRIPTION**

69244 The `wcwidth()` function shall determine the number of column positions required for the wide
 69245 character `wc`. The application shall ensure that the value of `wc` is a character representable as a
 69246 `wchar_t`, and is a wide-character code corresponding to a valid character in the current locale.

69247 **RETURN VALUE**

69248 The `wcwidth()` function shall either return 0 (if `wc` is a null wide-character code), or return the
 69249 number of column positions to be occupied by the wide-character code `wc`, or return -1 (if `wc`
 69250 does not correspond to a printable wide-character code).

69251 **ERRORS**

69252 No errors are defined.

69253 **EXAMPLES**

69254 None.

69255 **APPLICATION USAGE**

69256 This function was removed from the final ISO/IEC 9899:1990/Amendment 1:1995 (E), and the
 69257 return value for a non-printable wide character is not specified.

69258 **RATIONALE**

69259 None.

69260 **FUTURE DIRECTIONS**

69261 None.

69262 **SEE ALSO**

69263 [*wcswidth\(\)*](#)
 69264 XBD [**<wchar.h>**](#)

69265 **CHANGE HISTORY**

69266 First released as a World-wide Portability Interface in Issue 4. Derived from the MSE working
 69267 draft.

69268 **Issue 6**

69269 The Open Group Corrigendum U021/12 is applied. This function is marked as an extension.

69270 The normative text is updated to avoid use of the term “must” for application requirements.

69271 **NAME**
 69272 `wmemchr` — find a wide character in memory

69273 **SYNOPSIS**
 69274 `#include <wchar.h>`

69275 `wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);`

69276 **DESCRIPTION**

69277 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69278 conflict between the requirements described here and the ISO C standard is unintentional. This
 69279 volume of POSIX.1-200x defers to the ISO C standard.

69280 The `wmemchr()` function shall locate the first occurrence of `wc` in the initial `n` wide characters of
 69281 the object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values
 69282 shall be treated identically. The null wide character and `wchar_t` values not corresponding to
 69283 valid characters shall not be treated specially.

69284 If `n` is zero, the application shall ensure that `ws` is a valid pointer and the function behaves as if
 69285 no valid occurrence of `wc` is found.

69286 **RETURN VALUE**

69287 The `wmemchr()` function shall return a pointer to the located wide character, or a null pointer if
 69288 the wide character does not occur in the object.

69289 **ERRORS**

69290 No errors are defined.

69291 **EXAMPLES**

69292 None.

69293 **APPLICATION USAGE**

69294 None.

69295 **RATIONALE**

69296 None.

69297 **FUTURE DIRECTIONS**

69298 None.

69299 **SEE ALSO**

69300 [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)

69301 XBD [<wchar.h>](#)

69302 **CHANGE HISTORY**

69303 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69304 (E).

69305 **Issue 6**

69306 The normative text is updated to avoid use of the term “must” for application requirements.

69307 **NAME**
 69308 wmemcmp — compare wide characters in memory

69309 **SYNOPSIS**

69310 #include <wchar.h>

69311 int wmemcmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);

69312 **DESCRIPTION**

69313 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69314 conflict between the requirements described here and the ISO C standard is unintentional. This
 69315 volume of POSIX.1-200x defers to the ISO C standard.

69316 The *wmemcmp()* function shall compare the first *n* wide characters of the object pointed to by
 69317 *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be
 69318 affected by locale and all **wchar_t** values shall be treated identically. The null wide character and
 69319 **wchar_t** values not corresponding to valid characters shall not be treated specially.

69320 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
 69321 shall behave as if the two objects compare equal.

69322 **RETURN VALUE**

69323 The *wmemcmp()* function shall return an integer greater than, equal to, or less than zero,
 69324 respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object
 69325 pointed to by *ws2*.

69326 **ERRORS**

69327 No errors are defined.

69328 **EXAMPLES**

69329 None.

69330 **APPLICATION USAGE**

69331 None.

69332 **RATIONALE**

69333 None.

69334 **FUTURE DIRECTIONS**

69335 None.

69336 **SEE ALSO**

69337 [wmemchr\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)

69338 XBD [<wchar.h>](#)

69339 **CHANGE HISTORY**

69340 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69341 (E).

69342 **Issue 6**

69343 The normative text is updated to avoid use of the term “must” for application requirements.

69344 **NAME**
 69345 `wmemcpy` — copy wide characters in memory

69346 **SYNOPSIS**
 69347 `#include <wchar.h>`

69348 `wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,`
 69349 `size_t n);`

69350 DESCRIPTION

69351 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69352 conflict between the requirements described here and the ISO C standard is unintentional. This
 69353 volume of POSIX.1-200x defers to the ISO C standard.

69354 The `wmemcpy()` function shall copy *n* wide characters from the object pointed to by *ws2* to the
 69355 object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values
 69356 shall be treated identically. The null wide character and **wchar_t** values not corresponding to
 69357 valid characters shall not be treated specially.

69358 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
 69359 shall copy zero wide characters.

69360 RETURN VALUE

69361 The `wmemcpy()` function shall return the value of *ws1*.

69362 ERRORS

69363 No errors are defined.

69364 EXAMPLES

69365 None.

69366 APPLICATION USAGE

69367 None.

69368 RATIONALE

69369 None.

69370 FUTURE DIRECTIONS

69371 None.

69372 SEE ALSO

69373 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemmove\(\)](#), [wmemset\(\)](#)

69374 XBD [<wchar.h>](#)

69375 CHANGE HISTORY

69376 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69377 (E).

69378 Issue 6

69379 The normative text is updated to avoid use of the term “must” for application requirements.

69380 The `wmemcpy()` prototype is updated for alignment with the ISO/IEC 9899:1999 standard.

69381 **NAME**
 69382 wmemmove — copy wide characters in memory with overlapping areas

69383 **SYNOPSIS**
 69384 #include <wchar.h>

69385 wchar_t *wmemmove(wchar_t *ws1, const wchar_t *ws2, size_t n);

69386 DESCRIPTION

69387 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69388 conflict between the requirements described here and the ISO C standard is unintentional. This
 69389 volume of POSIX.1-200x defers to the ISO C standard.

69390 The *wmemmove()* function shall copy *n* wide characters from the object pointed to by *ws2* to the
 69391 object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object
 69392 pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not
 69393 overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary
 69394 array are copied into the object pointed to by *ws1*.

69395 This function shall not be affected by locale and all **wchar_t** values shall be treated identically.
 69396 The null wide character and **wchar_t** values not corresponding to valid characters shall not be
 69397 treated specially.

69398 If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function
 69399 shall copy zero wide characters.

69400 RETURN VALUE

69401 The *wmemmove()* function shall return the value of *ws1*.

69402 ERRORS

69403 No errors are defined

69404 EXAMPLES

69405 None.

69406 APPLICATION USAGE

69407 None.

69408 RATIONALE

69409 None.

69410 FUTURE DIRECTIONS

69411 None.

69412 SEE ALSO

69413 *wmemchr()*, *wmemcmp()*, *wmemcpy()*, *wmemset()*

69414 XBD <wchar.h>

69415 CHANGE HISTORY

69416 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69417 (E).

69418 Issue 6

69419 The normative text is updated to avoid use of the term “must” for application requirements.

69420 **NAME**
 69421 `wmemset` — set wide characters in memory

69422 **SYNOPSIS**
 69423 `#include <wchar.h>`
 69424 `wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);`

69425 **DESCRIPTION**
 69426 CX The functionality described on this reference page is aligned with the ISO C standard. Any
 69427 conflict between the requirements described here and the ISO C standard is unintentional. This
 69428 volume of POSIX.1-200x defers to the ISO C standard.

69429 The `wmemset()` function shall copy the value of `wc` into each of the first n wide characters of the
 69430 object pointed to by `ws`. This function shall not be affected by locale and all `wchar_t` values shall
 69431 be treated identically. The null wide character and `wchar_t` values not corresponding to valid
 69432 characters shall not be treated specially.

69433 If n is zero, the application shall ensure that `ws` is a valid pointer, and the function shall copy
 69434 zero wide characters.

69435 **RETURN VALUE**
 69436 The `wmemset()` functions shall return the value of `ws`.

69437 **ERRORS**
 69438 No errors are defined.

69439 **EXAMPLES**
 69440 None.

69441 **APPLICATION USAGE**
 69442 None.

69443 **RATIONALE**
 69444 None.

69445 **FUTURE DIRECTIONS**
 69446 None.

69447 **SEE ALSO**
 69448 [wmemchr\(\)](#), [wmemcmp\(\)](#), [wmemcpy\(\)](#), [wmemmove\(\)](#)

69449 XBD [<wchar.h>](#)

69450 **CHANGE HISTORY**
 69451 First released in Issue 5. Included for alignment with ISO/IEC 9899:1990/Amendment 1:1995
 69452 (E).

69453 **Issue 6**
 69454 The normative text is updated to avoid use of the term “must” for application requirements.

69455 **NAME**
 69456 wordexp, wordfree — perform word expansions

69457 **SYNOPSIS**
 69458 #include <wordexp.h>
 69459 int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
 69460 int flags);
 69461 void wordfree(wordexp_t *pwordexp);

69462 **DESCRIPTION**
 69463 The *wordexp()* function shall perform word expansions as described in XCU Section 2.6 (on page
 69464 2253), subject to quoting as described in XCU Section 2.2 (on page 2246), and place the list of
 69465 expanded words into the structure pointed to by *pwordexp*.

69466 The *words* argument is a pointer to a string containing one or more words to be expanded. The
 69467 expansions shall be the same as would be performed by the command line interpreter if *words*
 69468 were the part of a command line representing the arguments to a utility. Therefore, the
 69469 application shall ensure that *words* does not contain an unquoted <newline> or any of the
 69470 unquoted shell special characters '|', '&', ';', '<', '>' except in the context of command
 69471 substitution as specified in XCU Section 2.6.3 (on page 2256). It also shall not contain unquoted
 69472 parentheses or braces, except in the context of command or variable substitution. The
 69473 application shall ensure that every member of *words* which it expects to have expanded by
 69474 *wordexp()* does not contain an unquoted initial comment character. The application shall also
 69475 ensure that any words which it intends to be ignored (because they begin or continue a
 69476 comment) are deleted from *words*. If the argument *words* contains an unquoted comment
 69477 character (number sign) that is the beginning of a token, *wordexp()* shall either treat the
 69478 comment character as a regular character, or interpret it as a comment indicator and ignore the
 69479 remainder of *words*.

69480 The structure type **wordexp_t** is defined in the <wordexp.h> header and includes at least the
 69481 following members:

Member Type	Member Name	Description
size_t	<i>we_wordc</i>	Count of words matched by <i>words</i> .
char **	<i>we_wordv</i>	Pointer to list of expanded words.
size_t	<i>we_offs</i>	Slots to reserve at the beginning of <i>pwordexp->we_wordv</i> .

69486 The *wordexp()* function shall store the number of generated words into *pwordexp->we_wordc* and
 69487 a pointer to a list of pointers to words in *pwordexp->we_wordv*. Each individual field created
 69488 during field splitting (see XCU Section 2.6.5, on page 2258) or pathname expansion (see XCU
 69489 Section 2.6.6, on page 2259) shall be a separate word in the *pwordexp->we_wordv* list. The words
 69490 shall be in order as described in XCU Section 2.6 (on page 2253). The first pointer after the last
 69491 word pointer shall be a null pointer. The expansion of special parameters described in XCU
 69492 Section 2.5.2 (on page 2250) is unspecified.

69493 It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp()*
 69494 function shall allocate other space as needed, including memory pointed to by
 69495 *pwordexp->we_wordv*. The *wordfree()* function frees any memory associated with *pwordexp* from a
 69496 previous call to *wordexp()*.

69497 The *flags* argument is used to control the behavior of *wordexp()*. The value of *flags* is the bitwise-
 69498 inclusive OR of zero or more of the following constants, which are defined in <wordexp.h>:

69499	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
69500	WRDE_DOOFFS	Make use of <i>pwordexp->we_offs</i> . If this flag is set, <i>pwordexp->we_offs</i> is used to specify how many null pointers to add to the beginning of <i>pwordexp->we_wordv</i> . In other words, <i>pwordexp->we_wordv</i> shall point to <i>pwordexp->we_offs</i> null pointers, followed by <i>pwordexp->we_wordc</i> word pointers, followed by a null pointer.
69501		
69502		
69503		
69504		
69505	WRDE_NOCMD	If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-200x, fail if command substitution, as specified in XCU Section 2.6.3 (on page 2256), is requested.
69506		
69507		
69508	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>wordexp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> without WRDE_REUSE.
69509		
69510		
69511		
69512	WRDE_SHOWERR	Do not redirect <i>stderr</i> to /dev/null .
69513	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
69514		The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to <i>wordexp()</i> . The following rules apply to applications when two or more calls to <i>wordexp()</i> are made with the same value of <i>pwordexp</i> and without intervening calls to <i>wordfree()</i> :
69515		
69516		
69517		1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
69518		2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
69519		3. After the second and each subsequent call, <i>pwordexp->we_wordv</i> shall point to a list containing the following:
69520		
69521		a. Zero or more null pointers, as specified by WRDE_DOOFFS and <i>pwordexp->we_offs</i>
69522		
69523		b. Pointers to the words that were in the <i>pwordexp->we_wordv</i> list before the call, in the same order as before
69524		
69525		c. Pointers to the new words generated by the latest call, in the specified order
69526		4. The count returned in <i>pwordexp->we_wordc</i> shall be the total number of words from all of the calls.
69527		
69528		5. The application can change any of the fields after a call to <i>wordexp()</i> , but if it does it shall reset them to the original value before a subsequent call, using the same <i>pwordexp</i> value, to <i>wordfree()</i> or <i>wordexp()</i> with the WRDE_APPEND or WRDE_REUSE flag.
69529		
69530		
69531		If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-200x, and <i>words</i> contains an unquoted character—<newline>, ' ', ' & ', ' ; ', ' < ', ' > ', ' (', ') ', ' { ', ' } '—in an inappropriate context, <i>wordexp()</i> shall fail, and the number of expanded words shall be 0.
69532		
69533		
69534		
69535		Unless WRDE_SHOWERR is set in <i>flags</i> , <i>wordexp()</i> shall redirect <i>stderr</i> to /dev/null for any utilities executed as a result of command substitution while expanding <i>words</i> . If WRDE_SHOWERR is set, <i>wordexp()</i> may write messages to <i>stderr</i> if syntax errors are detected while expanding <i>words</i> .
69536		
69537		
69538		
69539		The application shall ensure that if WRDE_DOOFFS is set, then <i>pwordexp->we_offs</i> has the same value for each <i>wordexp()</i> call and <i>wordfree()</i> call using a given <i>pwordexp</i> .
69540		
69541		The following constants are defined as error return values:

69542	WRDE_BADCHAR	One of the unquoted characters—<newline>, ' ', '&', ';', '<', '>', '(', ')', '{', '}'—appears in <i>words</i> in an inappropriate context.
69543		
69544	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
69545	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
69546	WRDE_NOSPACE	Attempt to allocate memory failed.
69547	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.
69548		

RETURN VALUE

69549
69550 Upon successful completion, *wordexp()* shall return 0. Otherwise, a non-zero value, as described
69551 in <*wordexp.h*>, shall be returned to indicate an error. If *wordexp()* returns the value
69552 WRDE_NOSPACE, then *pwordexp*→*we_wordc* and *pwordexp*→*we_wordv* shall be updated to
69553 reflect any words that were successfully expanded. In other cases, they shall not be modified.

69554 The *wordfree()* function shall not return a value.

ERRORS

69555
69556 No errors are defined.

EXAMPLES

69557
69558 None.

APPLICATION USAGE

69559
69560 The *wordexp()* function is intended to be used by an application that wants to do all of the shell's
69561 expansions on a word or words obtained from a user. For example, if the application prompts
69562 for a filename (or list of filenames) and then uses *wordexp()* to process the input, the user could
69563 respond with anything that would be valid as input to the shell.

69564
69565 The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to
69566 prevent a user from executing shell commands. Disallowing unquoted shell special characters
also prevents unwanted side effects, such as executing a command or writing a file.

RATIONALE

69567
69568 This function was included as an alternative to *glob()*. There had been continuing controversy
69569 over exactly what features should be included in *glob()*. It is hoped that by providing *wordexp()*
69570 (which provides all of the shell word expansions, but which may be slow to execute) and *glob()*
69571 (which is faster, but which only performs pathname expansion, without tilde or parameter
69572 expansion) this will satisfy the majority of applications.

69573
69574 While *wordexp()* could be implemented entirely as a library routine, it is expected that most
implementations run a shell in a subprocess to do the expansion.

69575
69576 Two different approaches have been proposed for how the required information might be
presented to the shell and the results returned. They are presented here as examples.

69577
69578 One proposal is to extend the *echo* utility by adding a *-q* option. This option would cause *echo* to
69579 add a backslash before each backslash and <blank> that occurs within an argument. The
wordexp() function could then invoke the shell as follows:

```
69580 (void) strcpy(buffer, "echo -q");
69581 (void) strcat(buffer, words);
69582 if ((flags & WRDE_SHOWERR) == 0)
69583     (void) strcat(buffer, "2>/dev/null");
69584 f = popen(buffer, "r");
```

69585
69586 The *wordexp()* function would read the resulting output, remove unquoted backslashes, and
69587 break into words at unquoted <blank>s. If the WRDE_NOCMD flag was set, *wordexp()* would
have to scan *words* before starting the subshell to make sure that there would be no command

69588 substitution. In any case, it would have to scan *words* for unquoted special characters.

69589 Another proposal is to add the following options to *sh*:

69590 **-w** *wordlist*

69591 This option provides a wordlist expansion service to applications. The words in *wordlist*
69592 shall be expanded and the following written to standard output:

- 69593 1. The count of the number of words after expansion, in decimal, followed by a null
69594 byte
- 69595 2. The number of bytes needed to represent the expanded words (not including null
69596 separators), in decimal, followed by a null byte
- 69597 3. The expanded words, each terminated by a null byte

69598 If an error is encountered during word expansion, *sh* exits with a non-zero status after
69599 writing the former to report any words successfully expanded

69600 **-P** Run in “protected” mode. If specified with the **-w** option, no command substitution shall
69601 be performed.

69602 With these options, *wordexp()* could be implemented fairly simply by creating a subprocess
69603 using *fork()* and executing *sh* using the line:

69604 `execl(<shell path>, "sh", "-P", "-w", words, (char *)0);`

69605 after directing standard error to **/dev/null**.

69606 It seemed objectionable for a library routine to write messages to standard error, unless explicitly
69607 requested, so *wordexp()* is required to redirect standard error to **/dev/null** to ensure that no
69608 messages are generated, even for commands executed for command substitution. The
69609 **WRDE_SHOWERR** flag can be specified to request that error messages be written.

69610 The **WRDE_REUSE** flag allows the implementation to avoid the expense of freeing and
69611 reallocating memory, if that is possible. A minimal implementation can call *wordfree()* when
69612 **WRDE_REUSE** is set.

69613 FUTURE DIRECTIONS

69614 None.

69615 SEE ALSO

69616 [fnmatch\(\)](#), [glob\(\)](#)

69617 XBD [<wordexp.h>](#)

69618 XCU [Chapter 2](#) (on page 2245)

69619 CHANGE HISTORY

69620 First released in Issue 4. Derived from the ISO POSIX-2 standard.

69621 Issue 5

69622 Moved from POSIX2 C-language Binding to BASE.

69623 Issue 6

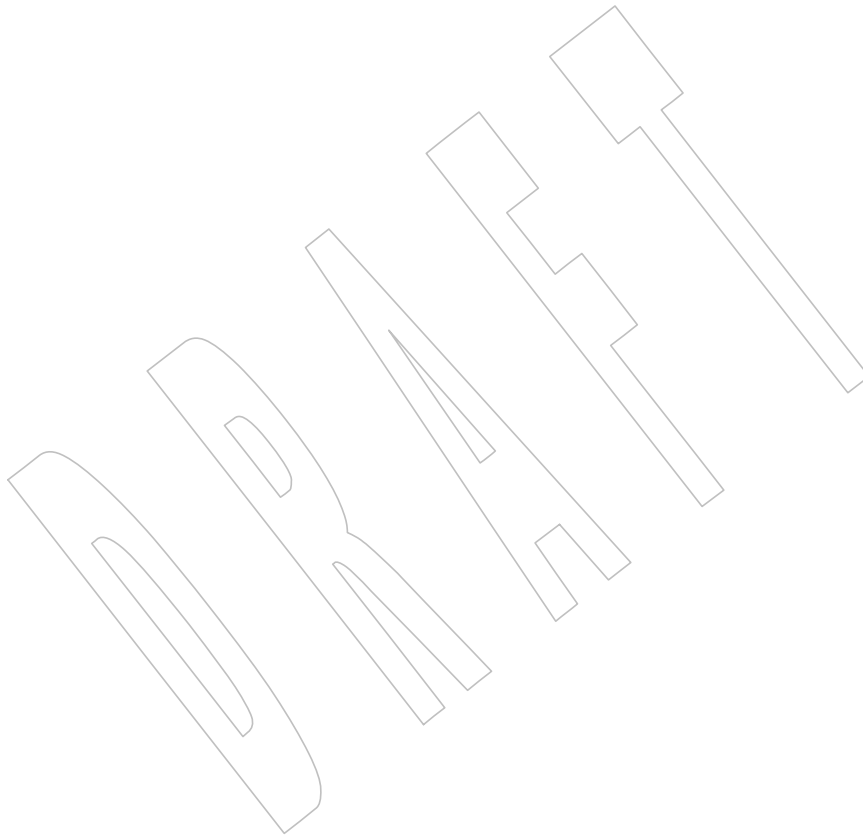
69624 The normative text is updated to avoid use of the term “must” for application requirements.

69625 The **restrict** keyword is added to the *wordexp()* prototype for alignment with the
69626 ISO/IEC 9899:1999 standard.

69627 **NAME**
69628 `wprintf` — print formatted wide-character output

69629 **SYNOPSIS**
69630 `#include <stdio.h>`
69631 `#include <wchar.h>`
69632 `int wprintf(const wchar_t *restrict format, ...);`

69633 **DESCRIPTION**
69634 Refer to *fwprintf()*.



69635 **NAME**

69636 pwrite, write — write on a file

69637 **SYNOPSIS**

69638 #include <unistd.h>

69639 ssize_t pwrite(int *fildes*, const void **buf*, size_t *nbyte*,
69640 off_t *offset*);69641 ssize_t write(int *fildes*, const void **buf*, size_t *nbyte*);69642 **DESCRIPTION**69643 The *write()* function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the
69644 file associated with the open file descriptor, *fildes*.69645 Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the
69646 *write()* function may detect and return errors as described below. In the absence of errors, or if
69647 error detection is not performed, the *write()* function shall return zero and have no other results.
69648 If *nbyte* is zero and the file is not a regular file, the results are unspecified.69649 On a regular file or other file capable of seeking, the actual writing of data shall proceed from
69650 the position in the file indicated by the file offset associated with *fildes*. Before successful return
69651 from *write()*, the file offset shall be incremented by the number of bytes actually written. On a
69652 regular file, if the position of the last byte written is greater than or equal to the length of the file,
69653 the length of the file shall be set to this position plus one.69654 On a file not capable of seeking, writing shall always take place starting at the current position.
69655 The value of a file offset associated with such a device is undefined.69656 If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file
69657 prior to each write and no intervening file modification operation shall occur between changing
69658 the file offset and the write operation.69659 XSI If a *write()* requests that more bytes be written than there is room for (for example, the file size
69660 limit of the process or the physical end of a medium), only as many bytes as there is room for
69661 shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a
69662 limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would
69663 give a failure return (except as noted below).69664 XSI If the request would cause the file size to exceed the soft file size limit for the process and there
69665 is no room for any bytes to be written, the request shall fail and the implementation shall
69666 generate the SIGXFSZ signal for the thread.69667 If *write()* is interrupted by a signal before it writes any data, it shall return -1 with *errno* set to
69668 [EINTR].69669 If *write()* is interrupted by a signal after it successfully writes some data, it shall return the
69670 number of bytes written.69671 If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.69672 After a *write()* to a regular file has successfully returned:69673 • Any successful *read()* from each byte position in the file that was modified by that write
69674 shall return the data specified by the *write()* for that position until such byte positions are
69675 again modified.69676 • Any subsequent successful *write()* to the same byte position in the file shall overwrite that
69677 file data.

69678 Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the

69679

following exceptions:

69680

- There is no file offset associated with a pipe, hence each write request shall append to the end of the pipe.

69681

69682

- Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag of the file status flags is set.

69683

69684

69685

69686

- If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it shall return *nbyte*.

69687

69688

- If the O_NONBLOCK flag is set, *write()* requests shall be handled differently, in the following ways:

69689

69690

— The *write()* function shall not block the thread.

69691

— A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is sufficient space available in the pipe, *write()* shall transfer all the data and return the number of bytes requested. Otherwise, *write()* shall transfer no data and return -1 with *errno* set to [EAGAIN].

69692

69693

69694

69695

— A write request for more than {PIPE_BUF} bytes shall cause one of the following:

69696

— When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe is read, it shall transfer at least {PIPE_BUF} bytes.

69697

69698

69699

— When no data can be written, transfer no data, and return -1 with *errno* set to [EAGAIN].

69700

69701

When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

69702

69703

- If the O_NONBLOCK flag is clear, *write()* shall block the calling thread until the data can be accepted.

69704

69705

- If the O_NONBLOCK flag is set, *write()* shall not block the thread. If some data can be written without blocking the thread, *write()* shall write what it can and return the number of bytes written. Otherwise, it shall return -1 and set *errno* to [EAGAIN].

69706

69707

69708

Upon successful completion, where *nbyte* is greater than 0, *write()* shall mark for update the last data modification and last file status change timestamps of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

69709

69710

69711

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *filides*.

69712

69713

If *filides* refers to a socket, *write()* shall be equivalent to *send()* with no flags set.

69714 SIO

If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

69715

69716

If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

69717

69718 SHM

If *filides* refers to a shared memory object, the result of the *write()* function is unspecified.

69719 TYM

If *filides* refers to a typed memory object, the result of the *write()* function is unspecified.

69720 OB_XSR

If *filides* refers to a STREAM, the operation of *write()* shall be determined by the values of the minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte*

69721

69722

69723 bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value
 69724 is 0, *write()* shall break the buffer into maximum packet size segments prior to sending the data
 69725 downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not
 69726 fall within the range and the minimum value is non-zero, *write()* shall fail with *errno* set to
 69727 [ERANGE]. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0
 69728 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no
 69729 message and 0 is returned. The process may issue `I_SWROPT ioctl()` to enable zero-length
 69730 messages to be sent across the pipe or FIFO.

69731 When writing to a STREAM, data messages are created with a priority band of 0. When writing
 69732 to a STREAM that is not a pipe or FIFO:

- 69733 • If `O_NONBLOCK` is clear, and the STREAM cannot accept data (the STREAM write queue
 69734 is full due to internal flow control conditions), *write()* shall block until data can be
 69735 accepted.
- 69736 • If `O_NONBLOCK` is set and the STREAM cannot accept data, *write()* shall return `-1` and
 69737 set *errno* to [EAGAIN].
- 69738 • If `O_NONBLOCK` is set and part of the buffer has been written while a condition in which
 69739 the STREAM cannot accept additional data occurs, *write()* shall terminate and return the
 69740 number of bytes written.

69741 In addition, *write()* shall fail if the STREAM head has processed an asynchronous error before
 69742 the call. In this case, the value of *errno* does not reflect the result of *write()*, but reflects the prior
 69743 error.

69744 The *pwrite()* function shall be equivalent to *write()*, except that it writes into a given position
 69745 and does not change the file offset (regardless of whether `O_APPEND` is set). The first three
 69746 arguments to *pwrite()* are the same as *write()* with the addition of a fourth argument *offset* for
 69747 the desired position inside the file.

69748 RETURN VALUE

69749 Upon successful completion, these functions shall return the number of bytes actually written to
 69750 the file associated with *fildev*. This number shall never be greater than *nbyte*. Otherwise, `-1` shall
 69751 be returned and *errno* set to indicate the error.

69752 ERRORS

69753 These functions shall fail if:

69754 69755	[EAGAIN]	The <code>O_NONBLOCK</code> flag is set for the file descriptor and the thread would be delayed in the <i>write()</i> operation.
69756	[EBADF]	The <i>fildev</i> argument is not a valid file descriptor open for writing.
69757 69758 69759	[EFBIG]	An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written.
69760 69761 69762	[EFBIG]	The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fildev</i> .
69763 69764	[EINTR]	The write operation was terminated due to the receipt of a signal, and no data was transferred.
69765 69766 69767 69768	[EIO]	The process is a member of a background process group attempting to write to its controlling terminal, <code>TOSTOP</code> is set, the process is neither ignoring nor blocking <code>SIGTTOU</code> , and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

69769		[ENOSPC]	There was no free space remaining on the device containing the file.
69770		[EPIPE]	An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.
69771			
69772			
69773	OB XSR	[ERANGE]	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .
69774			
69775			The <i>write()</i> function shall fail if:
69776		[EAGAIN] or [EWOULDBLOCK]	The file descriptor is for a socket, is marked O_NONBLOCK, and write would block.
69777			
69778			
69779		[ECONNRESET]	A write was attempted on a socket that is not connected.
69780		[EPIPE]	A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.
69781			
69782			
69783			These functions may fail if:
69784	OB XSR	[EINVAL]	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
69785			
69786		[EIO]	A physical I/O error has occurred.
69787		[ENOBUFS]	Insufficient resources were available in the system to perform the operation.
69788		[ENXIO]	A request was made of a nonexistent device, or the request was outside the capabilities of the device.
69789			
69790	OB XSR	[ENXIO]	A hangup occurred on the STREAM being written to.
69791	OB XSR		A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, <i>errno</i> is set to the value included in the error message.
69792			
69793			The <i>write()</i> function may fail if:
69794		[EACCES]	A write was attempted on a socket and the calling process does not have appropriate privileges.
69795			
69796		[ENETDOWN]	A write was attempted on a socket and the local network interface used to reach the destination is down.
69797			
69798		[ENETUNREACH]	A write was attempted on a socket and no route to the network is present.
69799			
69800			The <i>pwrite()</i> function shall fail and the file pointer remain unchanged if:
69801	XSI	[EINVAL]	The <i>offset</i> argument is invalid. The value is negative.
69802	XSI	[ESPIPE]	<i>fildev</i> is associated with a pipe or FIFO.

69803 **EXAMPLES**

69804 **Writing from a Buffer**

69805 The following example writes data from the buffer pointed to by *buf* to the file associated with
69806 the file descriptor *fd*.

```
69807 #include <sys/types.h>
69808 #include <string.h>
69809 ...
69810 char buf[20];
69811 size_t nbytes;
69812 ssize_t bytes_written;
69813 int fd;
69814 ...
69815 strcpy(buf, "This is a test\n");
69816 nbytes = strlen(buf);
69817
69818 bytes_written = write(fd, buf, nbytes);
69819 ...
```

69819 **APPLICATION USAGE**

69820 None.

69821 **RATIONALE**

69822 See also the RATIONALE section in [read](#).

69823 An attempt to write to a pipe or FIFO has several major characteristics:

- 69824 • *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not
69825 interleaved with data from any other process. This is useful when there are multiple
69826 writers sending data to a single reader. Applications need to know how large a write
69827 request can be expected to be performed atomically. This maximum is called {PIPE_BUF}.
69828 This volume of POSIX.1-200x does not say whether write requests for more than
69829 {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall
69830 be atomic.
- 69831 • *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough
69832 space for all the data requested to be written immediately, the implementation should do
69833 so. Otherwise, the calling thread may block; that is, pause until enough space is available
69834 for writing. The effective size of a pipe or FIFO (the maximum amount that can be written
69835 in one operation without blocking) may vary dynamically, depending on the
69836 implementation, so it is not possible to specify a fixed value for it.

- 69837 • *Complete/partial/deferred*: A write request:

```
69838 int fildes;
69839 size_t nbyte;
69840 ssize_t ret;
69841 char *buf;
69842
69843 ret = write(fildes, buf, nbyte);
```

69843 may return:

69844 Complete *ret*=*nbyte*

69845 Partial *ret*<*nbyte*

69846 This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with
69847 *nbyte*>{PIPE_BUF}), this volume of POSIX.1-200x does not guarantee
69848 atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according
69849 to the amount *requested*, not the amount *written*.

69850
69851
69852
69853
69854
69855
69856
69857
69858
69859
69860
69861
69862
69863
69864
69865
69866
69867
69868
69869
69870
69871
69872
69873
69874
69875
69876
69877
69878
69879
69880
69881
69882
69883
69884
69885
69886
69887
69888
69889
69890
69891
69892
69893
69894
69895

Deferred: *ret* = -1, *errno* = [EAGAIN]

This error indicates that a later request may succeed. It does not indicate that it *shall* succeed, even if *nbyte* ≤ {PIPE_BUF}, because if no process reads from the pipe or FIFO, the write never succeeds. An application could usefully count the number of times [EAGAIN] is caused by a particular value of *nbyte* > {PIPE_BUF} and perhaps do later writes with a smaller value, on the assumption that the effective size of the pipe may have decreased.

Partial and deferred writes are only possible with O_NONBLOCK set.

The relations of these properties are shown in the following tables:

Write to a Pipe or FIFO with O_NONBLOCK clear			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤ {PIPE_BUF}	Atomic blocking <i>nbyte</i>	Atomic blocking <i>nbyte</i>	Atomic immediate <i>nbyte</i>
<i>nbyte</i> > {PIPE_BUF}	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>	Blocking <i>nbyte</i>

If the O_NONBLOCK flag is clear, a write request shall block if the amount writable immediately is less than that requested. If the flag is set (by *fcntl()*), a write request shall never block.

Write to a Pipe or FIFO with O_NONBLOCK set			
Immediately Writable:	None	Some	<i>nbyte</i>
<i>nbyte</i> ≤ {PIPE_BUF}	-1, [EAGAIN]	-1, [EAGAIN]	Atomic <i>nbyte</i>
<i>nbyte</i> > {PIPE_BUF}	-1, [EAGAIN]	< <i>nbyte</i> or -1, [EAGAIN]	≤ <i>nbyte</i> or -1, [EAGAIN]

There is no exception regarding partial writes when O_NONBLOCK is set. With the exception of writing to an empty pipe, this volume of POSIX.1-200x does not specify exactly when a partial write is performed since that would require specifying internal details of the implementation. Every application should be prepared to handle partial writes when O_NONBLOCK is set and the requested amount is greater than {PIPE_BUF}, just as every application should be prepared to handle partial writes on other kinds of file descriptors.

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be written; if not, at least some progress must have been made.

Where this volume of POSIX.1-200x requires -1 to be returned and *errno* set to [EAGAIN], most historical implementations return zero (with the O_NDELAY flag set, which is the historical predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1-200x). The error indications in this volume of POSIX.1-200x were chosen so that an application can distinguish these cases from end-of-file. While *write()* cannot receive an indication of end-of-file, *read()* can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from *write()* indicates a successful write of an end-of-file indication.

Implementations are allowed, but not required, to perform error checking for *write()* requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

See also the discussion of O_NONBLOCK in *read*.

Writes can be serialized with respect to other reads and writes. If a *read()* of file data can be proven (by any means) to occur after a *write()* of the data, it must reflect that *write()*, even if the calls are made by different processes. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write()* calls to subsequent *read()* calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

Note that this is specified in terms of *read()* and *write()*. The XSI extensions *readv()* and *writew()* also obey these semantics. A new “high-performance” write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of POSIX.1-200x is also silent about any effects of application-level caching (such as that done by *stdio*).

This volume of POSIX.1-200x does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as [EBADF], the concept is meaningless since no file is involved. For errors that are detected immediately, such as [EAGAIN], clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of POSIX.1-200x does not specify behavior of concurrent writes to a file from multiple processes. Applications should use some form of concurrency control.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *creat()*, *dup()*, *fcntl()*, *getrlimit()*, *lseek()*, *open()*, *pipe()*, *ulimit*, *writew()*

XBD [<limits.h>](#), [<stropts.h>](#), [<sys/uio.h>](#), [<unistd.h>](#)

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Realtime Extension and the POSIX Threads Extension.

Large File Summit extensions are added.

The *pwrite()* function is added.

Issue 6

The DESCRIPTION states that the *write()* function does not block the thread. Previously this said “process” rather than “thread”.

The DESCRIPTION and ERRORS sections are updated so that references to STREAMS are marked as part of the XSI STREAMS Option Group.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The DESCRIPTION now states that if *write()* is interrupted by a signal after it has successfully written some data, it returns the number of bytes written. In the POSIX.1-1988 standard, it was optional whether *write()* returned the number of bytes written, or whether it returned -1 with *errno* set to [EINTR]. This is a FIPS requirement.
- The following changes are made to support large files:
 - For regular files, no data transfer occurs past the offset maximum established in the open file description associated with the *files*.

69940
69941
69942
69943
69944
69945
69946
69947
69948
69949
69950
69951
69952
69953
69954
69955
69956
69957
69958
69959
69960
69961
69962
69963
69964
69965

— A second [EFBIG] error condition is added.

- The [EIO] error condition is added.
- The [EPIPE] error condition is added for when a pipe has only one end open.
- The [ENXIO] optional error condition is added.

Text referring to sockets is added to the DESCRIPTION.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The effect of reading zero bytes is clarified.

The DESCRIPTION is updated for alignment with IEEE Std 1003.1j-2000 by specifying that *write()* results are unspecified for typed memory objects.

The following error conditions are added for operations on sockets: [EAGAIN], [EWOULDBLOCK], [ECONNRESET], [ENOTCONN], and [EPIPE].

The [EIO] error is made optional.

The [ENOBUS] error is added for sockets.

The following error conditions are added for operations on sockets: [EACCES], [ENETDOWN], and [ENETUNREACH].

The *writew()* function is split out into a separate reference page.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/146 is applied, updating text in the ERRORS section from “a SIGPIPE signal is generated to the calling process” to “a SIGPIPE signal shall also be sent to the thread”.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/147 is applied, making a correction to the RATIONALE.

Issue 7

The *pwrite()* function is moved from the XSI option to the Base.

Functionality relating to the XSI STREAMS option is marked obsolescent.

SD5-XSH-ERN-160 is applied, updating the DESCRIPTION to clarify the requirements for the *pwrite()* function, and to change the use of the phrase “file pointer” to “file offset”.

69966 **NAME**

69967 writev — write a vector

69968 **SYNOPSIS**

```
69969 XSI #include <sys/uio.h>
69970 ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

69971 **DESCRIPTION**

69972 The *writev()* function shall be equivalent to *write()*, except as described below. The *writev()*
 69973 function shall gather output data from the *iovcnt* buffers specified by the members of the *iov*
 69974 array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*-1]. The *iovcnt* argument is valid if greater than 0 and less than
 69975 or equal to {IOV_MAX}, as defined in <limits.h>.

69976 Each *iovec* entry specifies the base address and length of an area in memory from which data
 69977 should be written. The *writev()* function shall always write a complete area before proceeding to
 69978 the next.

69979 If *fildes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0,
 69980 *writev()* shall return 0 and have no other effect. For other file types, the behavior is unspecified.

69981 If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data
 69982 shall be transferred.

69983 **RETURN VALUE**

69984 Upon successful completion, *writev()* shall return the number of bytes actually written.
 69985 Otherwise, it shall return a value of -1, the file-pointer shall remain unchanged, and *errno* shall
 69986 be set to indicate an error.

69987 **ERRORS**69988 Refer to *write*.69989 In addition, the *writev()* function shall fail if:69990 [EINVAL] The sum of the *iov_len* values in the *iov* array would overflow an *ssize_t*.69991 The *writev()* function may fail and set *errno* to:69992 [EINVAL] The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.69993 **EXAMPLES**69994 **Writing Data from an Array**

69995 The following example writes data from the buffers specified by members of the *iov* array to the
 69996 file associated with the file descriptor *fd*.

```
69997 #include <sys/types.h>
69998 #include <sys/uio.h>
69999 #include <unistd.h>
70000 ...
70001 ssize_t bytes_written;
70002 int fd;
70003 char *buf0 = "short string\n";
70004 char *buf1 = "This is a longer string\n";
70005 char *buf2 = "This is the longest string in this example\n";
70006 int iocnt;
70007 struct iovec iov[3];
```

```
70008     iov[0].iov_base = buf0;
70009     iov[0].iov_len = strlen(buf0);
70010     iov[1].iov_base = buf1;
70011     iov[1].iov_len = strlen(buf1);
70012     iov[2].iov_base = buf2;
70013     iov[2].iov_len = strlen(buf2);
70014     ...
70015     iovcnt = sizeof(iov) / sizeof(struct iovec);
70016     bytes_written = writev(fd, iov, iovcnt);
70017     ...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *write*.

FUTURE DIRECTIONS

None.

SEE ALSO

readv(), *write*

XBD [<limits.h>](#), [<sys/uio.h>](#)

CHANGE HISTORY

First released in Issue 4, Version 2.

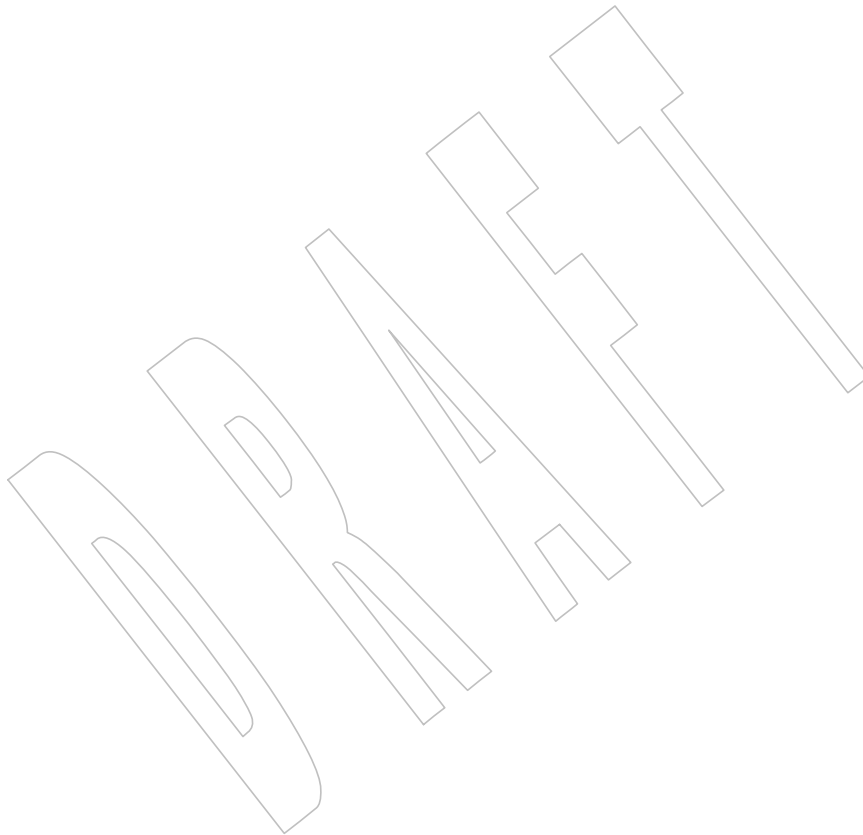
Issue 6

Split out from the *write()* reference page.

70031 **NAME**
70032 `wscanf` — convert formatted wide-character input

70033 **SYNOPSIS**
70034 `#include <stdio.h>`
70035 `#include <wchar.h>`
70036 `int wscanf(const wchar_t *restrict format, ...);`

70037 **DESCRIPTION**
70038 Refer to *fwscanf()*.



70039 **NAME**
 70040 y_0 , y_1 , y_n — Bessel functions of the second kind

70041 **SYNOPSIS**

```
70042 xSI #include <math.h>
70043 double y0(double x);
70044 double y1(double x);
70045 double yn(int n, double x);
```

70046 **DESCRIPTION**

70047 The $y_0()$, $y_1()$, and $y_n()$ functions shall compute Bessel functions of x of the second kind of
 70048 orders 0, 1, and n , respectively.

70049 An application wishing to check for error situations should set *errno* to zero and call
 70050 *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or
 70051 *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-
 70052 zero, an error has occurred.

70053 **RETURN VALUE**

70054 Upon successful completion, these functions shall return the relevant Bessel value of x of the
 70055 second kind.

70056 If x is NaN, NaN shall be returned.

70057 If the x argument to these functions is negative, $-\text{HUGE_VAL}$ or NaN shall be returned, and a
 70058 domain error may occur.

70059 If x is 0.0, $-\text{HUGE_VAL}$ shall be returned and a pole error may occur.

70060 If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

70061 If the correct result would cause overflow, $-\text{HUGE_VAL}$ or 0.0 shall be returned and a range
 70062 error may occur.

70063 **ERRORS**

70064 These functions may fail if:

70065 Domain Error The value of x is negative.

70066 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 70067 then *errno* shall be set to [EDOM]. If the integer expression (*math_errhandling*
 70068 & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception
 70069 shall be raised.

70070 Pole Error The value of x is zero.

70071 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 70072 then *errno* shall be set to [ERANGE]. If the integer expression
 70073 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero
 70074 floating-point exception shall be raised.

70075 Range Error The correct result would cause overflow.

70076 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 70077 then *errno* shall be set to [ERANGE]. If the integer expression
 70078 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow
 70079 floating-point exception shall be raised.

70080 Range Error The value of x is too large in magnitude, or the correct result would cause
 70081 underflow.
 70082 If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero,
 70083 then *errno* shall be set to [ERANGE]. If the integer expression
 70084 (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow
 70085 floating-point exception shall be raised.

EXAMPLES

70086 None.
 70087

APPLICATION USAGE

70088 On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* &
 70089 MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.
 70090

RATIONALE

70091 None.
 70092

FUTURE DIRECTIONS

70093 None.
 70094

SEE ALSO

70095 *feclearexcept()*, *fetestexcept()*, *isnan()*, *j0()*
 70096

70097 XBD Section 4.19 (on page 104), `<math.h>`

CHANGE HISTORY

70098 First released in Issue 1. Derived from Issue 1 of the SVID.
 70099

Issue 5

70100 The DESCRIPTION is updated to indicate how an application should check for an error. This
 70101 text was previously published in the APPLICATION USAGE section.
 70102

Issue 6

70103 The normative text is updated to avoid use of the term “must” for application requirements.
 70104

70105 The RETURN VALUE and ERRORS sections are reworked for alignment of the error handling
 70106 with the ISO/IEC 9899:1999 standard.

70107 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/148 is applied, updating the RETURN
 70108 VALUE and ERRORS sections. The changes are made for consistency with the general rules
 70109 stated in “Treatment of Error Conditions for Mathematical Functions” in the Base Definitions
 70110 volume of POSIX.1-200x.

70111

Technical Standard

70112

Volume 3:

70113

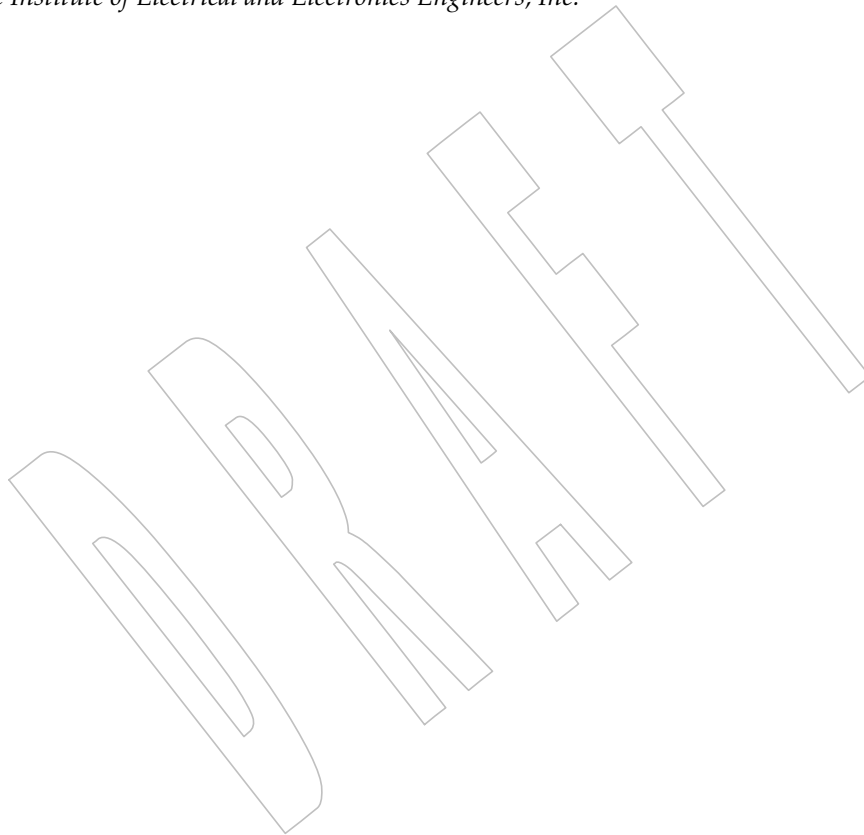
Shell and Utilities, Issue 7

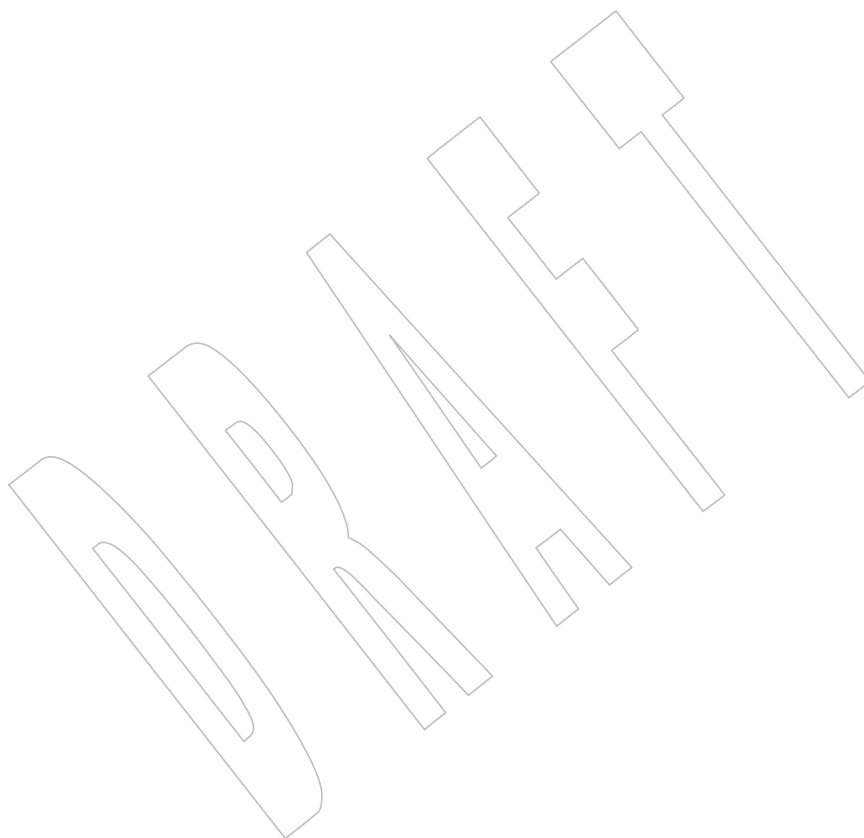
70114

The Open Group

70115

The Institute of Electrical and Electronics Engineers, Inc.





The Shell and Utilities volume of POSIX.1-200x describes the commands and utilities offered to application programs by POSIX-conformant systems.

1.1 Relationship to Other Documents

1.1.1 System Interfaces

This subsection describes some of the features provided by the System Interfaces volume of POSIX.1-200x that are assumed to be globally available on all systems conforming to this volume of POSIX.1-200x. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of POSIX.1-200x that are required by all of the utilities defined in this volume of POSIX.1-200x; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of POSIX.1-200x. Utility and function description statements override these defaults when appropriate.

1.1.1.1 Process Attributes

The following process attributes, as described in the System Interfaces volume of POSIX.1-200x, are assumed to be supported for all processes in this volume of POSIX.1-200x:

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

A conforming implementation may include additional process attributes.

1.1.1.2 Concurrent Execution of Processes

The following functionality of the *fork()* function defined in the System Interfaces volume of POSIX.1-200x shall be available on all systems conforming to this volume of POSIX.1-200x:

1. Independent processes shall be capable of executing independently without either process terminating.
2. A process shall be able to create a new process with all of the attributes referenced in [Section 1.1.1.1](#), determined according to the semantics of a call to the *fork()* function defined in the System Interfaces volume of POSIX.1-200x followed by a call in the child process to one of the *exec* functions defined in the System Interfaces volume of

70153 POSIX.1-200x.

70154 1.1.1.3 *File Access Permissions*

70155 The file access control mechanism described by XBD [Section 4.4](#) (on page 96) shall apply to all
70156 files on an implementation conforming to this volume of POSIX.1-200x.

70157 1.1.1.4 *File Read, Write, and Creation*

70158 If a file that does not exist is to be written, it shall be created as described below, unless the
70159 utility description states otherwise.

70160 When a file that does not exist is created, the following features defined in the System Interfaces
70161 volume of POSIX.1-200x shall apply unless the utility or function description states otherwise:

- 70162 1. The user ID of the file shall be set to the effective user ID of the calling process.
- 70163 2. The group ID of the file shall be set to the effective group ID of the calling process or the
70164 group ID of the directory in which the file is being created.
- 70165 3. If the file is a regular file, the permission bits of the file shall be set to:

70166 S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

70167 (see the description of *File Modes* in XBD [Chapter 13](#) (on page 205), `<sys/stat.h>`) except
70168 that the bits specified by the file mode creation mask of the process shall be cleared. If the
70169 file is a directory, the permission bits shall be set to:

70170 S_IRWXU | S_IRWXG | S_IRWXO

70171 except that the bits specified by the file mode creation mask of the process shall be
70172 cleared.

- 70173 4. The last data access, last data modification, and last file status change timestamps of the
70174 file shall be updated as specified in XBD [Section 4.8](#) (on page 97).
- 70175 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length
70176 zero.
- 70177 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS}
70178 variable is in effect for the directory in which the symbolic link would be created.
- 70179 7. Unless otherwise specified, the file created shall be a regular file.

70180 When an attempt is made to create a file that already exists, the utility shall take the action
70181 indicated in [Table 1-1](#) (on page 2229) corresponding to the type of the file the utility is trying to
70182 create and the type of the existing file, unless the utility description states otherwise.

70183

Table 1-1 Actions when Creating a File that Already Exists

70184

70185

70186

70187

70188

70189

70190

70191

70192

70193

70194

70195

70196

70197

Existing Type	New Type											Function Creating New
	B	C	D	F	L	M	P	Q	R	S	T	
A <i>fattach()</i> -ed STREAM	F	F	F	F	F	—	—	—	OF	—	U	N/A
B Block Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
C Character Special	F	F	F	F	F	U	U	U	OF	U	U	<i>mknod()</i> **
D Directory	F	F	F	F	F	—	—	—	F	—	U	<i>mkdir()</i>
F FIFO Special File	F	F	F	F	F	—	—	—	O	—	U	<i>mkfifo()</i>
L Symbolic Link	F	F	F	F	F	—	—	—	FL	—	U	<i>symlink()</i>
M Shared Memory	F	F	F	F	F	—	—	—	—	—	U	<i>shm_open()</i>
P Semaphore	F	F	F	F	F	—	—	—	—	—	U	<i>sem_open()</i>
Q Message Queue	F	F	F	F	F	—	—	—	—	—	U	<i>mq_open()</i>
R Regular File	F	F	F	F	F	—	—	—	RF	—	U	<i>open()</i>
S Socket	F	F	F	F	F	—	—	—	—	—	U	<i>bind()</i>
T Typed Memory	F	F	F	F	F	U	U	U	U	U	U	*

70198

The following codes are used in [Table 1-1](#):

70199

70200

70201

F Fail. The attempt to create the new file shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

70202

70203

70204

70205

FL Follow link. Unless otherwise specified, the symbolic link shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

70206

70207

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

70208

70209

70210

70211

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.
2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

70212

OF The named file shall be opened with the consequences defined for that file type.

70213

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

70214

70215

70216

70217

1. The user ID, group ID, and permission bits of the file shall not be changed.
2. The file shall be truncated to zero length.
3. The last data modification and last file status change timestamps shall be marked for update.

70218

— The effect is implementation-defined unless specified by the utility description.

70219

U The effect is unspecified unless specified by the utility description.

70220

***** There is no portable way to create a file of this type.

70221

****** Not portable.

70222

70223

70224

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the `open()` function defined in the System Interfaces volume of POSIX.1-200x.

70225 When a file is to be read or written, the file shall be opened with an access mode corresponding
 70226 to the operation to be performed. If file access permissions deny access, the requested operation
 70227 shall fail.

70228 1.1.1.5 File Removal

70229 When a directory that is the root directory or current working directory of any process is
 70230 removed, the effect is implementation-defined. If file access permissions deny access, the
 70231 requested operation shall fail. Otherwise, when a file is removed:

- 70232 1. Its directory entry shall be removed from the file system.
- 70233 2. The link count of the file shall be decremented.
- 70234 3. If the file is an empty directory (see XBD Section 3.143, on page 53):
 - 70235 a. If no process has the directory open, the space occupied by the directory shall be
 70236 freed and the directory shall no longer be accessible.
 - 70237 b. If one or more processes have the directory open, the directory contents shall be
 70238 preserved until all references to the file have been closed.
- 70239 4. If the file is a directory that is not empty, the last file status change timestamp shall be
 70240 marked for update.
- 70241 5. If the file is not a directory:
 - 70242 a. If the link count becomes zero:
 - 70243 i. If no process has the file open, the space occupied by the file shall be freed
 70244 and the file shall no longer be accessible.
 - 70245 ii. If one or more processes have the file open, the file contents shall be
 70246 preserved until all references to the file have been closed.
 - 70247 b. If the link count is not reduced to zero, the last file status change timestamp shall
 70248 be marked for update.
- 70249 6. The last data modification and last file status change timestamps of the containing
 70250 directory shall be marked for update.

70251 1.1.1.6 File Time Values

70252 All files shall have the three time values described by XBD Section 4.8 (on page 97).

70253 1.1.1.7 File Contents

70254 When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of
 70255 the data placed in the space pointed to by *buf* when performing the *read()* function calls in the
 70256 following operations defined in the System Interfaces volume of POSIX.1-200x:

```
70257 while (read (fildes, buf, nbytes) > 0)
70258     ;
```

70259 If the file is indicated by a *pathname*, the file descriptor shall be determined by the
 70260 equivalent of the following operation defined in the System Interfaces volume of POSIX.1-200x:

```
70261 fildes = open (pathname, O_RDONLY);
```

70262 The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data
 70263 returned by *read()* would vary with different values, the value shall be one that results in the
 70264 most data being returned.

70265 If the *read()* function calls would return an error, it is unspecified whether the contents of the file
70266 are considered to include any data from offsets in the file beyond where the error would be
70267 returned.

70268 1.1.1.8 Pathname Resolution

70269 The pathname resolution algorithm, described by XBD Section 4.12 (on page 99), shall be used |
70270 by implementations conforming to this volume of POSIX.1-200x; see also XBD Section 4.5 (on |
70271 page 96).

70272 1.1.1.9 Changing the Current Working Directory

70273 When the current working directory (see XBD Section 3.121, on page 50) is to be changed, unless |
70274 the utility or function description states otherwise, the operation shall succeed unless a call to |
70275 the *chdir()* function defined in the System Interfaces volume of POSIX.1-200x would fail when
70276 invoked with the new working directory pathname as its argument.

70277 1.1.1.10 Establish the Locale

70278 The functionality of the *setlocale()* function defined in the System Interfaces volume of
70279 POSIX.1-200x shall be available on all systems conforming to this volume of POSIX.1-200x; that
70280 is, utilities that require the capability of establishing an international operating environment
70281 shall be permitted to set the specified category of the international environment.

70282 1.1.1.11 Actions Equivalent to Functions

70283 Some utility descriptions specify that a utility performs actions equivalent to a function defined
70284 in the System Interfaces volume of POSIX.1-200x. Such specifications require only that the
70285 external effects be equivalent, not that any effect within the utility and visible only to the utility
70286 be equivalent.

70287 1.1.2 Concepts Derived from the ISO C Standard

70288 Some of the standard utilities perform complex data manipulation using their own procedure
70289 and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS
70290 sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type
70291 conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard,
70292 as described in the following sections. Note that there is no requirement that the standard
70293 utilities be implemented in any particular programming language.

70294 1.1.2.1 Arithmetic Precision and Operations

70295 Integer variables and constants, including the values of operands and option-arguments, used
70296 by the standard utilities listed in this volume of POSIX.1-200x shall be implemented as
70297 equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as
70298 equivalent to the ISO C standard **double** type. Conversions between types shall be as described
70299 in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned
70300 by the input to the application.

70301 Arithmetic operators and control flow keywords shall be implemented as equivalent to those in
70302 the cited ISO C standard section, as listed in Table 1-2 (on page 2232).

70303

Table 1-2 Selected ISO C Standard Operators and Control Flow Keywords

70304

70305

70306

70307

70308

70309

70310

70311

70312

70313

70314

70315

70316

70317

70318

70319

70320

70321

70322

70323

70324

70325

70326

70327

70328

70329

70330

70331

70332

70333

70334

70335

70336

70337

70338

70339

70340

70341

70342

70343

70344

70345

Operation	ISO C Standard Equivalent Reference
()	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! sizeof()	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
== !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
	Section 6.5.14, Logical OR Operator
<i>expr?expr:expr</i>	Section 6.5.15, Conditional Operator
=, *=, /=, %=, +=, -= <<=, >>=, &=, ^=, =	Section 6.5.16, Assignment Operators
if () if () ... else switch ()	Section 6.8.4, Selection Statements
while () do ... while () for ()	Section 6.8.5, Iteration Statements
goto continue break return	Section 6.8.6, Jump Statements

The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5, Expressions, of the ISO C standard.

70346 1.1.2.2 *Mathematical Functions*70347 Any mathematical functions with the same names as those in the following sections of the ISO C
70348 standard:

- 70349
- Section 7.12, Mathematics, <math.h>
 - Section 7.20.2, Pseudo-Random Sequence Generation Functions
- 70350

70351 shall be implemented to return the results equivalent to those returned from a call to the
70352 corresponding function described in the ISO C standard.70353 **1.2 Utility Limits**70354 This section lists magnitude limitations imposed by a specific implementation. The braces
70355 notation, {LIMIT}, is used in this volume of POSIX.1-200x to indicate these values, but the braces
70356 are not part of the name.70357 **Table 1-3** Utility Limit Minimum Values70358
70359
70360
70361
70362
70363
70364
70365
70366
70367
70368
70369
70370
70371
70372
70373
70374
70375
70376
70377
70378
70379
70380
70381
70382

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	2 048
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	1 000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the border_start keyword in XBD Section 7.3.2 (on page 132).	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	2 048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see XBD Section 9.3.6 (on page 172).	255

70383 The values specified in [Table 1-3](#) represent the lowest values conforming implementations shall
70384 provide and, consequently, the largest values on which an application can rely without further
70385 enquiries, as described below. These values shall be accessible to applications via the *getconf*
70386 utility (see *getconf*, on page 2701).70387 Implementations may provide more liberal, or less restrictive, values than shown in [Table 1-3](#).
70388 These possibly more liberal values are accessible using the symbols in [Table 1-4](#) (on page 2234).

70389 The *sysconf()* function defined in the System Interfaces volume of POSIX.1-200x or the *getconf*
 70390 utility return the value of each symbol on each specific implementation. The value so retrieved is
 70391 the largest, or most liberal, value that is available throughout the session lifetime, as determined
 70392 at session creation. The literal names shown in the table apply only to the *getconf* utility; the
 70393 high-level language binding describes the exact form of each name to be used by the interfaces
 70394 in that binding.

70395 All numeric limits defined by the System Interfaces volume of POSIX.1-200x, such as
 70396 {PATH_MAX}, shall also apply to this volume of POSIX.1-200x. All the utilities defined by this
 70397 volume of POSIX.1-200x are implicitly limited by these values, unless otherwise noted in the
 70398 utility descriptions.

70399 It is not guaranteed that the application can actually reach the specified limit of an
 70400 implementation in any given case, or at all, as a lack of virtual memory or other resources may
 70401 prevent this. The limit value indicates only that the implementation does not specifically impose
 70402 any arbitrary, more restrictive limit.

70403 **Table 1-4** Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <i>bc</i> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <i>bc</i> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <i>bc</i> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <i>LC_COLLATE</i> order keyword in the locale definition file; see the order_start keyword in XBD Section 7.3.2 (on page 132).	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <i>expr</i> utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <newline>.	{POSIX2_LINE_MAX}

70436
70437
70438
70439
70440
70441
70442

Name	Description	Minimum Value
{RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation $\{m,n\}$; see XBD Section 9.3.6 (on page 172).	{POSIX2_RE_DUP_MAX}

70443
70444
70445
70446
70447

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2_SYMLINKS}

When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

70448

1.3 Grammar Conventions

70449
70450
70451
70452
70453
70454
70455
70456
70457
70458

Portions of this volume of POSIX.1-200x are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

70459
70460

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

70461
70462
70463
70464
70465

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

70466

1.4 Utility Description Defaults

70467
70468
70469
70470
70471

This section describes all of the subsections used within the utility descriptions, including:

- Intended usage of the section
- Global defaults that affect all the standard utilities
- The meanings of notations used in this volume of POSIX.1-200x that are specific to individual utility sections

70472
70473

NAME

This section gives the name or names of the utility and briefly states its purpose.

70474
70475
70476

SYNOPSIS

The SYNOPSIS section summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are

70477 described in XBD Section 12.2 (on page 201); for describing the utility's arguments in
70478 XBD Section 12.1 (on page 199).

70479 DESCRIPTION

70480 The DESCRIPTION section describes the actions of the utility. If the utility has a very
70481 complex set of subcommands or its own procedural language, an EXTENDED
70482 DESCRIPTION section is also provided. Most explanations of optional functionality are
70483 omitted here, as they are usually explained in the OPTIONS section.

70484 As stated in Section 1.1.1.11 (on page 2231), some functions are described in terms of
70485 equivalent functionality. When specific functions are cited, the implementation shall
70486 provide equivalent functionality including side effects associated with successful
70487 execution of the function. The treatment of errors and intermediate results from the
70488 individual functions cited is generally not specified by this volume of POSIX.1-200x.
70489 See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all
70490 actions associated with errors encountered by the utility.

70491 OPTIONS

70492 The OPTIONS section describes the utility options and option-arguments, and how
70493 they modify the actions of the utility. Standard utilities that have options either fully
70494 comply with XBD Section 12.2 (on page 201) or describe all deviations. Apparent
70495 disagreements between functionality descriptions in the OPTIONS and DESCRIPTION
70496 (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS
70497 section.

70498 Each OPTIONS section that uses the phrase "The ... utility shall conform to the Utility
70499 Syntax Guidelines ..." refers only to the use of the utility as specified by this volume of
70500 POSIX.1-200x; implementation extensions should also conform to the guidelines, but
70501 may allow exceptions for historical practice.

70502 Unless otherwise stated in the utility description, when given an option unrecognized
70503 by the implementation, or when a required option-argument is not provided, standard
70504 utilities shall issue a diagnostic message to standard error and exit with a non-zero exit
70505 status.

70506 All utilities in this volume of POSIX.1-200x shall be capable of processing arguments
70507 using eight-bit transparency.

70508 **Default Behavior:** When this section is listed as "None.", it means that the
70509 implementation need not support any options. Standard utilities that do not accept
70510 options, but that do accept operands, shall recognize "--" as a first argument to be
70511 discarded.

70512 The requirement for recognizing "--" is because conforming applications need a way
70513 to shield their operands from any arbitrary options that the implementation may
70514 provide as an extension. For example, if the standard utility *foo* is listed as taking no
70515 options, and the application needed to give it a pathname with a leading hyphen, it
70516 could safely do it as:

```
70517 foo -- -myfile
```

70518 and avoid any problems with **-m** used as an extension.

70519 OPERANDS

70520 The OPERANDS section describes the utility operands, and how they affect the actions
70521 of the utility. Apparent disagreements between functionality descriptions in the
70522 OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be
70523 resolved in favor of the OPERANDS section.

70524 If an operand naming a file can be specified as ‘-’, which means to use the standard
70525 input instead of a named file, this is explicitly stated in this section. Unless otherwise
70526 stated, the use of multiple instances of ‘-’ to mean standard input in a single
70527 command produces unspecified results.

70528 Unless otherwise stated, the standard utilities that accept operands shall process those
70529 operands in the order specified in the command line.

70530 **Default Behavior:** When this section is listed as “None.”, it means that the
70531 implementation need not support any operands.

70532 STDIN

70533 The STDIN section describes the standard input of the utility. This section is frequently
70534 merely a reference to the following section, as many utilities treat standard input and
70535 input files in the same manner. Unless otherwise stated, all restrictions described in the
70536 INPUT FILES section shall apply to this section as well.

70537 Use of a terminal for standard input can cause any of the standard utilities that read
70538 standard input to stop when used in the background. For this reason, applications
70539 should not use interactive features in scripts to be placed in the background.

70540 The specified standard input format of the standard utilities shall not depend on the
70541 existence or value of the environment variables defined in this volume of
70542 POSIX.1-200x, except as provided by this volume of POSIX.1-200x.

70543 **Default Behavior:** When this section is listed as “Not used.”, it means that the standard
70544 input shall not be read when the utility is used as described by this volume of
70545 POSIX.1-200x.

70546 INPUT FILES

70547 The INPUT FILES section describes the files, other than the standard input, used as
70548 input by the utility. It includes files named as operands and option-arguments as well
70549 as other files that are referred to, such as start-up and initialization files, databases, and
70550 so on. Commonly-used files are generally described in one place and cross-referenced
70551 by other utilities.

70552 All utilities in this volume of POSIX.1-200x shall be capable of processing input files
70553 using eight-bit transparency.

70554 When a standard utility reads a seekable input file and terminates without an error
70555 before it reaches end-of-file, the utility shall ensure that the file offset in the open file
70556 description is properly positioned just past the last byte processed by the utility. For
70557 files that are not seekable, the state of the file offset in the open file description for that
70558 file is unspecified. A conforming application shall not assume that the following three
70559 commands are equivalent:

```
70560 tail -n +2 file
70561 (sed -n 1q; cat) < file
70562 cat file | (sed -n 1q; cat)
```

70563 The second command is equivalent to the first only when the file is seekable. The third
70564 command leaves the file offset in the open file description in an unspecified state. Other
70565 utilities, such as *head*, *read*, and *sh*, have similar properties.

70566 Some of the standard utilities, such as filters, process input files a line or a block at a
70567 time and have no restrictions on the maximum input file size. Some utilities may have
70568 size limitations that are not as obvious as file space or memory limitations. Such
70569 limitations should reflect resource limitations of some sort, not arbitrary limits set by
70570 implementors. Implementations shall document those utilities that are limited by

70571 constraints other than file system space, available memory, and other limits specifically
 70572 cited by this volume of POSIX.1-200x, and identify what the constraint is and indicate a
 70573 way of estimating when the constraint would be reached. Similarly, some utilities
 70574 descend the directory tree (recursively). Implementations shall also document any
 70575 limits that they may have in descending the directory tree that are beyond limits cited
 70576 by this volume of POSIX.1-200x.

70577 When an input file is described as a “text file”, the utility produces undefined results if
 70578 given input that is not from a text file, unless otherwise stated. Some utilities (for
 70579 example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline>
 70580 convention; unless otherwise stated, the utility need not be able to accumulate more
 70581 than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a
 70582 conforming application the total of all the continued lines in a set cannot exceed
 70583 {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-
 70584 file condition immediately after an escaped <newline>, the results are unspecified.

70585 Record formats are described in a notation similar to that used by the C-language
 70586 function, *printf()*. See XBD Chapter 5 (on page 107) for a description of this notation.
 70587 The format description is intended to be sufficiently rigorous to allow other
 70588 applications to generate these input files. However, since <blank>s can legitimately be
 70589 included in some of the fields described by the standard utilities, particularly in locales
 70590 other than the POSIX locale, this intent is not always realized.

70591 **Default Behavior:** When this section is listed as “None.”, it means that no input files
 70592 are required to be supplied when the utility is used as described by this volume of
 70593 POSIX.1-200x.

70594 ENVIRONMENT VARIABLES

70595 The ENVIRONMENT VARIABLES section lists what variables affect the utility’s
 70596 execution.

70597 The entire manner in which environment variables described in this volume of
 70598 POSIX.1-200x affect the behavior of each utility is described in the ENVIRONMENT
 70599 VARIABLES section for that utility, in conjunction with the global effects of the *LANG*,
 70600 XSI *LC_ALL*, and *NLSPATH* environment variables described in XBD Chapter 8 (on page
 70601 159). The existence or value of environment variables described in this volume of
 70602 POSIX.1-200x shall not otherwise affect the specified behavior of the standard utilities.
 70603 Any effects of the existence or value of environment variables not described by this
 70604 volume of POSIX.1-200x upon the standard utilities are unspecified.

70605 For those standard utilities that use environment variables as a means for selecting a
 70606 utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to
 70607 the path search described for *PATH* in XBD Chapter 8 (on page 159).

70608 All utilities in this volume of POSIX.1-200x shall be capable of processing environment
 70609 variable names and values using eight-bit transparency.

70610 **Default Behavior:** When this section is listed as “None.”, it means that the behavior of
 70611 the utility is not directly affected by environment variables described by this volume of
 70612 POSIX.1-200x when the utility is used as described by this volume of POSIX.1-200x.

70613 ASYNCHRONOUS EVENTS

70614 The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as
 70615 signals and what signals are caught.

70616 **Default Behavior:** When this section is listed as “Default.”, or it refers to “the standard
 70617 action for all other signals; see Section 1.4 (on page 2235)” it means that the action taken
 70618 as a result of the signal shall be one of the following:

- 70619 1. The action shall be that inherited from the parent according to the rules of
- 70620 inheritance of signal actions defined in the System Interfaces volume of
- 70621 POSIX.1-200x.
- 70622 2. When no action has been taken to change the default, the default action shall be
- 70623 that specified by the System Interfaces volume of POSIX.1-200x.
- 70624 3. The result of the utility's execution is as if default actions had been taken.

70625 A utility is permitted to catch a signal, perform some additional processing (such as

70626 deleting temporary files), restore the default signal action (or action inherited from the

70627 parent process), and resignal itself.

70628 STDOUT

70629 The STDOUT section completely describes the standard output of the utility. This

70630 section is frequently merely a reference to the following section, OUTPUT FILES,

70631 because many utilities treat standard output and output files in the same manner.

70632 Use of a terminal for standard output may cause any of the standard utilities that write

70633 standard output to stop when used in the background. For this reason, applications

70634 should not use interactive features in scripts to be placed in the background.

70635 Record formats are described in a notation similar to that used by the C-language

70636 function, *printf()*. See XBD [Chapter 5](#) (on page 107) for a description of this notation.

70637 The specified standard output of the standard utilities shall not depend on the

70638 existence or value of the environment variables defined in this volume of

70639 POSIX.1-200x, except as provided by this volume of POSIX.1-200x.

70640 Some of the standard utilities describe their output using the verb *display*, defined in

70641 XBD [Section 3.132](#) (on page 51). Output described in the STDOUT sections of such

70642 utilities may be produced using means other than standard output. When standard

70643 output is directed to a terminal, the output described shall be written directly to the

70644 terminal. Otherwise, the results are undefined.

70645 **Default Behavior:** When this section is listed as "Not used.", it means that the standard

70646 output shall not be written when the utility is used as described by this volume of

70647 POSIX.1-200x.

70648 STDERR

70649 The STDERR section describes the standard error output of the utility. Only those

70650 messages that are purposely sent by the utility are described.

70651 Use of a terminal for standard error may cause any of the standard utilities that write

70652 standard error output to stop when used in the background. For this reason,

70653 applications should not use interactive features in scripts to be placed in the

70654 background.

70655 The format of diagnostic messages for most utilities is unspecified, but the language

70656 and cultural conventions of diagnostic and informative messages whose format is

70657 unspecified by this volume of POSIX.1-200x should be affected by the setting of

70658 XSI `LC_MESSAGES` and `NLSPATH`.

70659 The specified standard error output of standard utilities shall not depend on the

70660 existence or value of the environment variables defined in this volume of

70661 POSIX.1-200x, except as provided by this volume of POSIX.1-200x.

70662 **Default Behavior:** When this section is listed as "The standard error shall be used only

70663 for diagnostic messages.", it means that, unless otherwise stated, the diagnostic

70664 messages shall be sent to the standard error only when the exit status indicates that an

70665 error occurred and the utility is used as described by this volume of POSIX.1-200x.

70666 When this section is listed as “Not used.”, it means that the standard error shall not be
70667 used when the utility is used as described in this volume of POSIX.1-200x.

70668 OUTPUT FILES

70669 The OUTPUT FILES section completely describes the files created or modified by the
70670 utility. Temporary or system files that are created for internal usage by this utility or
70671 other parts of the implementation (for example, spool, log, and audit files) are not
70672 described in this, or any, section. The utilities creating such files and the names of such
70673 files are unspecified. If applications are written to use temporary or intermediate files,
70674 they should use the *TMPDIR* environment variable, if it is set and represents an
70675 accessible directory, to select the location of temporary files.

70676 Implementations shall ensure that temporary files, when used by the standard utilities,
70677 are named so that different utilities or multiple instances of the same utility can operate
70678 simultaneously without regard to their working directories, or any other process
70679 characteristic other than process ID. There are two exceptions to this rule:

- 70680 1. Resources for temporary files other than the name space (for example, disk
70681 space, available directory entries, or number of processes allowed) are not
70682 guaranteed.
- 70683 2. Certain standard utilities generate output files that are intended as input for
70684 other utilities (for example, *lex* generates *lex.yy.c*), and these cannot have unique
70685 names. These cases are explicitly identified in the descriptions of the respective
70686 utilities.

70687 Any temporary file created by the implementation shall be removed by the
70688 implementation upon a utility’s successful exit, exit because of errors, or before
70689 termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified
70690 otherwise by the utility description.

70691 Receipt of the SIGQUIT signal should generally cause termination (unless in some
70692 debugging mode) that would bypass any attempted recovery actions.

70693 Record formats are described in a notation similar to that used by the C-language
70694 function, *printf()*; see XBD [Chapter 5](#) (on page 107) for a description of this notation.

70695 **Default Behavior:** When this section is listed as “None.”, it means that no files are
70696 created or modified as a consequence of direct action on the part of the utility when the
70697 utility is used as described by this volume of POSIX.1-200x. However, the utility may
70698 create or modify system files, such as log files, that are outside the utility’s normal
70699 execution environment.

70700 EXTENDED DESCRIPTION

70701 The EXTENDED DESCRIPTION section provides a place for describing the actions of
70702 very complicated utilities, such as text editors or language processors, which typically
70703 have elaborate command languages.

70704 **Default Behavior:** When this section is listed as “None.”, no further description is
70705 necessary.

70706 EXIT STATUS

70707 The EXIT STATUS section describes the values the utility shall return to the calling
70708 program, or shell, and the conditions that cause these values to be returned. Usually,
70709 utilities return zero for successful completion and values greater than zero for various
70710 error conditions. If specific numeric values are listed in this section, the system shall
70711 use those values for the errors described. In some cases, status values are listed more

70712 loosely, such as >0 . A strictly conforming application shall not rely on any specific
70713 value in the range shown and shall be prepared to receive any value in the range.

70714 For example, a utility may list zero as a successful return, 1 as a failure for a specific
70715 reason, and >1 as “an error occurred”. In this case, unspecified conditions may cause a
70716 2 or 3, or other value, to be returned. A conforming application should be written so
70717 that it tests for successful exit status values (zero in this case), rather than relying upon
70718 the single specific error value listed in this volume of POSIX.1-200x. In that way, it has
70719 maximum portability, even on implementations with extensions.

70720 Unspecified error conditions may be represented by specific values not listed in this
70721 volume of POSIX.1-200x.

70722 CONSEQUENCES OF ERRORS

70723 The CONSEQUENCES OF ERRORS section describes the effects on the environment,
70724 file systems, process state, and so on, when error conditions occur. It does not describe
70725 error messages produced or exit status values used.

70726 The many reasons for failure of a utility are generally not specified by the utility
70727 descriptions. Utilities may terminate prematurely if they encounter: invalid usage of
70728 options, arguments, or environment variables; invalid usage of the complex syntaxes
70729 expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating,
70730 reading, or writing files; or difficulties associated with the privileges of the process.

70731 The following shall apply to each utility, unless otherwise stated:

70732 • If the requested action cannot be performed on an operand representing a file,
70733 directory, user, process, and so on, the utility shall issue a diagnostic message to
70734 standard error and continue processing the next operand in sequence, but the
70735 final exit status shall be returned as non-zero.

70736 For a utility that recursively traverses a file hierarchy (such as *find* or *chown -R*), if
70737 the requested action cannot be performed on a file or directory encountered in the
70738 hierarchy, the utility shall issue a diagnostic message to standard error and
70739 continue processing the remaining files in the hierarchy, but the final exit status
70740 shall be returned as non-zero.

70741 • If the requested action characterized by an option or option-argument cannot be
70742 performed, the utility shall issue a diagnostic message to standard error and the
70743 exit status returned shall be non-zero.

70744 • When an unrecoverable error condition is encountered, the utility shall exit with a
70745 non-zero exit status.

70746 • A diagnostic message shall be written to standard error whenever an error
70747 condition occurs.

70748 When a utility encounters an error condition several actions are possible, depending on
70749 the severity of the error and the state of the utility. Included in the possible actions of
70750 various utilities are: deletion of temporary or intermediate work files; deletion of
70751 incomplete files; validity checking of the file system or directory.

70752 **Default Behavior:** When this section is listed as “Default.”, it means that any changes
70753 to the environment are unspecified.

70754 APPLICATION USAGE

70755 This section is informative.

70756 The APPLICATION USAGE section gives advice to the application programmer or
70757 user about the way the utility should be used.

70758
70759
70760
70761
70762
70763
70764
70765
70766
70767
70768
70769
70770
70771
70772
70773
70774
70775
70776
70777
70778
70779
70780
70781
70782
70783
70784
70785
70786
70787
70788
70789

EXAMPLES

This section is informative.

The EXAMPLES section gives one or more examples of usage, where appropriate. In the event of conflict between an example and a normative part of the specification, the normative material is to be taken as correct.

In all examples, quoting has been used, showing how sample commands (utility names combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to the *system()* function defined in the System Interfaces volume of POSIX.1-200x. Such quoting would not be used if the utility is invoked using one of the *exec* functions defined in the System Interfaces volume of POSIX.1-200x.

RATIONALE

This section is informative.

This section contains historical information concerning the contents of this volume of POSIX.1-200x and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is informative.

The FUTURE DIRECTIONS section should be used as a guide to current thinking; there is not necessarily a commitment to implement all of these future directions in their entirety.

SEE ALSO

This section is informative.

The SEE ALSO section lists related entries.

CHANGE HISTORY

This section is informative.

This section shows the derivation of the entry and any significant changes that have been made to it.

Certain of the standard utilities describe how they can invoke other utilities or applications, such as by passing a command string to the command interpreter. The external influences (STDIN, ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF ERRORS, and so on) of such invoked utilities are not described in the section concerning the standard utility that invokes them.

1.5 Considerations for Utilities in Support of Files of Arbitrary Size

The following utilities support files of any size up to the maximum that can be created by the implementation. This support includes correct writing of file size-related values (such as file sizes and offsets, line numbers, and block counts) and correct interpretation of command line arguments that contain such values.

70795	<i>basename</i>	Return non-directory portion of pathname.
70796	<i>cat</i>	Concatenate and print files.
70797	<i>cd</i>	Change working directory.
70798	<i>chgrp</i>	Change file group ownership.
70799	<i>chmod</i>	Change file modes.
70800	<i>chown</i>	Change file ownership.
70801	<i>cksum</i>	Write file checksums and sizes.
70802	<i>cmp</i>	Compare two files.
70803	<i>cp</i>	Copy files.
70804	<i>dd</i>	Convert and copy a file.
70805	<i>df</i>	Report free disk space.
70806	<i>dirname</i>	Return directory portion of pathname.
70807	<i>du</i>	Estimate file space usage.
70808	<i>find</i>	Find files.
70809	<i>ln</i>	Link files.
70810	<i>ls</i>	List directory contents.
70811	<i>mkdir</i>	Make directories.
70812	<i>mv</i>	Move files.
70813	<i>pathchk</i>	Check pathnames.
70814	<i>pwd</i>	Return working directory name.
70815	<i>rm</i>	Remove directory entries.
70816	<i>rmdir</i>	Remove directories.
70817	<i>sh</i>	Shell, the standard command language interpreter.
70818	<i>sum</i>	Print checksum and block or byte count of a file.
70819	<i>test</i>	Evaluate expression.
70820	<i>touch</i>	Change file access and modification times.
70821	<i>ulimit</i>	Set or report file size limit.

Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:

1. Uses of files as command scripts, or for configuration or control, are exempt. For example, it is not required that *sh* be able to read an arbitrarily large **.profile**.

- 70826 2. Shell input and output redirection are exempt. For example, it is not required that the
70827 redirections *sum < file* or *echo foo > file* succeed for an arbitrarily large existing file.

70828 1.6 Built-In Utilities

70829 Any of the standard utilities may be implemented as regular built-in utilities within the
70830 command language interpreter. This is usually done to increase the performance of frequently
70831 used utilities or to achieve functionality that would be more difficult in a separate environment.
70832 The utilities named in Table 1-5 are frequently provided in built-in form. All of the utilities
70833 named in the table have special properties in terms of command search order within the shell, as
70834 described in Section 2.9.1.1 (on page 2264).

70835 **Table 1-5** Regular Built-In Utilities

70836	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
70837	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
70838	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
70839	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

70840 However, all of the standard utilities, including the regular built-ins in the table, but not the
70841 special built-ins described in Section 2.14 (on page 2280), shall be implemented in a manner so
70842 that they can be accessed via the *exec* family of functions as defined in the System Interfaces
70843 volume of POSIX.1-200x and can be invoked directly by those standard utilities that require it
70844 (*env, find, nice, nohup, time, xargs*).

70847 This chapter contains the definition of the Shell Command Language.

70848 2.1 Shell Introduction

70849 The shell is a command language interpreter. This chapter describes the syntax of that command
70850 language as it is used by the *sh* utility and the *system()* and *popen()* functions defined in the
70851 System Interfaces volume of POSIX.1-200x.

70852 The shell operates according to the following general overview of operations. The specific
70853 details are included in the cited sections of this chapter.

- 70854 1. The shell reads its input from a file (see *sh*), from the `-c` option or from the *system()* and
70855 *popen()* functions defined in the System Interfaces volume of POSIX.1-200x. If the first
70856 line of a file of shell commands starts with the characters "#!", the results are
70857 unspecified.
- 70858 2. The shell breaks the input into tokens: words and operators; see [Section 2.3](#) (on page
70859 2247).
- 70860 3. The shell parses the input into simple commands (see [Section 2.9.1](#), on page 2263) and
70861 compound commands (see [Section 2.9.4](#), on page 2268).
- 70862 4. The shell performs various expansions (separately) on different parts of each command,
70863 resulting in a list of pathnames and fields to be treated as a command and arguments; see
70864 [Section 2.6](#) (on page 2253).
- 70865 5. The shell performs redirection (see [Section 2.7](#), on page 2259) and removes redirection
70866 operators and their operands from the parameter list.
- 70867 6. The shell executes a function (see [Section 2.9.5](#), on page 2270), built-in (see [Section 2.14](#),
70868 on page 2280), executable file, or script, giving the names of the arguments as positional
70869 parameters numbered 1 to *n*, and the name of the command (or in the case of a function
70870 within a script, the name of the script) as the positional parameter numbered 0 (see
70871 [Section 2.9.1.1](#), on page 2264).
- 70872 7. The shell optionally waits for the command to complete and collects the exit status (see
70873 [Section 2.8.2](#), on page 2262).

2.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph, prevent reserved words from being recognized as such, and prevent parameter expansion and command substitution within here-document processing (see [Section 2.7.4](#), on page 2260).

The application shall quote the following characters if they are to represent themselves:

```
| & ; < > ( ) $ ` \ " ' <space> <tab> <newline>
```

and the following may need to be quoted under certain circumstances. That is, these characters may be special depending on conditions described elsewhere in this volume of POSIX.1-200x:

```
* ? [ # ~ = %
```

The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The here-document represents another form of quoting; see [Section 2.7.4](#) (on page 2260).

2.2.1 Escape Character (Backslash)

A backslash that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> follows the backslash, the shell shall interpret this as line continuation. The backslash and <newline>s shall be removed before splitting the input into tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by any white space, it cannot serve as a token separator.

2.2.2 Single-Quotes

Enclosing characters in single-quotes (' ') shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

2.2.3 Double-Quotes

Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters within the double-quotes, with the exception of the characters dollar sign, backquote, and backslash, as follows:

§ The dollar sign shall retain its special meaning introducing parameter expansion (see [Section 2.6.2](#), on page 2254), a form of command substitution (see [Section 2.6.3](#), on page 2256), and arithmetic expansion (see [Section 2.6.4](#), on page 2257).

The input characters within the quoted string that are also enclosed between "\$ (" and the matching ') ' shall not be affected by the double-quotes, but rather shall define that command whose output replaces the "\$ (. . .) " when the word is expanded. The tokenizing rules in [Section 2.3](#) (on page 2247), not including the alias substitutions in [Section 2.3.1](#) (on page 2248), shall be applied recursively to find the matching ') '.

Within the string of characters from an enclosed "\$ { " to the matching ' } ' , an even number of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash character shall be used to escape a literal ' { ' or ' } ' . The rule in [Section 2.6.2](#) (on page 2254) shall be used to determine the matching ' } ' .

` The backquote shall retain its special meaning introducing the other form of command substitution (see [Section 2.6.3](#), on page 2256). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a backslash, having escape characters removed, defines that command whose output replaces

70916 " ` . . . ` " when the word is expanded. Either of the following cases produces undefined
70917 results:

- 70918 • A single-quoted or double-quoted string that begins, but does not end, within the
70919 " ` . . . ` " sequence
- 70920 • A " ` . . . ` " sequence that begins, but does not end, within the same double-quoted
70921 string

70922 \ The backslash shall retain its special meaning as an escape character (see [Section 2.2.1](#), on
70923 page 2246) only when followed by one of the following characters when considered special:

70924 \$ ` " \ <newline>

70925 The application shall ensure that a double-quote is preceded by a backslash to be included
70926 within double-quotes. The parameter '@' has special meaning inside double-quotes and is
70927 described in [Section 2.5.2](#) (on page 2250).

70928 2.3 Token Recognition

70929 The shell shall read its input in terms of lines from a file, from a terminal in the case of an
70930 interactive shell, or from a string in the case of *sh -c* or *system()*. The input lines can be of
70931 unlimited length. These lines shall be parsed using two major modes: ordinary token recognition
70932 and processing of here-documents.

70933 When an **io_here** token has been recognized by the grammar (see [Section 2.10](#), on page 2271),
70934 one or more of the subsequent lines immediately following the next **NEWLINE** token form the
70935 body of one or more here-documents and shall be parsed according to the rules of [Section 2.7.4](#)
70936 (on page 2260).

70937 When it is not processing an **io_here**, the shell shall break its input into tokens by applying the
70938 first applicable rule below to the next character in its input. The token shall be from the current
70939 position in the input until a token is delimited according to one of the rules below; the characters
70940 forming the token are exactly those in the input, including any quoting characters. If it is
70941 indicated that a token is delimited, and no characters have been included in a token, processing
70942 shall continue until an actual token is delimited.

- 70943 1. If the end of input is recognized, the current token shall be delimited. If there is no
70944 current token, the end-of-input indicator shall be returned as the token.
- 70945 2. If the previous character was used as part of an operator and the current character is not
70946 quoted and can be used with the current characters to form an operator, it shall be used as
70947 part of that (operator) token.
- 70948 3. If the previous character was used as part of an operator and the current character cannot
70949 be used with the current characters to form an operator, the operator containing the
70950 previous character shall be delimited.
- 70951 4. If the current character is backslash, single-quote, or double-quote (` ` , ' ' , or " ") and
70952 it is not quoted, it shall affect quoting for subsequent characters up to the end of the
70953 quoted text. The rules for quoting are as described in [Section 2.2](#) (on page 2246). During
70954 token recognition no substitutions shall be actually performed, and the result token shall
70955 contain exactly the characters that appear in the input (except for <newline> joining),
70956 unmodified, including any embedded or enclosing quotes or substitution operators,
70957 between the quote mark and the end of the quoted text. The token shall not be delimited
70958 by the end of the quoted field.

- 70959 5. If the current character is an unquoted '\$' or '\', the shell shall identify the start of any
 70960 candidates for parameter expansion (Section 2.6.2, on page 2254), command substitution
 70961 (Section 2.6.3, on page 2256), or arithmetic expansion (Section 2.6.4, on page 2257) from
 70962 their introductory unquoted character sequences: '\$' or "\${", "\$(" or '\', and "\$(",
 70963 respectively. The shell shall read sufficient input to determine the end of the unit to be
 70964 expanded (as explained in the cited sections). While processing the characters, if
 70965 instances of expansions or quoting are found nested within the substitution, the shell
 70966 shall recursively process them in the manner specified for the construct that is found. The
 70967 characters found from the beginning of the substitution to its end, allowing for any
 70968 recursion necessary to recognize embedded constructs, shall be included unmodified in
 70969 the result token, including any embedded or enclosing substitution operators or quotes.
 70970 The token shall not be delimited by the end of the substitution.
- 70971 6. If the current character is not quoted and can be used as the first character of a new
 70972 operator, the current token (if any) shall be delimited. The current character shall be used
 70973 as the beginning of the next (operator) token.
- 70974 7. If the current character is an unquoted <newline>, the current token shall be delimited.
- 70975 8. If the current character is an unquoted <blank>, any token containing the previous
 70976 character is delimited and the current character shall be discarded.
- 70977 9. If the previous character was part of a word, the current character shall be appended to
 70978 that word.
- 70979 10. If the current character is a '#', it and all subsequent characters up to, but excluding, the
 70980 next <newline> shall be discarded as a comment. The <newline> that ends the line is not
 70981 considered part of the comment.
- 70982 11. The current character is used as the start of a new word.

70983 Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page
 70984 2271).

70985 2.3.1 Alias Substitution

70986 After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on
 70987 page 2271), a resulting word that is identified to be the command name word of a simple
 70988 command shall be examined to determine whether it is an unquoted, valid alias name. However,
 70989 reserved words in correct grammatical context shall not be candidates for alias substitution. A
 70990 valid alias name (see XBD Section 3.10, on page 34) shall be one that has been defined by the
 70991 *alias* utility and not subsequently undefined using *unalias*. Implementations also may provide
 70992 predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in
 70993 recursive aliasing, if the shell is not currently processing an alias of the same name, the word
 70994 shall be replaced by the value of the alias; otherwise, it shall not be replaced.

70995 If the value of the alias replacing the word ends in a <blank>, the shell shall check the next
 70996 command word for alias substitution; this process shall continue until a word is found that is
 70997 not a valid alias or an alias value does not end in a <blank>.

70998 When used as specified by this volume of POSIX.1-200x, alias definitions shall not be inherited
 70999 by separate invocations of the shell or by the utility execution environments invoked by the
 71000 shell; see Section 2.12 (on page 2277).

71001 **2.4 Reserved Words**

71002 Reserved words are words that have special meaning to the shell; see [Section 2.9](#) (on page 2263).
 71003 The following words shall be recognized as reserved words:

71004	!	do	esac	in
71005	{	done	fi	then
71006	}	elif	for	until
71007	case	else	if	while

71008 This recognition shall only occur when none of the characters is quoted and when the word is
 71009 used as:

- 71010 • The first word of a command
- 71011 • The first word following one of the reserved words other than **case**, **for**, or **in**
- 71012 • The third word in a **case** command (only **in** is valid in this case)
- 71013 • The third word in a **for** command (only **in** and **do** are valid in this case)

71014 See the grammar in [Section 2.10](#) (on page 2271).

71015 The following words may be recognized as reserved words on some implementations (when
 71016 none of the characters are quoted), causing unspecified results:

71017	[[]]	function	select
-------	-----------	-----------	-----------------	---------------

71018 Words that are the concatenation of a name and a colon (' : ') are reserved; their use produces
 71019 unspecified results.

71020 **2.5 Parameters and Variables**

71021 A parameter can be denoted by a name, a number, or one of the special characters listed in
 71022 [Section 2.5.2](#) (on page 2250). A variable is a parameter denoted by a name.

71023 A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can
 71024 only be unset by using the *unset* special built-in command.

71025 **2.5.1 Positional Parameters**

71026 A positional parameter is a parameter denoted by the decimal value represented by one or more
 71027 digits, other than the single digit 0. The digits denoting the positional parameters shall always
 71028 be interpreted as a decimal value, even if there is a leading zero. When a positional parameter
 71029 with more than one digit is specified, the application shall enclose the digits in braces (see
 71030 [Section 2.6.2](#), on page 2254). Positional parameters are initially assigned when the shell is
 71031 invoked (see *sh*), temporarily replaced when a shell function is invoked (see [Section 2.9.5](#), on
 71032 page 2270), and can be reassigned with the *set* special built-in command.

71033 **2.5.2 Special Parameters**

71034 Listed below are the special parameters and the values to which they shall expand. Only the
71035 values of the special parameters are listed; see [Section 2.6](#) (on page 2253) for a detailed summary
71036 of all the stages involved in expanding words.

- 71037 @ Expands to the positional parameters, starting from one. When the expansion occurs within
71038 double-quotes, and where field splitting (see [Section 2.6.5](#), on page 2258) is performed, each
71039 positional parameter shall expand as a separate field, with the provision that the expansion
71040 of the first parameter shall still be joined with the beginning part of the original word
71041 (assuming that the expanded parameter was embedded within a word), and the expansion
71042 of the last parameter shall still be joined with the last part of the original word. If there are
71043 no positional parameters, the expansion of '@' shall generate zero fields, even when '@' is
71044 double-quoted.
- 71045 * Expands to the positional parameters, starting from one. When the expansion occurs within
71046 a double-quoted string (see [Section 2.2.3](#), on page 2246), it shall expand to a single field with
71047 the value of each parameter separated by the first character of the *IFS* variable, or by a
71048 <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its
71049 first character does not exist, so the parameter values are concatenated.
- 71050 # Expands to the decimal number of positional parameters. The command name (parameter
71051 0) shall not be counted in the number given by '# ' because it is a special parameter, not a
71052 positional parameter.
- 71053 ? Expands to the decimal exit status of the most recent pipeline (see [Section 2.9.2](#), on page
71054 2265).
- 71055 – (Hyphen.) Expands to the current option flags (the single-letter option names concatenated
71056 into a string) as specified on invocation, by the *set* special built-in command, or implicitly
71057 by the shell.
- 71058 \$ Expands to the decimal process ID of the invoked shell. In a subshell (see [Section 2.12](#), on
71059 page 2277), '\$ ' shall expand to the same value as that of the current shell.
- 71060 ! Expands to the decimal process ID of the most recent background command (see [Section
71061 2.9.3](#), on page 2266) executed from the current shell. (For example, background commands
71062 executed from subshells do not affect the value of "\$!" in the current shell environment.)
71063 For a pipeline, the process ID is that of the last command in the pipeline.
- 71064 0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3074) for a detailed
71065 description of how this name is derived.

71066 See the description of the *IFS* variable in [Section 2.5.3](#).

71067 **2.5.3 Shell Variables**

71068 Variables shall be initialized from the environment (as defined by XBD [Chapter 8](#) (on page 159) |
71069 and the *exec* function in the System Interfaces volume of POSIX.1-200x) and can be given new
71070 values with variable assignment commands. If a variable is initialized from the environment, it
71071 shall be marked for export immediately; see the *export* special built-in. New variables can be
71072 defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name*
71073 parameter in a **for** loop, with the $\${name=word}$ expansion, or with other mechanisms provided
71074 as implementation extensions.

71075 The following variables shall affect the execution of the shell:

71076	UP XSI	ENV	The processing of the <i>ENV</i> shell variable shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option.
71077			
71078			
71079			This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2 , on page 2254) by the shell and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of <i>ENV</i> is not an absolute pathname, the results are unspecified. <i>ENV</i> shall be ignored if the user's real and effective user IDs or real and effective group IDs are different.
71080			
71081			
71082			
71083			
71084			
71085			
71086		HOME	The pathname of the user's home directory. The contents of <i>HOME</i> are used in tilde expansion (see Section 2.6.1 , on page 2253).
71087			
71088		IFS	A string treated as a list of characters that is used for field splitting and to split lines into fields with the <i>read</i> command.
71089			
71090			If <i>IFS</i> is not set, it shall behave as normal for an unset variable, except that field splitting by the shell and line splitting by the <i>read</i> command shall be performed as if the value of <i>IFS</i> is <space><tab><newline>; see Section 2.6.5 (on page 2258).
71091			
71092			
71093			
71094			Implementations may ignore the value of <i>IFS</i> in the environment, or the absence of <i>IFS</i> from the environment, at the time the shell is invoked, in which case the shell shall set <i>IFS</i> to <space><tab><newline> when it is invoked.
71095			
71096			
71097		LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)
71098			
71099			
71100			
71101		LC_ALL	The value of this variable overrides the <i>LC_*</i> variables and <i>LANG</i> , as described in XBD Chapter 8 (on page 159).
71102			
71103		LC_COLLATE	Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
71104			
71105		LC_CTYPE	Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class alpha) and <blank>s (character class blank), and the behavior of character classes within pattern matching. Changing the value of <i>LC_CTYPE</i> after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells. Invoking a shell script or performing <i>exec sh</i> subjects the new shell to the changes in <i>LC_CTYPE</i> .
71106			
71107			
71108			
71109			
71110			
71111			
71112			
71113		LC_MESSAGES	Determine the language in which messages should be written.
71114		LINENO	Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. If the user unsets or resets <i>LINENO</i> , the variable may lose its special meaning for the life of the shell. If the shell is not currently executing a script or function, the value of <i>LINENO</i> is unspecified. This volume of POSIX.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
71115			
71116			
71117			
71118			
71119			
71120			

71121	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
71122			
71123		<i>PATH</i>	A string formatted as described in XBD Chapter 8 (on page 159), used to effect command interpretation; see Section 2.9.1.1 (on page 2264).
71124			
71125		<i>PPID</i>	Set by the shell to the decimal process ID of the process that invoked this shell. In a subshell (see Section 2.12, on page 2277), <i>PPID</i> shall be set to the same value as that of the parent of the current shell. For example, <i>echo \$PPID</i> and (<i>echo \$PPID</i>) would produce the same value. This volume of POSIX.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
71126			
71127			
71128			
71129			
71130			
71131		<i>PS1</i>	Each time an interactive shell is ready to read a command, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be "\$ ". For users who have specific additional implementation-defined privileges, the default may be another, implementation-defined value. The shell shall replace each instance of the character '!' in <i>PS1</i> with the history file number of the next command to be typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal character '!' in the prompt. This volume of POSIX.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
71132			
71133			
71134			
71135			
71136			
71137			
71138			
71139			
71140			
71141		<i>PS2</i>	Each time the user enters a <newline> prior to completing a command line in an interactive shell, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value is "> ". This volume of POSIX.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
71142			
71143			
71144			
71145			
71146		<i>PS4</i>	When an execution trace (<i>set -x</i>) is being performed in an interactive shell, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value is "+ ". This volume of POSIX.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.
71147			
71148			
71149			
71150			
71151		<i>PWD</i>	Set by the shell to be an absolute pathname of the current working directory, containing no components of type symbolic link, no components that are dot, and no components that are dot-dot when the shell is initialized. If an application sets or unsets the value of <i>PWD</i> , the behaviors of the <i>cd</i> and <i>pwd</i> utilities are unspecified.
71152			
71153			
71154			
71155			

2.6 Word Expansions

This section describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following sections.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word expand to a single field. It is only field splitting or pathname expansion that can create multiple fields from a single word. The single exception to this rule is the expansion of the special parameter '@' within double-quotes, as described in [Section 2.5.2](#) (on page 2250).

The order of word expansion shall be as follows:

1. Tilde expansion (see [Section 2.6.1](#)), parameter expansion (see [Section 2.6.2](#), on page 2254), command substitution (see [Section 2.6.3](#), on page 2256), and arithmetic expansion (see [Section 2.6.4](#), on page 2257) shall be performed, beginning to end. See item 5 in [Section 2.3](#) (on page 2247).
2. Field splitting (see [Section 2.6.5](#), on page 2258) shall be performed on the portions of the fields generated by step 1, unless *IFS* is null.
3. Pathname expansion (see [Section 2.6.6](#), on page 2259) shall be performed, unless *set -f* is in effect.
4. Quote removal (see [Section 2.6.7](#), on page 2259) shall always be performed last.

The expansions described in this section shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The '\$' character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted '\$' is followed by a character that is either not numeric, the name of one of the special parameters (see [Section 2.5.2](#), on page 2250), a valid first character of a variable name, a left curly brace ('{') or a left parenthesis, the result is unspecified.

2.6.1 Tilde Expansion

A “tilde-prefix” consists of an unquoted tilde character at the beginning of a word, followed by all of the characters preceding the first unquoted slash in the word, or all the characters in the word if there is no slash. In an assignment (see [XBD Section 4.22](#), on page 106), multiple tilde-prefixes can be used: at the beginning of the word (that is, following the equal sign of the assignment), following any unquoted colon, or both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde are treated as a possible login name from the user database. A portable login name cannot contain characters outside the set given in the description of the *LOGNAME* environment variable in [XBD Section 8.3](#) (on page 163). If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name obtained using the *getpwnam()* function as defined in the System Interfaces volume of POSIX.1-200x. If the system does not recognize the login name, the results are undefined.

2.6.2 Parameter Expansion

The format for parameter expansion is as follows:

```
${expression}
```

where *expression* consists of all characters until the matching `'}'`. Any `'}'` escaped by a backslash or within a quoted string, and characters in embedded arithmetic expansions, command substitutions, and variable expansions, shall not be examined in determining the matching `'}'`.

The simplest form for parameter expansion is:

```
${parameter}
```

The value, if any, of *parameter* shall be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when *parameter* is followed by a character that could be interpreted as part of the name. The matching closing brace shall be determined by counting brace levels, skipping over enclosed quoted strings, and command substitutions.

If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest valid name (see XBD Section 3.229, on page 65), whether or not the symbol represented by that name exists.

If a parameter expansion occurs inside double-quotes:

- Pathname expansion shall not be performed on the results of the expansion.
- Field splitting shall not be performed on the results of the expansion, with the exception of `'@'`; see Section 2.5.2 (on page 2250).

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of *word* is needed (based on the state of *parameter*, as described below), *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. If *word* is not needed, it shall not be expanded. The `'}'` character that delimits the following parameter expansion modifications shall be determined as described previously in this section and in Section 2.2.3 (on page 2246). (For example, `${foo-bar}xyz` would result in the expansion of `foo` followed by the string `xyz` if `foo` is set, else the string `"barxyz"`).

`${parameter:-word}` **Use Default Values.** If *parameter* is unset or null, the expansion of *word* shall be substituted; otherwise, the value of *parameter* shall be substituted.

`${parameter:=word}` **Assign Default Values.** If *parameter* is unset or null, the expansion of *word* shall be assigned to *parameter*. In all cases, the final value of *parameter* shall be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.

`${parameter:?[word]}` **Indicate Error if Null or Unset.** If *parameter* is unset or null, the expansion of *word* (or a message indicating it is unset if *word* is omitted) shall be written to standard error and the shell exits with a non-zero exit status. Otherwise, the value of *parameter* shall be substituted. An interactive shell need not exit.

`${parameter:+word}` **Use Alternative Value.** If *parameter* is unset or null, null shall be substituted; otherwise, the expansion of *word* shall be substituted.

In the parameter expansions shown previously, use of the colon in the format shall result in a test for a parameter that is unset or null; omission of the colon shall result in a test for a

71242 parameter that is only unset. The following table summarizes the effect of the colon:

71243
71244

	<i>parameter</i> Set and Not Null	<i>parameter</i> Set But Null	<i>parameter</i> Unset
71245 71246	<code>\${parameter:-word}</code> <code>\${parameter-word}</code>	substitute <i>parameter</i> substitute <i>parameter</i>	substitute <i>word</i> substitute <i>word</i>
71247 71248	<code>\${parameter:=word}</code> <code>\${parameter=word}</code>	substitute <i>parameter</i> substitute <i>parameter</i>	assign <i>word</i> assign <i>word</i>
71249 71250	<code>\${parameter?word}</code> <code>\${parameter?word}</code>	substitute <i>parameter</i> substitute <i>parameter</i>	error, exit error, exit
71251 71252	<code>\${parameter+word}</code> <code>\${parameter+word}</code>	substitute <i>word</i> substitute <i>word</i>	substitute null substitute null

71253
71254

In all cases shown with “substitute”, the expression is replaced with the value shown. In all cases shown with “assign”, *parameter* is assigned that value, which also replaces the expression.

71255
71256
71257

`#{parameter}` **String Length.** The length in characters of the value of *parameter* shall be substituted. If *parameter* is '*' or '@', the result of the expansion is unspecified.

71258
71259
71260
71261
71262
71263

The following four varieties of parameter expansion provide for substring processing. In each case, pattern matching notation (see Section 2.13, on page 2278), rather than regular expression notation, shall be used to evaluate the patterns. If *parameter* is '*' or '@', the result of the expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect.

71264
71265
71266

`{parameter%word}` **Remove Smallest Suffix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the suffix matched by the *pattern* deleted.

71267
71268
71269

`{parameter%%word}` **Remove Largest Suffix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the suffix matched by the *pattern* deleted.

71270
71271
71272

`{parameter#word}` **Remove Smallest Prefix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the prefix matched by the *pattern* deleted.

71273
71274
71275

`{parameter##word}` **Remove Largest Prefix Pattern.** The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the prefix matched by the *pattern* deleted.

71276

Examples

71277
71278
71279

`{parameter :-word}`
In this example, *ls* is executed only if *x* is null or unset. (The `$(ls)` command substitution notation is explained in Section 2.6.3 (on page 2256).)

71280

```
{x:-$(ls)}
```

71281
71282
71283
71284

```
{parameter :=word}
unset X
echo ${X:=abc}
abc
```

```

71285  ${parameter:?word}
71286      unset posix
71287      echo ${posix:?}
71288      sh: posix: parameter null or not set

```

```

71289  ${parameter:+word}
71290      set a b c
71291      echo ${3:+posix}
71292      posix

```

```

71293  $#parameter}
71294      HOME=/usr/posix
71295      echo $#HOME}
71296      10

```

```

71297  ${parameter%word}
71298      x=file.c
71299      echo ${x%.c}.o
71300      file.o

```

```

71301  ${parameter%%word}
71302      x=posix/src/std
71303      echo ${x%%/*}
71304      posix

```

```

71305  ${parameter#word}
71306      x=$HOME/src/cmd
71307      echo ${x#$HOME}
71308      /src/cmd

```

```

71309  ${parameter##word}
71310      x=/one/two/three
71311      echo ${x##*/}
71312      three

```

The double-quoting of patterns is different depending on where the double-quotes are placed:

```

71313  " ${x#*} "   The asterisk is a pattern character.
71314  ${x#"*" }  The literal asterisk is quoted and not special.
71315

```

2.6.3 Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution shall occur when the command is enclosed as follows:

```

71320  $(command)

```

or (backquoted version):

```

71322  `command`

```

The shell shall expand the command substitution by executing *command* in a subshell environment (see [Section 2.12](#), on page 2277) and replacing the command substitution (the text of *command* plus the enclosing "\$ ()" or backquotes) with the standard output of the command, removing sequences of one or more <newline>s at the end of the substitution. Embedded <newline>s before the end of the output shall not be removed; however, they may be treated as field delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting

71329 that is in effect.

71330 Within the backquoted style of command substitution, backslash shall retain its literal meaning,
71331 except when followed by: '\$', '`', or '\`' (dollar sign, backquote, backslash). The search for
71332 the matching backquote shall be satisfied by the first unquoted non-escaped backquote; during
71333 this search, if a non-escaped backquote is encountered within a shell comment, a here-
71334 document, an embedded command substitution of the \$(*command*) form, or a quoted string,
71335 undefined results occur. A single-quoted or double-quoted string that begins, but does not end,
71336 within the "`...`" sequence produces undefined results.

71337 With the \$(*command*) form, all characters following the open parenthesis to the matching closing
71338 parenthesis constitute the *command*. Any valid shell script can be used for *command*, except a
71339 script consisting solely of redirections which produces unspecified results.

71340 The results of command substitution shall not be processed for further tilde expansion,
71341 parameter expansion, command substitution, or arithmetic expansion. If a command
71342 substitution occurs inside double-quotes, field splitting and pathname expansion shall not be
71343 performed on the results of the substitution.

71344 Command substitution can be nested. To specify nesting within the backquoted version, the
71345 application shall precede the inner backquotes with backslashes, for example:

```
71346 \ `command` `
```

71347 If the command substitution consists of a single subshell, such as:

```
71348 $( (command) )
```

71349 a conforming application shall separate the "\$(" and "(" into two tokens (that is, separate
71350 them with white space). This is required to avoid any ambiguities with arithmetic expansion.

71351 2.6.4 Arithmetic Expansion

71352 Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and
71353 substituting its value. The format for arithmetic expansion shall be as follows:

```
71354 $( (expression) )
```

71355 The expression shall be treated as if it were in double-quotes, except that a double-quote inside
71356 the expression is not treated specially. The shell shall expand all tokens in the expression for
71357 parameter expansion, command substitution, and quote removal.

71358 Next, the shell shall treat this as an arithmetic expression and substitute the value of the
71359 expression. The arithmetic expression shall be processed according to the rules given in [Section](#)
71360 [1.1.2.1](#) (on page 2231), with the following exceptions:

- 71361 • Only signed long integer arithmetic is required.
- 71362 • Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in
71363 the ISO C standard, Section 6.4.4.1 are required to be recognized as constants.
- 71364 • The *sizeof*() operator and the prefix and postfix "++" and "--" operators are not required.
- 71365 • Selection, iteration, and jump statements are not supported.

71366 All changes to variables in an arithmetic expression shall be in effect after the arithmetic
71367 expansion, as in the parameter expansion "\${x=value}".

71368 If the shell variable *x* contains a value that forms a valid integer constant, then the arithmetic
71369 expansions "\$({x})" and "\$({\$x})" shall return the same value.

71370 As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell

71371 may use a signed integer type with a rank larger than the rank of **signed long**. The shell may
 71372 use a real-floating type instead of **signed long** as long as it does not affect the results in cases
 71373 where there is no overflow. If the expression is invalid, the expansion fails and the shell shall
 71374 write a message to standard error indicating the failure.

71375 Examples

71376 A simple example using arithmetic expansion:

```
71377 # repeat a command 100 times
71378 x=100
71379 while [ $x -gt 0 ]
71380 do
71381     command
71382     x=$(( $x-1 ))
71383 done
```

71384 2.6.5 Field Splitting

71385 After parameter expansion (Section 2.6.2, on page 2254), command substitution (Section 2.6.3, on
 71386 page 2256), and arithmetic expansion (Section 2.6.4, on page 2257), the shell shall scan the results
 71387 of expansions and substitutions that did not occur in double-quotes for field splitting and
 71388 multiple fields can result.

71389 The shell shall treat each character of the *IFS* as a delimiter and use the delimiters as field
 71390 terminators to split the results of parameter expansion and command substitution into fields.

- 71391 1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of
 71392 <space>s, <tab>s, or <newline>s at the beginning or end of the input shall be ignored and
 71393 any sequence of those characters within the input shall delimit a field. For example, the
 71394 input:

```
71395 <newline><space><tab>foo<tab><tab>bar<space>
```

71396 yields two fields, **foo** and **bar**.

- 71397 2. If the value of *IFS* is null, no field splitting shall be performed.
- 71398 3. Otherwise, the following rules shall be applied in sequence. The term “*IFS* white space”
 71399 is used to mean any sequence (zero or more instances) of white space characters that are
 71400 in the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of
 71401 <space>s and <tab>s is considered *IFS* white space).
 - 71402 a. *IFS* white space shall be ignored at the beginning and end of the input.
 - 71403 b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along
 71404 with any adjacent *IFS* white space, shall delimit a field, as described previously.
 - 71405 c. Non-zero-length *IFS* white space shall delimit a field.

71406 2.6.6 Pathname Expansion

71407 After field splitting, if *set -f* is not in effect, each field in the resulting command line shall be
 71408 expanded using the algorithm described in [Section 2.13](#) (on page 2278), qualified by the rules in
 71409 [Section 2.13.3](#) (on page 2279).

71410 2.6.7 Quote Removal

71411 The quote characters: `'\'`, `'\''`, and `'\"'` (backslash, single-quote, double-quote) that were
 71412 present in the original word shall be removed unless they have themselves been quoted.

71413 2.7 Redirection

71414 Redirection is used to open and close files for the current shell execution environment (see
 71415 [Section 2.12](#), on page 2277) or for any command. Redirection operators can be used with
 71416 numbers representing file descriptors (see [XBD Section 3.165](#), on page 56) as described below.

71417 The overall format used for redirection is:

71418 `[n]redir-op word`

71419 The number *n* is an optional decimal number designating the file descriptor number; the
 71420 application shall ensure it is delimited from any preceding text and immediately precede the
 71421 redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the
 71422 redirection expression. For example:

71423 `echo \2>a`

71424 writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is
 71425 recognized. For example:

71426 `echo 2\>a`

71427 writes the characters `2>a` to standard output. The optional number, redirection operator, and
 71428 *word* shall not appear in the arguments provided to the command to be executed (if any).

71429 Open files are represented by decimal numbers starting with zero. The largest possible value is
 71430 implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for
 71431 use by the application. These numbers are called "file descriptors". The values 0, 1, and 2 have
 71432 special meaning and conventional uses and are implied by certain redirection operations; they
 71433 are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs
 71434 usually take their input from standard input, and write output on standard output. Error
 71435 messages are usually written on standard error. The redirection operators can be preceded by
 71436 one or more digits (with no intervening <blank>s allowed) to designate the file descriptor
 71437 number.

71438 If the redirection operator is `"<<"` or `"<<-"`, the word that follows the redirection operator shall
 71439 be subjected to quote removal; it is unspecified whether any of the other expansions occur. For
 71440 the other redirection operators, the word that follows the redirection operator shall be subjected
 71441 to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and
 71442 quote removal. Pathname expansion shall not be performed on the word by a non-interactive
 71443 shell; an interactive shell may perform it, but shall do so only when the expansion would result
 71444 in one word.

71445 If more than one redirection operator is specified with a command, the order of evaluation is
 71446 from beginning to end.

71447 A failure to open or create a file shall cause a redirection to fail.

71448 2.7.1 Redirecting Input

71449 Input redirection shall cause the file whose name results from the expansion of *word* to be
71450 opened for reading on the designated file descriptor, or standard input if the file descriptor is
71451 not specified.

71452 The general format for redirecting input is:

71453 `[n]<word`

71454 where the optional *n* represents the file descriptor number. If the number is omitted, the
71455 redirection shall refer to standard input (file descriptor 0).

71456 2.7.2 Redirecting Output

71457 The two general formats for redirecting output are:

71458 `[n]>word`

71459 `[n]>|word`

71460 where the optional *n* represents the file descriptor number. If the number is omitted, the
71461 redirection shall refer to standard output (file descriptor 1).

71462 Output redirection using the '>' format shall fail if the *noclobber* option is set (see the
71463 description of *set -C*) and the file named by the expansion of *word* exists and is a regular file.
71464 Otherwise, redirection using the '>' or '>|' formats shall cause the file whose name results
71465 from the expansion of *word* to be created and opened for output on the designated file
71466 descriptor, or standard output if none is specified. If the file does not exist, it shall be created;
71467 otherwise, it shall be truncated to be an empty file after being opened.

71468 2.7.3 Appending Redirected Output

71469 Appended output redirection shall cause the file whose name results from the expansion of
71470 *word* to be opened for output on the designated file descriptor. The file is opened as if the *open()*
71471 function as defined in the System Interfaces volume of POSIX.1-200x was called with the
71472 *O_APPEND* flag. If the file does not exist, it shall be created.

71473 The general format for appending redirected output is as follows:

71474 `[n]>>word`

71475 where the optional *n* represents the file descriptor number. If the number is omitted, the
71476 redirection refers to standard output (file descriptor 1).

71477 2.7.4 Here-Document

71478 The redirection operators "<<" and "<<-" both allow redirection of lines contained in a shell
71479 input file, known as a "here-document", to the input of a command.

71480 The here-document shall be treated as a single word that begins after the next <newline> and
71481 continues until there is a line containing only the delimiter and a <newline>, with no <blank>s
71482 in between. Then the next here-document starts, if there is one. The format is as follows:

71483 `[n]<<word`

71484 `here-document`

71485 `delimiter`

71486 where the optional *n* represents the file descriptor number. If the number is omitted, the here-
71487 document refers to standard input (file descriptor 0).

71488 If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on
 71489 *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the
 71490 *word* itself.

71491 If no characters in *word* are quoted, all lines of the here-document shall be expanded for
 71492 parameter expansion, command substitution, and arithmetic expansion. In this case, the
 71493 backslash in the input behaves as the backslash inside double-quotes (see Section 2.2.3, on page
 71494 2246). However, the double-quote character (' " ') shall not be treated specially within a here-
 71495 document, except when the double-quote appears within "\$ ()", "` `", or "\$ { }".

71496 If the redirection symbol is "<<-", all leading <tab>s shall be stripped from input lines and the
 71497 line containing the trailing delimiter. If more than one "<<" or "<<-" operator is specified on a
 71498 line, the here-document associated with the first operator shall be supplied first by the
 71499 application and shall be read first by the shell.

71500 Examples

71501 An example of a here-document follows:

```
71502 cat <<eof1; cat <<eof2
71503 Hi,
71504 eof1
71505 Helene.
71506 eof2
```

71507 2.7.5 Duplicating an Input File Descriptor

71508 The redirection operator:

```
71509 [n]<&word
```

71510 shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one
 71511 or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be
 71512 made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a
 71513 file descriptor already open for input, a redirection error shall result; see Section 2.8.1 (on page
 71514 2262). If *word* evaluates to '-', file descriptor *n*, or standard input if *n* is not specified, shall be
 71515 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*
 71516 evaluates to something else, the behavior is unspecified.

71517 2.7.6 Duplicating an Output File Descriptor

71518 The redirection operator:

```
71519 [n]>&word
```

71520 shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to
 71521 one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall
 71522 be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent
 71523 a file descriptor already open for output, a redirection error shall result; see Section 2.8.1 (on
 71524 page 2262). If *word* evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is
 71525 closed. Attempts to close a file descriptor that is not open shall not constitute an error. If *word*
 71526 evaluates to something else, the behavior is unspecified.

71527 2.7.7 Open File Descriptors for Reading and Writing

71528 The redirection operator:

71529 `[n]<>word`

71530 shall cause the file whose name is the expansion of *word* to be opened for both reading and
71531 writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does
71532 not exist, it shall be created.

71533 2.8 Exit Status and Errors

71534 2.8.1 Consequences of Shell Errors

71535 For a non-interactive shell, an error condition encountered by a special built-in (see [Section 2.14](#),
71536 on page 2280) or other type of utility shall cause the shell to write a diagnostic message to
71537 standard error and exit as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	Shall exit	Shall exit
Utility syntax error (option or operand error)	Shall exit	Shall not exit
Redirection error	Shall exit	Shall not exit
Variable assignment error	Shall exit	Shall not exit
Expansion error	Shall exit	Shall exit
Command not found	N/A	May exit
Dot script not found	Shall exit	N/A

71546 An expansion error is one that occurs when the shell expansions defined in [Section 2.6](#) (on page
71547 2253) are carried out (for example, " `${x!y}` ", because `!` is not a valid operator); an
71548 implementation may treat these as syntax errors if it is able to detect them during tokenization,
71549 rather than during expansion.

71550 If any of the errors shown as "shall exit" or "(may) exit" occur in a subshell, the subshell shall
71551 (respectively may) exit with a non-zero status, but the script containing the subshell shall not
71552 exit because of the error.

71553 In all of the cases shown in the table, an interactive shell shall write a diagnostic message to
71554 standard error without exiting.

71555 2.8.2 Exit Status for Commands

71556 Each command has an exit status that can influence the behavior of other shell commands. The
71557 exit status of commands that are not utilities is documented in this section. The exit status of the
71558 standard utilities is documented in their respective sections.

71559 If a command is not found, the exit status shall be 127. If the command name is found, but it is
71560 not an executable utility, the exit status shall be 126. Applications that invoke utilities without
71561 using the shell should use these exit status values to report similar errors.

71562 If a command fails during word expansion or redirection, its exit status shall be greater than
71563 zero.

71564 Internally, for purposes of deciding whether a command exits with a non-zero exit status, the
71565 shell shall recognize the entire status value retrieved for the command by the equivalent of the

71566 *wait()* function WEXITSTATUS macro (as defined in the System Interfaces volume of
 71567 POSIX.1-200x). When reporting the exit status with the special parameter '??', the shell shall
 71568 report the full eight bits of exit status available. The exit status of a command that terminated
 71569 because it received a signal shall be reported as greater than 128.

71570 2.9 Shell Commands

71571 This section describes the basic structure of shell commands. The following command
 71572 descriptions each describe a format of the command that is only used to aid the reader in
 71573 recognizing the command type, and does not formally represent the syntax. Each description
 71574 discusses the semantics of the command; for a formal definition of the command language,
 71575 consult [Section 2.10](#) (on page 2271).

71576 A *command* is one of the following:

- 71577 • Simple command (see [Section 2.9.1](#))
- 71578 • Pipeline (see [Section 2.9.2](#), on page 2265)
- 71579 • List compound-list (see [Section 2.9.3](#), on page 2266)
- 71580 • Compound command (see [Section 2.9.4](#), on page 2268)
- 71581 • Function definition (see [Section 2.9.5](#), on page 2270)

71582 Unless otherwise stated, the exit status of a command shall be that of the last simple command
 71583 executed by the command. There shall be no limit on the size of any shell command other than
 71584 that imposed by the underlying system (memory constraints, {ARG_MAX}, and so on).

71585 2.9.1 Simple Commands

71586 A “simple command” is a sequence of optional variable assignments and redirections, in any
 71587 sequence, optionally followed by words and redirections, terminated by a control operator.

71588 When a given simple command is required to be executed (that is, when any conditional
 71589 construct such as an AND-OR list or a **case** statement has not bypassed the simple command),
 71590 the following expansions, assignments, and redirections shall all be performed from the
 71591 beginning of the command text to the end:

- 71592 1. The words that are recognized as variable assignments or redirections according to
 71593 [Section 2.10.2](#) (on page 2272) are saved for processing in steps 3 and 4.
- 71594 2. The words that are not variable assignments or redirections shall be expanded. If any
 71595 fields remain following their expansion, the first field shall be considered the command
 71596 name and remaining fields are the arguments for the command.
- 71597 3. Redirections shall be performed as described in [Section 2.7](#) (on page 2259).
- 71598 4. Each variable assignment shall be expanded for tilde expansion, parameter expansion,
 71599 command substitution, arithmetic expansion, and quote removal prior to assigning the
 71600 value.

71601 In the preceding list, the order of steps 3 and 4 may be reversed for the processing of special
 71602 built-in utilities; see [Section 2.14](#) (on page 2280).

71603 If no command name results, variable assignments shall affect the current execution
 71604 environment. Otherwise, the variable assignments shall be exported for the execution
 71605 environment of the command and shall not affect the current execution environment (except for
 71606 special built-ins). If any of the variable assignments attempt to assign a value to a read-only

71607 variable, a variable assignment error shall occur. See [Section 2.8.1](#) (on page 2262) for the
71608 consequences of these errors.

71609 If there is no command name, any redirections shall be performed in a subshell environment; it
71610 is unspecified whether this subshell environment is the same one as that used for a command
71611 substitution within the command. (To affect the current execution environment, see the *exec*
71612 special built-in.) If any of the redirections performed in the current shell execution environment
71613 fail, the command shall immediately fail with an exit status greater than zero, and the shell shall
71614 write an error message indicating the failure. See [Section 2.8.1](#) (on page 2262) for the
71615 consequences of these failures on interactive and non-interactive shells.

71616 If there is a command name, execution shall continue as described in [Section 2.9.1.1](#). If there is
71617 no command name, but the command contained a command substitution, the command shall
71618 complete with the exit status of the last command substitution performed. Otherwise, the
71619 command shall complete with a zero exit status.

71620 2.9.1.1 Command Search and Execution

71621 If a simple command results in a command name and an optional list of arguments, the
71622 following actions shall be performed:

71623 1. If the command name does not contain any slashes, the first successful step in the
71624 following sequence shall occur:

71625 a. If the command name matches the name of a special built-in utility, that special
71626 built-in utility shall be invoked.

71627 b. If the command name matches the name of a function known to this shell, the
71628 function shall be invoked as described in [Section 2.9.5](#) (on page 2270). If the
71629 implementation has provided a standard utility in the form of a function, it shall
71630 not be recognized at this point. It shall be invoked in conjunction with the path
71631 search in step 1d.

71632 c. If the command name matches the name of a utility listed in the following table,
71633 that utility shall be invoked.

71634	<i>alias</i>	<i>false</i>	<i>jobs</i>	<i>read</i>	<i>wait</i>
71635	<i>bg</i>	<i>fc</i>	<i>kill</i>	<i>true</i>	
71636	<i>cd</i>	<i>fg</i>	<i>newgrp</i>	<i>umask</i>	
71637	<i>command</i>	<i>getopts</i>	<i>pwd</i>	<i>unalias</i>	

71638 d. Otherwise, the command shall be searched for using the *PATH* environment
71639 variable as described in XBD [Chapter 8](#) (on page 159):

71640 i. If the search is successful:

71641 a. If the system has implemented the utility as a regular built-in or as a
71642 shell function, it shall be invoked at this point in the path search.

71643 b. Otherwise, the shell executes the utility in a separate utility
71644 environment (see [Section 2.12](#), on page 2277) with actions equivalent
71645 to calling the *execve()* function as defined in the System Interfaces
71646 volume of POSIX.1-200x with the *path* argument set to the pathname
71647 resulting from the search, *arg0* set to the command name, and the
71648 remaining arguments set to the operands, if any.

71649 If the *execve()* function fails due to an error equivalent to the
71650 [ENOEXEC] error defined in the System Interfaces volume of
71651 POSIX.1-200x, the shell shall execute a command equivalent to

71652 having a shell invoked with the pathname resulting from the search
 71653 as its first operand, with any remaining arguments passed to the new
 71654 shell, except that the value of "\$0" in the new shell may be set to the
 71655 command name. If the executable file is not a text file, the shell may
 71656 bypass this command execution. In this case, it shall write an error
 71657 message, and shall return an exit status of 126.

71658 Once a utility has been searched for and found (either as a result of this
 71659 specific search or as part of an unspecified shell start-up activity), an
 71660 implementation may remember its location and need not search for the
 71661 utility again unless the *PATH* variable has been the subject of an assignment.
 71662 If the remembered location fails for a subsequent invocation, the shell shall
 71663 repeat the search to find the new location for the utility, if any.

71664 ii. If the search is unsuccessful, the command shall fail with an exit status of
 71665 127 and the shell shall write an error message.

71666 2. If the command name contains at least one slash, the shell shall execute the utility in a
 71667 separate utility environment with actions equivalent to calling the *execve()* function
 71668 defined in the System Interfaces volume of POSIX.1-200x with the *path* and *arg0*
 71669 arguments set to the command name, and the remaining arguments set to the operands, if
 71670 any.

71671 If the *execve()* function fails due to an error equivalent to the [ENOEXEC] error, the shell
 71672 shall execute a command equivalent to having a shell invoked with the command name
 71673 as its first operand, with any remaining arguments passed to the new shell. If the
 71674 executable file is not a text file, the shell may bypass this command execution. In this case,
 71675 it shall write an error message and shall return an exit status of 126.

71676 2.9.2 Pipelines

71677 A *pipeline* is a sequence of one or more commands separated by the control operator '|'. The
 71678 standard output of all but the last command shall be connected to the standard input of the next
 71679 command.

71680 The format for a pipeline is:

71681 [!] *command1* [| *command2* . . .]

71682 The standard output of *command1* shall be connected to the standard input of *command2*. The
 71683 standard input, standard output, or both of a command shall be considered to be assigned by
 71684 the pipeline before any redirection specified by redirection operators that are part of the
 71685 command (see Section 2.7, on page 2259).

71686 If the pipeline is not in the background (see Section 2.9.3.1, on page 2266), the shell shall wait for
 71687 the last command specified in the pipeline to complete, and may also wait for all commands to
 71688 complete.

71689 Exit Status

71690 If the reserved word ! does not precede the pipeline, the exit status shall be the exit status of the
 71691 last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the
 71692 exit status of the last command. That is, if the last command returns zero, the exit status shall be
 71693 1; if the last command returns greater than zero, the exit status shall be zero.

71694 **2.9.3 Lists**

71695 An *AND-OR list* is a sequence of one or more pipelines separated by the operators "&&" and
71696 "||".

71697 A *list* is a sequence of one or more AND-OR lists separated by the operators ';' and '&' and
71698 optionally terminated by ';&', '&', or <newline>.

71699 The operators "&&" and "||" shall have equal precedence and shall be evaluated with left
71700 associativity. For example, both of the following commands write solely **bar** to standard output:

```
71701 false && echo foo || echo bar
71702 true || echo foo && echo bar
```

71703 A ';' or <newline> terminator shall cause the preceding AND-OR list to be executed
71704 sequentially; an '&' shall cause asynchronous execution of the preceding AND-OR list.

71705 The term "compound-list" is derived from the grammar in [Section 2.10](#) (on page 2271); it is
71706 equivalent to a sequence of *lists*, separated by <newline>s, that can be preceded or followed by
71707 an arbitrary number of <newline>s.

71708 **Examples**

71709 The following is an example that illustrates <newline>s in compound-lists:

```
71710 while
71711     # a couple of <newline>s
71712     # a list
71713     date && who || ls; cat file
71714     # a couple of <newline>s
71715     # another list
71716     wc file > output & true
71717 do
71718     # 2 lists
71719     ls
71720     cat file
71721 done
```

71722 **2.9.3.1 Asynchronous Lists**

71723 If a command is terminated by the control operator ampersand ('&'), the shell shall execute the
71724 command asynchronously in a subshell. This means that the shell shall not wait for the
71725 command to finish before executing the next command.

71726 The format for running a command in the background is:

```
71727 command1 & [command2 & ... ]
```

71728 The standard input for an asynchronous list, before any explicit redirections are performed, shall
71729 be considered to be assigned to a file that has the same properties as **/dev/null**. If it is an
71730 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall
71731 override this activity.

71732 When an element of an asynchronous list (the portion of the list ended by an ampersand, such as
71733 *command1*, above) is started by the shell, the process ID of the last command in the asynchronous
71734 list element shall become known in the current shell execution environment; see [Section 2.12](#) (on

71735 page 2277). This process ID shall remain known until:

- 71736 1. The command terminates and the application waits for the process ID.
- 71737 2. Another asynchronous list is invoked before "\$!" (corresponding to the previous
- 71738 asynchronous list) is expanded in the current execution environment.

71739 The implementation need not retain more than the {CHILD_MAX} most recent entries in its list

71740 of known process IDs in the current shell execution environment.

71741 **Exit Status**

71742 The exit status of an asynchronous list shall be zero.

71743 2.9.3.2 *Sequential Lists*

71744 Commands that are separated by a semicolon (';') shall be executed sequentially.

71745 The format for executing commands sequentially shall be:

71746 *command1* [; *command2*] . . .

71747 Each command shall be expanded and executed in the order specified.

71748 **Exit Status**

71749 The exit status of a sequential list shall be the exit status of the last command in the list.

71750 2.9.3.3 *AND Lists*

71751 The control operator "&&" denotes an AND list. The format shall be:

71752 *command1* [&& *command2*] . . .

71753 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,

71754 until a command has a non-zero exit status or there are no more commands left to execute. The

71755 commands are expanded only if they are executed.

71756 **Exit Status**

71757 The exit status of an AND list shall be the exit status of the last command that is executed in the

71758 list.

71759 2.9.3.4 *OR Lists*

71760 The control operator "||" denotes an OR List. The format shall be:

71761 *command1* [|| *command2*] . . .

71762 First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and

71763 so on, until a command has a zero exit status or there are no more commands left to execute.

71764 **Exit Status**

71765 The exit status of an OR list shall be the exit status of the last command that is executed in the

71766 list.

71767 2.9.4 Compound Commands

71768 The shell has several programming constructs that are “compound commands”, which provide
 71769 control flow for commands. Each of these compound commands has a reserved word or control
 71770 operator at the beginning, and a corresponding terminator reserved word or operator at the end.
 71771 In addition, each can be followed by redirections on the same line as the terminator. Each
 71772 redirection shall apply to all the commands within the compound command that do not
 71773 explicitly override that redirection.

71774 2.9.4.1 Grouping Commands

71775 The format for grouping commands is as follows:

71776 (*compound-list*) Execute *compound-list* in a subshell environment; see [Section 2.12](#) (on page
 71777 2277). Variable assignments and built-in commands that affect the
 71778 environment shall not remain in effect after the list finishes.

71779 { *compound-list*;} Execute *compound-list* in the current process environment. The semicolon
 71780 shown here is an example of a control operator delimiting the } reserved
 71781 word. Other delimiters are possible, as shown in [Section 2.10](#) (on page
 71782 2271); a <newline> is frequently used.

71783 Exit Status

71784 The exit status of a grouping command shall be the exit status of *compound-list*.

71785 2.9.4.2 The for Loop

71786 The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for**
 71787 loop requires that the reserved words **do** and **done** be used to delimit the sequence of
 71788 commands.

71789 The format for the **for** loop is as follows:

```
71790 for name [ in [word ... ] ]
71791 do
71792     compound-list
71793 done
```

71794 First, the list of words following **in** shall be expanded to generate a list of items. Then, the
 71795 variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no
 71796 items result from the expansion, the *compound-list* shall not be executed. Omitting:

71797 **in** *word*...

71798 shall be equivalent to:

71799 **in** "\$@"

71800 Exit Status

71801 The exit status of a **for** command shall be the exit status of the last command that executes. If
 71802 there are no items, the exit status shall be zero.

71803 2.9.4.3 Case Conditional Construct

71804 The conditional construct **case** shall execute the *compound-list* corresponding to the first one of
 71805 several *patterns* (see Section 2.13, on page 2278) that is matched by the string resulting from the
 71806 tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote
 71807 removal of the given word. The reserved word **in** shall denote the beginning of the patterns to
 71808 be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|'
 71809 symbol. The control operator ')' terminates a list of patterns corresponding to a given action.
 71810 The *compound-list* for each list of patterns, with the possible exception of the last, shall be
 71811 terminated with ";;". The **case** construct terminates with the reserved word **esac** (**case**
 71812 reversed).

71813 The format for the **case** construct is as follows:

```
71814 case word in
71815     [(]pattern1) compound-list;;
71816     [(]pattern[ | pattern] ... ) compound-list;;] ...
71817     [(]pattern[ | pattern] ... ) compound-list]
71818 esac
```

71819 The " ; " is optional for the last *compound-list*.

71820 In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-*
 71821 *list* shall be subjected to tilde expansion, parameter expansion, command substitution, and
 71822 arithmetic expansion, and the result of these expansions shall be compared against the
 71823 expansion of *word*, according to the rules described in Section 2.13 (on page 2278) (which also
 71824 describes the effect of quoting parts of the pattern). After the first match, no more patterns shall
 71825 be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of
 71826 multiple *patterns* that label a *compound-list* statement is unspecified.

71827 **Exit Status**

71828 The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be
 71829 the exit status of the last command executed in the *compound-list*.

71830 2.9.4.4 The if Conditional Construct

71831 The **if** command shall execute a *compound-list* and use its exit status to determine whether to
 71832 execute another *compound-list*.

71833 The format for the **if** construct is as follows:

```
71834 if compound-list
71835 then
71836     compound-list
71837 [elif compound-list
71838 then
71839     compound-list] ...
71840 [else
71841     compound-list]
71842 fi
```

71843 The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be
 71844 executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed,
 71845 in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command
 71846 shall complete. Otherwise, the **else** *compound-list* shall be executed.

71847 **Exit Status**

71848 The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that
71849 was executed, or zero, if none was executed.

71850 2.9.4.5 *The while Loop*

71851 The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has
71852 a zero exit status.

71853 The format of the **while** loop is as follows:

```
71854 while compound-list-1
71855 do
71856     compound-list-2
71857 done
```

71858 The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command
71859 shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

71860 **Exit Status**

71861 The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or
71862 zero if none was executed.

71863 2.9.4.6 *The until Loop*

71864 The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has
71865 a non-zero exit status.

71866 The format of the **until** loop is as follows:

```
71867 until compound-list-1
71868 do
71869     compound-list-2
71870 done
```

71871 The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command
71872 completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

71873 **Exit Status**

71874 The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or
71875 zero if none was executed.

71876 **2.9.5 Function Definition Command**

71877 A function is a user-defined name that is used as a simple command to call a compound
71878 command with new positional parameters. A function is defined with a “function definition
71879 command”.

71880 The format of a function definition command is as follows:

```
71881 fname ( ) compound-command [io-redirect ...]
```

71882 The function is named *fname*; the application shall ensure that it is a name (see XBD [Section](#)
71883 [3.229](#), on page 65). An implementation may allow other characters in a function name as an
71884 extension. The implementation shall maintain separate name spaces for functions and variables.

71885 The argument *compound-command* represents a compound command, as described in [Section](#)
71886 [2.9.4](#) (on page 2268).

71887 When the function is declared, none of the expansions in [Section 2.6](#) (on page 2253) shall be
 71888 performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as
 71889 normal each time the function is called. Similarly, the optional *io-redirect* redirections and any
 71890 variable assignments within *compound-command* shall be performed during the execution of the
 71891 function itself, not the function definition. See [Section 2.8.1](#) (on page 2262) for the consequences
 71892 of failures of these operations on interactive and non-interactive shells.

71893 When a function is executed, it shall have the syntax-error and variable-assignment properties
 71894 described for special built-in utilities in the enumerated list at the beginning of [Section 2.14](#) (on
 71895 page 2280).

71896 The *compound-command* shall be executed whenever the function name is specified as the name
 71897 of a simple command (see [Section 2.9.1.1](#), on page 2264). The operands to the command
 71898 temporarily shall become the positional parameters during the execution of the *compound-*
 71899 *command*; the special parameter '#' also shall be changed to reflect the number of operands.
 71900 The special parameter 0 shall be unchanged. When the function completes, the values of the
 71901 positional parameters and the special parameter '#' shall be restored to the values they had
 71902 before the function was executed. If the special built-in *return* is executed in the *compound-*
 71903 *command*, the function completes and execution shall resume with the next command after the
 71904 function call.

71905 **Exit Status**

71906 The exit status of a function definition shall be zero if the function was declared successfully;
 71907 otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit
 71908 status of the last command executed by the function.

71909 **2.10 Shell Grammar**

71910 The following grammar defines the Shell Command Language. This formal syntax shall take
 71911 precedence over the preceding text syntax description.

71912 **2.10.1 Shell Grammar Lexical Conventions**

71913 The input language to the shell must be first recognized at the character level. The resulting
 71914 tokens shall be classified by their immediate context according to the following rules (applied in
 71915 order). These rules shall be used to determine what a "token" is that is subject to parsing at the
 71916 token level. The rules for token recognition in [Section 2.3](#) (on page 2247) shall apply.

- 71917 1. A <newline> shall be returned as the token identifier **NEWLINE**.
- 71918 2. If the token is an operator, the token identifier for that operator shall result.
- 71919 3. If the string consists solely of digits and the delimiter character is one of '<' or '>', the
 71920 token identifier **IO_NUMBER** shall be returned.
- 71921 4. Otherwise, the token identifier **TOKEN** results.

71922 Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields
 71923 **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the
 71924 context. Some of the productions in the grammar below are annotated with a rule number from
 71925 the following list. When a **TOKEN** is seen where one of those annotated productions could be
 71926 used to reduce the symbol, the applicable rule shall be applied to convert the token identifier
 71927 type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction
 71928 shall then proceed based upon the token identifier type yielded by the rule applied. When more
 71929 than one rule applies, the highest numbered rule shall apply (which in turn may refer to another

71930 rule). (Note that except in rule 7, the presence of an ' = ' in the token has no effect.)

71931 The **WORD** tokens shall have the word expansion rules applied to them immediately before the
71932 associated command is executed, not at the time the command is parsed.

71933 2.10.2 Shell Grammar Rules

71934 1. [Command Name]

71935 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
71936 shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any
71937 state where only a reserved word could be the next correct token, proceed as above.

71938 **Note:** Because at this point quote marks are retained in the token, quoted strings cannot be
71939 recognized as reserved words. This rule also implies that reserved words are not
71940 recognized except in certain positions in the input, such as after a <newline> or
71941 semicolon; the grammar presumes that if the reserved word is intended, it is properly
71942 delimited by the user, and does not attempt to reflect that requirement directly. Also
71943 note that line joining is done before tokenization, as described in Section 2.2.1 (on page
71944 2246), so escaped <newline>s are already removed at this point.

71945 Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or
71946 applies globally.

71947 2. [Redirection to or from filename]

71948 The expansions specified in Section 2.7 (on page 2259) shall occur. As specified there,
71949 exactly one field can result (or the result is unspecified), and there are additional
71950 requirements on pathname expansion.

71951 3. [Redirection from here-document]

71952 Quote removal shall be applied to the word to determine the delimiter that is used to find
71953 the end of the here-document that begins after the next <newline>.

71954 4. [Case statement termination]

71955 When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall
71956 result. Otherwise, the token **WORD** shall be returned.

71957 5. [NAME in for]

71958 When the **TOKEN** meets the requirements for a name (see XBD Section 3.229, on page
71959 65), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be
71960 returned.

71961 6. [Third word of for and case]

71962 a. [case only]

71963 When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall
71964 result. Otherwise, the token **WORD** shall be returned.

71965 b. [for only]

71966 When the **TOKEN** is exactly the reserved word **in** or **do**, the token identifier for **in**
71967 or **do** shall result, respectively. Otherwise, the token **WORD** shall be returned.

71968 (For a. and b.: As indicated in the grammar, a *linebreak* precedes the tokens **in** and **do**. If
71969 <newline>s are present at the indicated location, it is the token after them that is treated
71970 in this fashion.)

- 71971 7. [Assignment preceding command name]
- 71972 a. [When the first word]
- 71973 If the **TOKEN** does not contain the character '=' , rule 1 is applied. Otherwise, 7b
- 71974 shall be applied.
- 71975 b. [Not the first word]
- 71976 If the **TOKEN** contains the equal sign character:
- 71977 — If it begins with '=' , the token **WORD** shall be returned.
- 71978 — If all the characters preceding '=' form a valid name (see XBD Section 3.229, |
- 71979 on page 65), the token **ASSIGNMENT_WORD** shall be returned. (Quoted
- 71980 characters cannot participate in forming a valid name.)
- 71981 — Otherwise, it is unspecified whether it is **ASSIGNMENT_WORD** or **WORD**
- 71982 that is returned.

71983 Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 2263).

- 71984 8. [**NAME** in function]
- 71985 When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word
- 71986 shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token
- 71987 identifier **NAME** shall result. Otherwise, rule 7 applies.

- 71988 9. [Body of function]
- 71989 Word expansion and assignment shall never occur, even when required by the rules
- 71990 above, when this rule is being parsed. Each **TOKEN** that might either be expanded or
- 71991 have assignment applied to it shall instead be returned as a single **WORD** consisting only
- 71992 of characters that are exactly the token described in Section 2.3 (on page 2247).

```

71993 /* -----
71994 The grammar symbols
71995 ----- */

71996 %token WORD
71997 %token ASSIGNMENT_WORD
71998 %token NAME
71999 %token NEWLINE
72000 %token IO_NUMBER

72001 /* The following are the operators mentioned above. */
72002 %token AND_IF OR_IF DSEMI
72003 /* '&&' '|;' ';' */

72004 %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
72005 /* '<<' '>>' '<&' '>&' '<>' '<<-' */

72006 %token CLOBBER
72007 /* '>|' */

72008 /* The following are the reserved words. */
72009 %token If Then Else Elif Fi Do Done
72010 /* 'if' 'then' 'else' 'elif' 'fi' 'do' 'done' */

72011 %token Case Esac While Until For
72012 /* 'case' 'esac' 'while' 'until' 'for' */

```

```

72013      /* These are reserved words, not operator tokens, and are
72014         recognized when reserved words are recognized. */

72015      %token Lbrace      Rbrace      Bang
72016      /*      '{'        '}'        '!'    */

72017      %token In
72018      /*      'in'      */

72019      /* -----
72020         The Grammar
72021      ----- */

72022      %start complete_command
72023      %%
72024      complete_command : list separator
72025                       | list
72026                       ;
72027      list             : list separator_op and_or
72028                       |                               and_or
72029                       ;
72030      and_or           : pipeline
72031                       | and_or AND_IF linebreak pipeline
72032                       | and_or OR_IF linebreak pipeline
72033                       ;
72034      pipeline         : pipe_sequence
72035                       | Bang pipe_sequence
72036                       ;
72037      pipe_sequence    :                               command
72038                       | pipe_sequence '|' linebreak command
72039                       ;
72040      command          : simple_command
72041                       | compound_command
72042                       | compound_command redirect_list
72043                       | function_definition
72044                       ;
72045      compound_command : brace_group
72046                       | subshell
72047                       | for_clause
72048                       | case_clause
72049                       | if_clause
72050                       | while_clause
72051                       | until_clause
72052                       ;
72053      subshell         : '(' compound_list ')'
72054                       ;
72055      compound_list    : term
72056                       | newline_list term
72057                       | term separator
72058                       | newline_list term separator
72059                       ;
72060      term             : term separator and_or
72061                       |                               and_or
72062                       ;
72063      for_clause       : For name linebreak                               do_group

```

```

72064         | For name linebreak in          sequential_sep do_group
72065         | For name linebreak in wordlist sequential_sep do_group
72066         ;
72067     name      : NAME                      /* Apply rule 5 */
72068         ;
72069     in        : In                        /* Apply rule 6 */
72070         ;
72071     wordlist   : wordlist WORD
72072         |      WORD
72073         ;
72074     case_clause : Case WORD linebreak in linebreak case_list  Esac
72075         | Case WORD linebreak in linebreak case_list_ns Esac
72076         | Case WORD linebreak in linebreak                      Esac
72077         ;
72078     case_list_ns : case_list case_item_ns
72079         |      case_item_ns
72080         ;
72081     case_list   : case_list case_item
72082         |      case_item
72083         ;
72084     case_item_ns :      pattern ')' linebreak
72085         |      pattern ')' compound_list linebreak
72086         | '(' pattern ')' linebreak
72087         | '(' pattern ')' compound_list linebreak
72088         ;
72089     case_item   :      pattern ')' linebreak DSEMI linebreak
72090         |      pattern ')' compound_list DSEMI linebreak
72091         | '(' pattern ')' linebreak DSEMI linebreak
72092         | '(' pattern ')' compound_list DSEMI linebreak
72093         ;
72094     pattern     :      WORD                /* Apply rule 4 */
72095         | pattern '|' WORD                /* Do not apply rule 4 */
72096         ;
72097     if_clause   : If compound_list Then compound_list else_part Fi
72098         | If compound_list Then compound_list Fi
72099         ;
72100     else_part   : Elif compound_list Then else_part
72101         | Else compound_list
72102         ;
72103     while_clause : While compound_list do_group
72104         ;
72105     until_clause : Until compound_list do_group
72106         ;
72107     function_definition : fname '(' ')' linebreak function_body
72108         ;
72109     function_body : compound_command          /* Apply rule 9 */
72110         | compound_command redirect_list /* Apply rule 9 */
72111         ;
72112     fname       : NAME                      /* Apply rule 8 */
72113         ;
72114     brace_group : Lbrace compound_list Rbrace
72115         ;
72116     do_group    : Do compound_list Done          /* Apply rule 6 */

```

```

72117         ;
72118     simple_command : cmd_prefix cmd_word cmd_suffix
72119                   | cmd_prefix cmd_word
72120                   | cmd_prefix
72121                   | cmd_name cmd_suffix
72122                   | cmd_name
72123         ;
72124     cmd_name       : WORD                               /* Apply rule 7a */
72125         ;
72126     cmd_word       : WORD                               /* Apply rule 7b */
72127         ;
72128     cmd_prefix     : io_redirect
72129                   | cmd_prefix io_redirect
72130                   | ASSIGNMENT_WORD
72131                   | cmd_prefix ASSIGNMENT_WORD
72132         ;
72133     cmd_suffix     : io_redirect
72134                   | cmd_suffix io_redirect
72135                   | WORD
72136                   | cmd_suffix WORD
72137         ;
72138     redirect_list  : io_redirect
72139                   | redirect_list io_redirect
72140         ;
72141     io_redirect    : io_file
72142                   | IO_NUMBER io_file
72143                   | io_here
72144                   | IO_NUMBER io_here
72145         ;
72146     io_file        : '<' filename
72147                   | LESSAND filename
72148                   | '>' filename
72149                   | GREATAND filename
72150                   | DGREAT filename
72151                   | LESSGREAT filename
72152                   | CLOBBER filename
72153         ;
72154     filename       : WORD                               /* Apply rule 2 */
72155         ;
72156     io_here        : DLESS here_end
72157                   | DLESSDASH here_end
72158         ;
72159     here_end       : WORD                               /* Apply rule 3 */
72160         ;
72161     newline_list   : NEWLINE
72162                   | newline_list NEWLINE
72163         ;
72164     linebreak      : newline_list
72165                   | /* empty */
72166         ;
72167     separator_op   : '&'
72168                   | ';'
72169         ;

```

```

72170     separator      : separator_op linebreak
72171                   | newline_list
72172                   ;
72173     sequential_sep  : ';' linebreak
72174                   | newline_list
72175                   ;

```

2.11 Signals and Error Handling

72176

72177

72178

72179

72180

72181

When a command is in an asynchronous list, it shall inherit from the shell a signal action of ignored (SIG_IGN) for the SIGQUIT and SIGINT signals, and may inherit a signal mask in which SIGQUIT and SIGINT are blocked. Otherwise, the signal actions and signal mask inherited by the command shall be the same as those inherited by the shell from its parent unless a signal action is modified by the *trap* special built-in (see *trap*)

72182

72183

72184

72185

72186

72187

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal shall not be executed until after the foreground command has completed. When the shell is waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit status >128, immediately after which the trap associated with that signal shall be taken.

72188

72189

If multiple signals are pending for the shell for which there are associated trap actions, the order of execution of trap actions is unspecified.

2.12 Shell Execution Environment

72190

72191

72192

72193

72194

72195

72196

72197

72198

72199

72200

72201

72202

72203

A shell execution environment consists of the following:

- Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- Working directory as set by *cd*
- File creation mask set by *umask*
- Current traps set by *trap*
- Shell parameters that are set by variable assignment (see the *set* special built-in) or from the System Interfaces volume of POSIX.1-200x environment inherited by the shell when it begins (see the *export* special built-in)
- Shell functions; see [Section 2.9.5](#) (on page 2270)
- Options turned on at invocation or by *set*
- Process IDs of the last commands in asynchronous lists known to this shell environment; see [Section 2.9.3.1](#) (on page 2266)
- Shell aliases; see [Section 2.3.1](#) (on page 2248)

72204

72205

72206

Utilities other than the special built-ins (see [Section 2.14](#), on page 2280) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

72207

72208

- Open files inherited on invocation of the shell, open files controlled by the *exec* special built-in plus any modifications, and additions specified by any redirections to the utility

- 72209 • Current working directory
- 72210 • File creation mask
- 72211 • If the utility is a shell script, traps caught by the shell shall be set to the default values and
- 72212 traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell
- 72213 script, the trap actions (default or ignore) shall be mapped into the appropriate signal
- 72214 handling actions for the utility
- 72215 • Variables with the *export* attribute, along with those explicitly exported for the duration of
- 72216 the command, shall be passed to the utility environment variables

72217 The environment of the shell process shall not be changed by the utility unless explicitly
72218 specified by the utility description (for example, *cd* and *umask*).

72219 A subshell environment shall be created as a duplicate of the shell environment, except that
72220 signal traps set by that shell environment shall be set to the default values. Changes made to the
72221 subshell environment shall not affect the shell environment. Command substitution, commands
72222 that are grouped with parentheses, and asynchronous lists shall be executed in a subshell
72223 environment. Additionally, each command of a multi-command pipeline is in a subshell
72224 environment; as an extension, however, any or all commands in a pipeline may be executed in
72225 the current environment. All other commands shall be executed in the current shell
72226 environment.

72227 2.13 Pattern Matching Notation

72228 The pattern matching notation described in this section is used to specify patterns for matching
72229 strings in the shell. Historically, pattern matching notation is related to, but slightly different
72230 from, the regular expression notation described in XBD [Chapter 9](#) (on page 167). For this reason,
72231 the description of the rules for this pattern matching notation are based on the description of
72232 regular expression notation, modified to include backslash escape processing.

72233 2.13.1 Patterns Matching a Single Character

72234 The following patterns matching a single character shall match a single character: ordinary
72235 characters, special pattern characters, and pattern bracket expressions. The pattern bracket
72236 expression also shall match a single collating element. A backslash character shall escape the
72237 following character. The escaping backslash shall be discarded.

72238 An ordinary character is a pattern that shall match itself. It can be any character in the supported
72239 character set except for NUL, those special shell characters in [Section 2.2](#) (on page 2246) that
72240 require quoting, and the following three special pattern characters. Matching shall be based on
72241 the bit pattern used for encoding the character, not on the graphic representation of the
72242 character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall
72243 match the character itself. The shell special characters always require quoting.

72244 When unquoted and outside a bracket expression, the following three characters shall have
72245 special meaning in the specification of patterns:

- 72246 ? A question-mark is a pattern that shall match any character.
- 72247 * An asterisk is a pattern that shall match multiple characters, as described in [Section 2.13.2](#)
72248 (on page 2279).
- 72249 [The open bracket shall introduce a pattern bracket expression.

72250 The description of basic regular expression bracket expressions in XBD [Section 9.3.5](#) (on page
72251 170) shall also apply to the pattern bracket expression, except that the exclamation mark

72252 character (' ! ') shall replace the circumflex character (' ^ ') in its role in a “non-matching list” in
 72253 the regular expression notation. A bracket expression starting with an unquoted circumflex
 72254 character produces unspecified results.

72255 When pattern matching is used where shell quote removal is not performed (such as in the
 72256 argument to the *find -name* primary when *find* is being called using one of the *exec* functions as
 72257 defined in the System Interfaces volume of POSIX.1-200x, or in the *pattern* argument to the
 72258 *fnmatch()* function), special characters can be escaped to remove their special meaning by
 72259 preceding them with a backslash character. This escaping backslash is discarded. The sequence
 72260 " \ " represents one literal backslash. All of the requirements and effects of quoting on ordinary,
 72261 shell special, and special pattern characters shall apply to escaping in this context.

72262 2.13.2 Patterns Matching Multiple Characters

72263 The following rules are used to construct patterns matching multiple characters from patterns
 72264 matching a single character:

- 72265 1. The asterisk (' * ') is a pattern that shall match any string, including the null string.
- 72266 2. The concatenation of patterns matching a single character is a valid pattern that shall
 72267 match the concatenation of the single characters or collating elements matched by each of
 72268 the concatenated patterns.
- 72269 3. The concatenation of one or more patterns matching a single character with one or more
 72270 asterisks is a valid pattern. In such patterns, each asterisk shall match a string of zero or
 72271 more characters, matching the greatest possible number of characters that still allows the
 72272 remainder of the pattern to match the string.

72273 2.13.3 Patterns Used for Filename Expansion

72274 The rules described so far in [Section 2.13.1](#) (on page 2278) and [Section 2.13.2](#) are qualified by the
 72275 following rules that apply when pattern matching notation is used for filename expansion:

- 72276 1. The slash character in a pathname shall be explicitly matched by using one or more
 72277 slashes in the pattern; it shall neither be matched by the asterisk or question-mark special
 72278 characters nor by a bracket expression. Slashes in the pattern shall be identified before
 72279 bracket expressions; thus, a slash cannot be included in a pattern bracket expression used
 72280 for filename expansion. If a slash character is found following an unescaped open square
 72281 bracket character before a corresponding closing square bracket is found, the open
 72282 bracket shall be treated as an ordinary character. For example, the pattern "a[b/c]d"
 72283 does not match such pathnames as **abd** or **a/d**. It only matches a pathname of literally
 72284 **a[b/c]d**.
- 72285 2. If a filename begins with a period (' . '), the period shall be explicitly matched by using a
 72286 period as the first character of the pattern or immediately following a slash character. The
 72287 leading period shall not be matched by:
 - 72288 • The asterisk or question-mark special characters
 - 72289 • A bracket expression containing a non-matching list, such as "[!a]", a range
 72290 expression, such as "[%-0]", or a character class expression, such as
 72291 "[[:punct:]]"

72292 It is unspecified whether an explicit period in a bracket expression matching list, such as
 72293 "[.abc]", can match a leading period in a filename.

72294 3. Specified patterns shall be matched against existing filenames and pathnames, as
 72295 appropriate. Each component that contains a pattern character shall require read
 72296 permission in the directory containing that component. Any component, except the last,
 72297 that does not contain a pattern character shall require search permission. For example,
 72298 given the pattern:

72299 `/foo/bar/x*/bam`

72300 search permission is needed for directories `/` and `foo`, search and read permissions are
 72301 needed for directory `bar`, and search permission is needed for each `x*` directory. If the
 72302 pattern matches any existing filenames or pathnames, the pattern shall be replaced with
 72303 those filenames and pathnames, sorted according to the collating sequence in effect in the
 72304 current locale. If the pattern contains an invalid bracket expression or does not match any
 72305 existing filenames or pathnames, the pattern string shall be left unchanged.

72306 2.14 Special Built-In Utilities

72307 The following “special built-in” utilities shall be supported in the shell command language. The
 72308 output of each command, if any, shall be written to standard output, subject to the normal
 72309 redirection and piping possible with all commands.

72310 The term “built-in” implies that the shell can execute the utility directly and does not need to
 72311 search for it. An implementation may choose to make any utility a built-in; however, the special
 72312 built-in utilities described here differ from regular built-in utilities in two respects:

- 72313 1. A syntax error in a special built-in utility may cause a shell executing that utility to abort,
 72314 while a syntax error in a regular built-in utility shall not cause a shell executing that
 72315 utility to abort. (See [Section 2.8.1](#) (on page 2262) for the consequences of errors on
 72316 interactive and non-interactive shells.) If a special built-in utility encountering a syntax
 72317 error does not abort the shell, its exit value shall be non-zero.
- 72318 2. Variable assignments specified with special built-in utilities remain in effect after the
 72319 built-in completes; this shall not be the case with a regular built-in or other utility.

72320 The special built-in utilities in this section need not be provided in a manner accessible via the
 72321 `exec` family of functions defined in the System Interfaces volume of POSIX.1-200x.

72322 Some of the special built-ins are described as conforming to XBD [Section 12.2](#) (on page 201). For
 72323 those that are not, the requirement in [Section 1.4](#) (on page 2235) that “--” be recognized as a
 72324 first argument to be discarded does not apply and a conforming application shall not use that
 72325 argument.

72326 **NAME**

72327 `break` — exit from `for`, `while`, or `until` loop

72328 **SYNOPSIS**

72329 `break` [*n*]

72330 **DESCRIPTION**

72331 The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from

72332 the *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater

72333 than or equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of

72334 enclosing loops, the outermost enclosing loop shall be exited. Execution shall continue with the

72335 command immediately following the loop.

72336 **OPTIONS**

72337 None.

72338 **OPERANDS**

72339 See the DESCRIPTION.

72340 **STDIN**

72341 Not used.

72342 **INPUT FILES**

72343 None.

72344 **ENVIRONMENT VARIABLES**

72345 None.

72346 **ASYNCHRONOUS EVENTS**

72347 Default.

72348 **STDOUT**

72349 Not used.

72350 **STDERR**

72351 The standard error shall be used only for diagnostic messages.

72352 **OUTPUT FILES**

72353 None.

72354 **EXTENDED DESCRIPTION**

72355 None.

72356 **EXIT STATUS**

72357 0 Successful completion.

72358 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

72359 **CONSEQUENCES OF ERRORS**

72360 Default.

break72361 **APPLICATION USAGE**

72362 None.

72363 **EXAMPLES**

```
72364     for i in *
72365     do
72366         if test -d "$i"
72367         then break
72368         fi
72369     done
```

72370 **RATIONALE**

72371 In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer
 72372 to a label associated with the appropriate loop as a preferable alternative to the *n* method.
 72373 However, this volume of POSIX.1-200x does reserve the name space of command names ending
 72374 with a colon. It is anticipated that a future implementation could take advantage of this and
 72375 provide something like:

```
72376 outofloop: for i in a b c d e
72377 do
72378     for j in 0 1 2 3 4 5 6 7 8 9
72379     do
72380         if test -r "${i}${j}"
72381         then break outofloop
72382         fi
72383     done
72384 done
```

72385 and that this might be standardized after implementation experience is achieved.

72386 **FUTURE DIRECTIONS**

72387 None.

72388 **SEE ALSO**72389 [Section 2.14](#) (on page 2280)72390 **CHANGE HISTORY**72391 **Issue 6**

72392 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 72393 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 72394 behavior is intended.

72395 **NAME**

72396 colon — null utility

72397 **SYNOPSIS**72398 : [*argument...*]72399 **DESCRIPTION**72400 This utility shall only expand command *arguments*. It is used when a command is needed, as in
72401 the **then** condition of an **if** command, but nothing is to be done by the command.72402 **OPTIONS**

72403 None.

72404 **OPERANDS**

72405 See the DESCRIPTION.

72406 **STDIN**

72407 Not used.

72408 **INPUT FILES**

72409 None.

72410 **ENVIRONMENT VARIABLES**

72411 None.

72412 **ASYNCHRONOUS EVENTS**

72413 Default.

72414 **STDOUT**

72415 Not used.

72416 **STDERR**

72417 The standard error shall be used only for diagnostic messages.

72418 **OUTPUT FILES**

72419 None.

72420 **EXTENDED DESCRIPTION**

72421 None.

72422 **EXIT STATUS**

72423 Zero.

72424 **CONSEQUENCES OF ERRORS**

72425 Default.

72426 **APPLICATION USAGE**

72427 None.

72428 **EXAMPLES**

```

72429 : ${X=abc}
72430 if     false
72431 then  :
72432 else  echo $X
72433 fi
72434 abc

```

72435 As with any of the special built-ins, the null utility can also have variable assignments and
 72436 redirections associated with it, such as:

```

72437 x=y : > z

```

colon*Shell Command Language*

72438 which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates
72439 or truncates file *z*.

RATIONALE

72440
72441 None.

FUTURE DIRECTIONS

72442
72443 None.

SEE ALSO

72444
72445 [Section 2.14](#) (on page 2280)

CHANGE HISTORY**Issue 6**

72447
72448 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
72449 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
72450 behavior is intended.

Issue 7

72451
72452 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

DRAFT

- 72453 **NAME**
- 72454 continue — continue for, while, or until loop
- 72455 **SYNOPSIS**
- 72456 continue [*n*]
- 72457 **DESCRIPTION**
- 72458 The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to
- 72459 the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a
- 72460 **while** or **until** loop or performing the next assignment of a **for** loop, and re-executing the loop if
- 72461 appropriate.
- 72462 The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to
- 72463 *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be
- 72464 used.
- 72465 **OPTIONS**
- 72466 None.
- 72467 **OPERANDS**
- 72468 See the DESCRIPTION.
- 72469 **STDIN**
- 72470 Not used.
- 72471 **INPUT FILES**
- 72472 None.
- 72473 **ENVIRONMENT VARIABLES**
- 72474 None.
- 72475 **ASYNCHRONOUS EVENTS**
- 72476 Default.
- 72477 **STDOUT**
- 72478 Not used.
- 72479 **STDERR**
- 72480 The standard error shall be used only for diagnostic messages.
- 72481 **OUTPUT FILES**
- 72482 None.
- 72483 **EXTENDED DESCRIPTION**
- 72484 None.
- 72485 **EXIT STATUS**
- 72486 0 Successful completion.
- 72487 >0 The *n* value was not an unsigned decimal integer greater than or equal to 1.
- 72488 **CONSEQUENCES OF ERRORS**
- 72489 Default.

72490 **APPLICATION USAGE**

72491 None.

72492 **EXAMPLES**

```
72493     for i in *
72494     do
72495         if test -d "$i"
72496         then continue
72497         fi
72498         printf '"$s" is not a directory.\n' "$i"
72499     done
```

72500 **RATIONALE**

72501 None.

72502 **FUTURE DIRECTIONS**

72503 None.

72504 **SEE ALSO**72505 [Section 2.14](#) (on page 2280)72506 **CHANGE HISTORY**72507 **Issue 6**

72508 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
72509 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
72510 behavior is intended.

72511	NAME
72512	dot — execute commands in the current environment
72513	SYNOPSIS
72514	. <i>file</i>
72515	DESCRIPTION
72516	The shell shall execute commands from the <i>file</i> in the current environment.
72517	If <i>file</i> does not contain a slash, the shell shall use the search path specified by <i>PATH</i> to find the
72518	directory containing <i>file</i> . Unlike normal command search, however, the file searched for by the
72519	<i>dot</i> utility need not be executable. If no readable file is found, a non-interactive shell shall abort;
72520	an interactive shell shall write a diagnostic message to standard error, but this condition shall
72521	not be considered a syntax error.
72522	OPTIONS
72523	None.
72524	OPERANDS
72525	See the DESCRIPTION.
72526	STDIN
72527	Not used.
72528	INPUT FILES
72529	See the DESCRIPTION.
72530	ENVIRONMENT VARIABLES
72531	See the DESCRIPTION.
72532	ASYNCHRONOUS EVENTS
72533	Default.
72534	STDOUT
72535	Not used.
72536	STDERR
72537	The standard error shall be used only for diagnostic messages.
72538	OUTPUT FILES
72539	None.
72540	EXTENDED DESCRIPTION
72541	None.
72542	EXIT STATUS
72543	Returns the value of the last command executed, or a zero exit status if no command is executed.
72544	CONSEQUENCES OF ERRORS
72545	Default.

72546 **APPLICATION USAGE**

72547 None.

72548 **EXAMPLES**72549 `cat foobar`72550 `foo=hello bar=world`72551 `. foobar`72552 `echo $foo $bar`72553 `hello world`72554 **RATIONALE**

72555 Some older implementations searched the current directory for the *file*, even if the value of *PATH*
72556 disallowed it. This behavior was omitted from this volume of POSIX.1-200x due to concerns
72557 about introducing the susceptibility to trojan horses that the user might be trying to avoid by
72558 leaving **dot** out of *PATH*.

72559 The KornShell version of *dot* takes optional arguments that are set to the positional parameters.
72560 This is a valid extension that allows a *dot* script to behave identically to a function.

72561 **FUTURE DIRECTIONS**

72562 None.

72563 **SEE ALSO**72564 [Section 2.14](#) (on page 2280)72565 **CHANGE HISTORY**72566 **Issue 6**

72567 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
72568 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
72569 behavior is intended.

72570 **NAME**

72571 eval — construct command by concatenating arguments

72572 **SYNOPSIS**

72573 eval [*argument...*]

72574 **DESCRIPTION**

72575 The *eval* utility shall construct a command by concatenating *arguments* together, separating each

72576 with a <space>. The constructed command shall be read and executed by the shell.

72577 **OPTIONS**

72578 None.

72579 **OPERANDS**

72580 See the DESCRIPTION.

72581 **STDIN**

72582 Not used.

72583 **INPUT FILES**

72584 None.

72585 **ENVIRONMENT VARIABLES**

72586 None.

72587 **ASYNCHRONOUS EVENTS**

72588 Default.

72589 **STDOUT**

72590 Not used.

72591 **STDERR**

72592 The standard error shall be used only for diagnostic messages.

72593 **OUTPUT FILES**

72594 None.

72595 **EXTENDED DESCRIPTION**

72596 None.

72597 **EXIT STATUS**

72598 If there are no *arguments*, or only null arguments, *eval* shall return a zero exit status; otherwise, it

72599 shall return the exit status of the command defined by the string of concatenated *arguments*

72600 separated by <space>s.

72601 **CONSEQUENCES OF ERRORS**

72602 Default.

72603 **APPLICATION USAGE**

72604 None.

72605 **EXAMPLES**

72606 foo=10 x=foo

72607 y=' '\$x

72608 echo \$y

72609 **\$foo**

72610 eval y=' '\$x

72611 echo \$y

72612 **10**

72613

RATIONALE

72614

None.

72615

FUTURE DIRECTIONS

72616

None.

72617

SEE ALSO

72618

[Section 2.14](#) (on page 2280)

72619

CHANGE HISTORY

72620

Issue 6

72621

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in behavior is intended.

72622

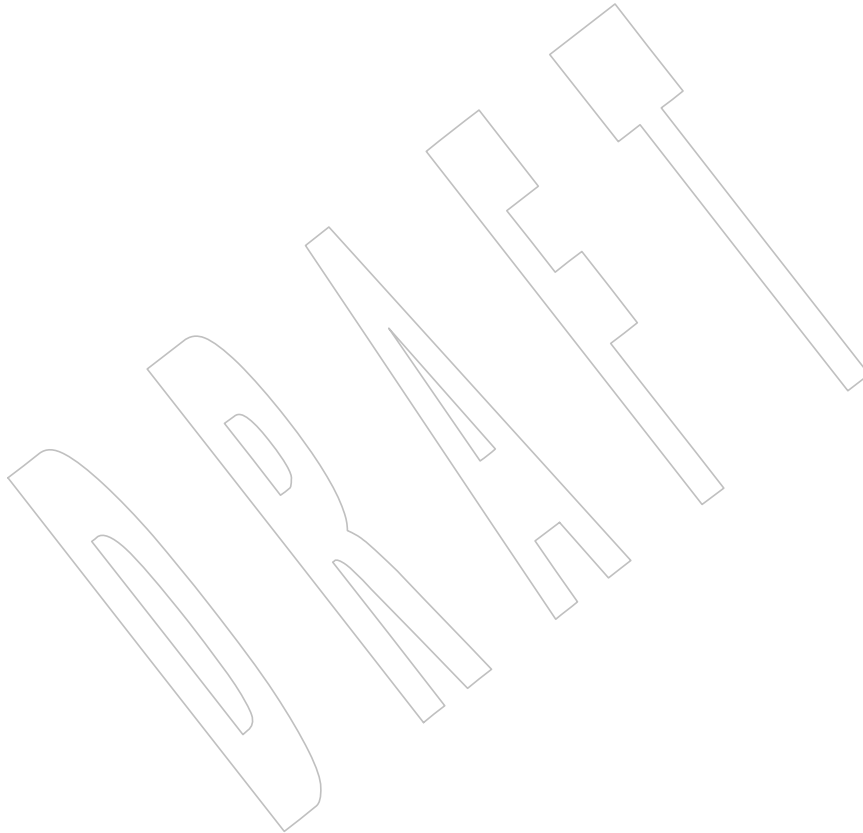
72623

72624

Issue 7

72625

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



72626 **NAME**
 72627 `exec` — execute commands and open, close, or copy file descriptors

72628 **SYNOPSIS**
 72629 `exec [command [argument...]]`

72630 **DESCRIPTION**
 72631 The `exec` utility shall open, close, and/or copy file descriptors as specified by any redirections as
 72632 part of the command.

72633 If `exec` is specified without `command` or `arguments`, and any file descriptors with numbers greater
 72634 than 2 are opened with associated redirection statements, it is unspecified whether those file
 72635 descriptors remain open when the shell invokes another utility. Scripts concerned that child
 72636 shells could misuse open file descriptors can always close them explicitly, as shown in one of the
 72637 following examples.

72638 If `exec` is specified with `command`, it shall replace the shell with `command` without creating a new
 72639 process. If `arguments` are specified, they shall be arguments to `command`. Redirection affects the
 72640 current shell execution environment.

72641 **OPTIONS**
 72642 None.

72643 **OPERANDS**
 72644 See the DESCRIPTION.

72645 **STDIN**
 72646 Not used.

72647 **INPUT FILES**
 72648 None.

72649 **ENVIRONMENT VARIABLES**
 72650 None.

72651 **ASYNCHRONOUS EVENTS**
 72652 Default.

72653 **STDOUT**
 72654 Not used.

72655 **STDERR**
 72656 The standard error shall be used only for diagnostic messages.

72657 **OUTPUT FILES**
 72658 None.

72659 **EXTENDED DESCRIPTION**
 72660 None.

72661 **EXIT STATUS**
 72662 If `command` is specified, `exec` shall not return to the shell; rather, the exit status of the process shall
 72663 be the exit status of the program implementing `command`, which overlaid the shell. If `command` is
 72664 not found, the exit status shall be 127. If `command` is found, but it is not an executable utility, the
 72665 exit status shall be 126. If a redirection error occurs (see [Section 2.8.1](#), on page 2262), the shell
 72666 shall exit with a value in the range 1–125. Otherwise, `exec` shall return a zero exit status.

72667 **CONSEQUENCES OF ERRORS**

72668 Default.

72669 **APPLICATION USAGE**

72670 None.

72671 **EXAMPLES**72672 Open *readfile* as file descriptor 3 for reading:72673 `exec 3< readfile`72674 Open *writefile* as file descriptor 4 for writing:72675 `exec 4> writefile`

72676 Make file descriptor 5 a copy of file descriptor 0:

72677 `exec 5<&0`

72678 Close file descriptor 3:

72679 `exec 3<&-`72680 Cat the file **maggie** by replacing the current shell with the *cat* utility:72681 `exec cat maggie`72682 **RATIONALE**

72683 Most historical implementations were not conformant in that:

72684 `foo=bar exec cmd`72685 did not pass **foo** to **cmd**.72686 **FUTURE DIRECTIONS**

72687 None.

72688 **SEE ALSO**72689 [Section 2.14](#) (on page 2280)72690 **CHANGE HISTORY**72691 **Issue 6**72692 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
72693 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
72694 behavior is intended.72695 **Issue 7**

72696 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

72697 **NAME**

72698 exit — cause the shell to exit

72699 **SYNOPSIS**72700 exit [*n*]72701 **DESCRIPTION**

72702 The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal
 72703 integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is
 72704 undefined.

72705 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is
 72706 invoked in that *trap* itself, in which case the shell shall exit immediately.

72707 **OPTIONS**

72708 None.

72709 **OPERANDS**

72710 See the DESCRIPTION.

72711 **STDIN**

72712 Not used.

72713 **INPUT FILES**

72714 None.

72715 **ENVIRONMENT VARIABLES**

72716 None.

72717 **ASYNCHRONOUS EVENTS**

72718 Default.

72719 **STDOUT**

72720 Not used.

72721 **STDERR**

72722 The standard error shall be used only for diagnostic messages.

72723 **OUTPUT FILES**

72724 None.

72725 **EXTENDED DESCRIPTION**

72726 None.

72727 **EXIT STATUS**

72728 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last
 72729 command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,
 72730 the last command is considered to be the command that executed immediately preceding the
 72731 *trap* action.

72732 **CONSEQUENCES OF ERRORS**

72733 Default.

72734

APPLICATION USAGE

72735

None.

72736

EXAMPLES

72737

Exit with a *true* value:

72738

`exit 0`

72739

Exit with a *false* value:

72740

`exit 1`

72741

RATIONALE

72742

As explained in other sections, certain exit status values have been reserved for special uses and should be used by applications only for those purposes:

72743

72744

126 A file to be executed was found, but it was not an executable utility.

72745

127 A utility to be executed was not found.

72746

>128 A command was interrupted by a signal.

72747

FUTURE DIRECTIONS

72748

None.

72749

SEE ALSO

72750

[Section 2.14](#) (on page 2280)

72751

CHANGE HISTORY

72752

Issue 6

72753

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in behavior is intended.

72754

72755

72756 **NAME**
 72757 export — set the export attribute for variables

72758 **SYNOPSIS**
 72759 export name[=word] . . .

72760 export -p

72761 **DESCRIPTION**

72762 The shell shall give the *export* attribute to the variables corresponding to the specified *names*,
 72763 which shall cause them to be in the environment of subsequently executed commands. If the
 72764 name of a variable is followed by =*word*, then the value of that variable shall be set to *word*.

72765 The *export* special built-in shall support XBD [Section 12.2](#) (on page 201).

72766 When **-p** is specified, *export* shall write to the standard output the names and values of all
 72767 exported variables, in the following format:

72768 "export %s=%s\n", <name>, <value>

72769 if *name* is set, and:

72770 "export %s\n", <name>

72771 if *name* is unset.

72772 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 72773 reinput to the shell as commands that achieve the same exporting results, except:

- 72774 1. Read-only variables with values cannot be reset.
 72775 2. Variables that were unset at the time they were output need not be reset to the unset state
 72776 if a value is assigned to the variable between the time the state was saved and the time at
 72777 which the saved output is reinput to the shell.

72778 When no arguments are given, the results are unspecified.

72779 **OPTIONS**
 72780 See the DESCRIPTION.

72781 **OPERANDS**
 72782 See the DESCRIPTION.

72783 **STDIN**
 72784 Not used.

72785 **INPUT FILES**
 72786 None.

72787 **ENVIRONMENT VARIABLES**
 72788 None.

72789 **ASYNCHRONOUS EVENTS**
 72790 Default.

72791 **STDOUT**
 72792 See the DESCRIPTION.

export

72793 **STDERR**
 72794 The standard error shall be used only for diagnostic messages.

72795 **OUTPUT FILES**
 72796 None.

72797 **EXTENDED DESCRIPTION**
 72798 None.

72799 **EXIT STATUS**
 72800 Zero.

72801 **CONSEQUENCES OF ERRORS**
 72802 Default.

72803 **APPLICATION USAGE**
 72804 None.

72805 **EXAMPLES**
 72806 Export *PWD* and *HOME* variables:
 72807 `export PWD HOME`
 72808 Set and export the *PATH* variable:
 72809 `export PATH=/local/bin:$PATH`
 72810 Save and restore all exported variables:
 72811 `export -p > temp-file`
 72812 `unset a lot of variables`
 72813 `... processing`
 72814 `. temp-file`

72815 **RATIONALE**
 72816 Some historical shells use the no-argument case as the functional equivalent of what is required
 72817 here with `-p`. This feature was left unspecified because it is not historical practice in all shells,
 72818 and some scripts may rely on the now-unspecified results on their implementations. Attempts to
 72819 specify the `-p` output as the default case were unsuccessful in achieving consensus. The `-p`
 72820 option was added to allow portable access to the values that can be saved and then later restored
 72821 using; for example, a *dot* script.

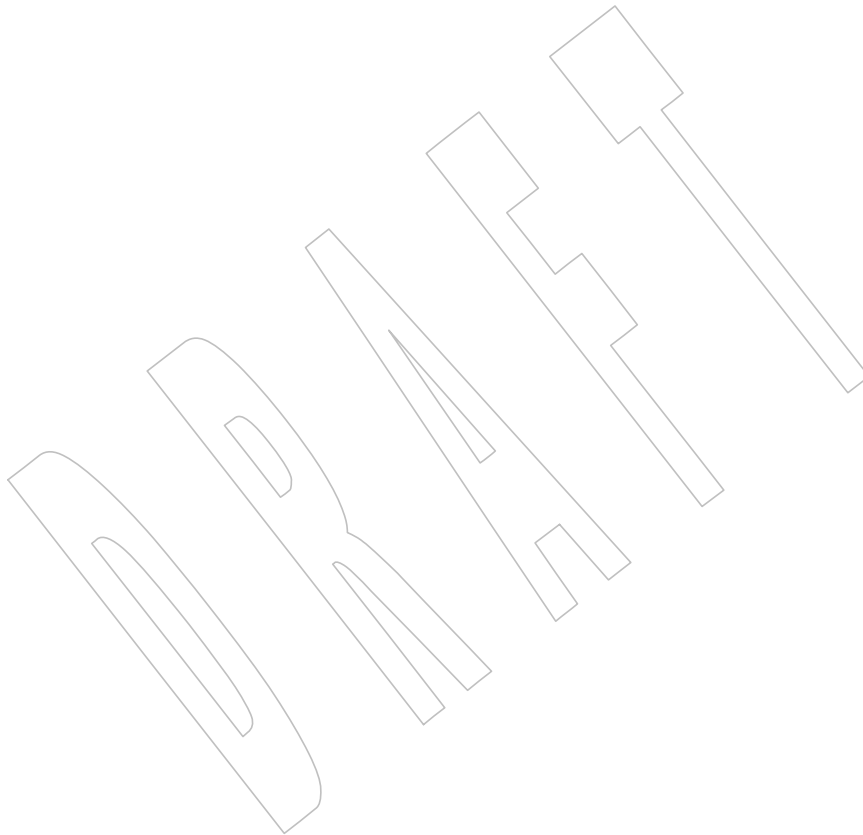
72822 **FUTURE DIRECTIONS**
 72823 None.

72824 **SEE ALSO**
 72825 [Section 2.14](#) (on page 2280)
 72826 XBD [Section 12.2](#) (on page 201)

CHANGE HISTORY

72827 **Issue 6**
 72828 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.
 72829
 72830 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 72831 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 72832 behavior is intended.
 72833
 72834 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/6 is applied, adding the following text to
 72835 the end of the first paragraph of the DESCRIPTION: "If the name of a variable is followed by
 72836 `=word`, then the value of that variable shall be set to *word*.". The reason for this change is that the
 SYNOPSIS for *export* includes:

72837 `export name[=word]...`
72838 but the meaning of the optional “=word” is never explained in the text.



72839 **NAME**
 72840 `readonly` — set the readonly attribute for variables

72841 **SYNOPSIS**
 72842 `readonly name[=word]. . .`

72843 `readonly -p`

72844 **DESCRIPTION**

72845 The variables whose *names* are specified shall be given the *readonly* attribute. The values of
 72846 variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those
 72847 variables be unset by the *unset* utility. If the name of a variable is followed by *=word*, then the
 72848 value of that variable shall be set to *word*.

72849 The *readonly* special built-in shall support XBD [Section 12.2](#) (on page 201).

72850 When `-p` is specified, *readonly* writes to the standard output the names and values of all read-
 72851 only variables, in the following format:

72852 `"readonly %s=%s\n", <name>, <value>`

72853 if *name* is set, and

72854 `"readonly %s\n", <name>`

72855 if *name* is unset.

72856 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 72857 reinput to the shell as commands that achieve the same value and *readonly* attribute-setting
 72858 results in a shell execution environment in which:

- 72859 1. Variables with values at the time they were output do not have the *readonly* attribute set.
 72860 2. Variables that were unset at the time they were output do not have a value at the time at
 72861 which the saved output is reinput to the shell.

72862 When no arguments are given, the results are unspecified.

72863 **OPTIONS**

72864 See the DESCRIPTION.

72865 **OPERANDS**

72866 See the DESCRIPTION.

72867 **STDIN**

72868 Not used.

72869 **INPUT FILES**

72870 None.

72871 **ENVIRONMENT VARIABLES**

72872 None.

72873 **ASYNCHRONOUS EVENTS**

72874 Default.

72875 **STDOUT**

72876 See the DESCRIPTION.

72877 **STDERR**
 72878 The standard error shall be used only for diagnostic messages.

72879 **OUTPUT FILES**
 72880 None.

72881 **EXTENDED DESCRIPTION**
 72882 None.

72883 **EXIT STATUS**
 72884 Zero.

72885 **CONSEQUENCES OF ERRORS**
 72886 Default.

72887 **APPLICATION USAGE**
 72888 None.

72889 **EXAMPLES**
 72890 `readonly HOME PWD`

72891 **RATIONALE**
 72892 Some historical shells preserve the *readonly* attribute across separate invocations. This volume of
 72893 POSIX.1-200x allows this behavior, but does not require it.

72894 The `-p` option allows portable access to the values that can be saved and then later restored
 72895 using, for example, a *dot* script. Also see the RATIONALE for *export* (on page 2295) for a
 72896 description of the no-argument and `-p` output cases and a related example.

72897 Read-only functions were considered, but they were omitted as not being historical practice or
 72898 particularly useful. Furthermore, functions must not be read-only across invocations to preclude
 72899 “spoofing” (spoofing is the term for the practice of creating a program that acts like a well-
 72900 known utility with the intent of subverting the real intent of the user) of administrative or
 72901 security-relevant (or security-conscious) shell scripts.

72902 **FUTURE DIRECTIONS**
 72903 None.

72904 **SEE ALSO**
 72905 [Section 2.14](#) (on page 2280)
 72906 XBD [Section 12.2](#) (on page 201)

72907 **CHANGE HISTORY**

72908 **Issue 6**
 72909 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.
 72910 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 72911 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 72912 behavior is intended.
 72913 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/7 is applied, adding the following text to
 72914 the end of the first paragraph of the DESCRIPTION: “If the name of a variable is followed by
 72915 `=word`, then the value of that variable shall be set to *word*.”. The reason for this change is that the
 72916 SYNOPSIS for *readonly* includes:
 72917 `readonly name[=word] . . .`
 72918 but the meaning of the optional “`=word`” is never explained in the text.

return*Shell Command Language*72919 **NAME**

72920 return — return from a function

72921 **SYNOPSIS**72922 return [*n*]72923 **DESCRIPTION**72924 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the
72925 shell is not currently executing a function or *dot* script, the results are unspecified.72926 **OPTIONS**

72927 None.

72928 **OPERANDS**

72929 See the DESCRIPTION.

72930 **STDIN**

72931 Not used.

72932 **INPUT FILES**

72933 None.

72934 **ENVIRONMENT VARIABLES**

72935 None.

72936 **ASYNCHRONOUS EVENTS**

72937 Default.

72938 **STDOUT**

72939 Not used.

72940 **STDERR**

72941 The standard error shall be used only for diagnostic messages.

72942 **OUTPUT FILES**

72943 None.

72944 **EXTENDED DESCRIPTION**

72945 None.

72946 **EXIT STATUS**72947 The value of the special parameter '*?*' shall be set to *n*, an unsigned decimal integer, or to the
72948 exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,
72949 the results are undefined. When *return* is executed in a *trap* action, the last command is
72950 considered to be the command that executed immediately preceding the *trap* action.72951 **CONSEQUENCES OF ERRORS**

72952 Default.

72953 **APPLICATION USAGE**

72954 None.

72955 **EXAMPLES**

72956 None.

72957 **RATIONALE**72958 The behavior of *return* when not in a function or *dot* script differs between the System V shell
72959 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is
72960 the same as *exit*.

72961 The results of returning a number greater than 255 are undefined because of differing practices

72962 in the various historical implementations. Some shells AND out all but the low-order 8 bits;
72963 others allow larger values, but not of unlimited size.

72964 See the discussion of appropriate exit status values under *exit* (on page 2293).

72965 FUTURE DIRECTIONS

72966 None.

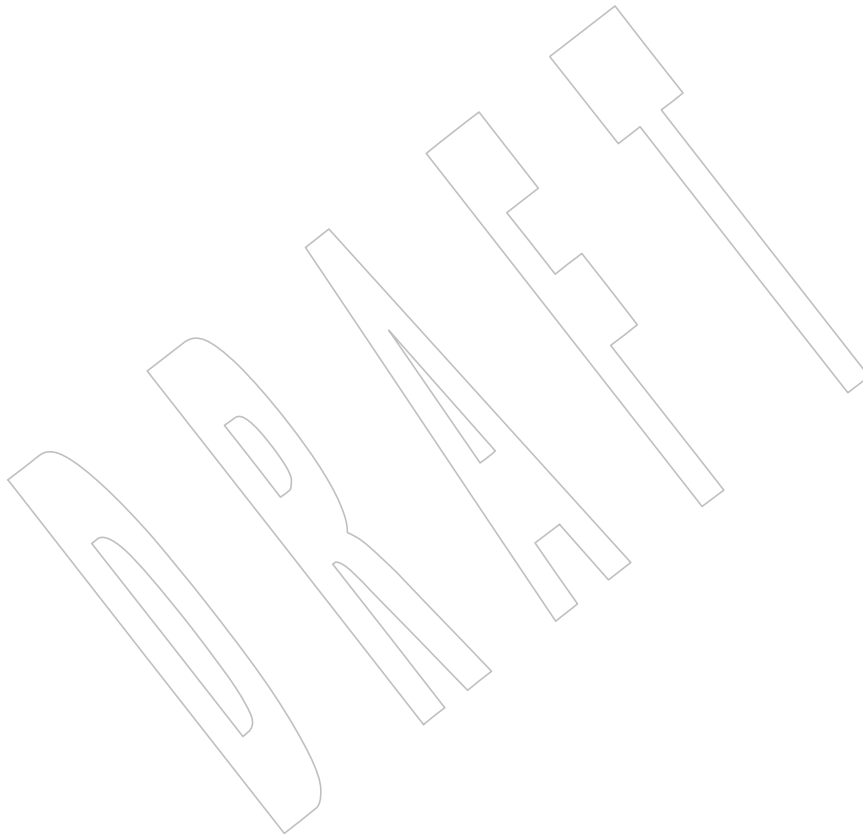
72967 SEE ALSO

72968 [Section 2.14](#) (on page 2280)

72969 CHANGE HISTORY

72970 Issue 6

72971 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
72972 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
72973 behavior is intended.



NAME

set — set or unset options and positional parameters

SYNOPSIS

XSI set [-abCefmnuvx] [-h] [-o *option*] [*argument...*]

XSI set [+abCefmnuvx] [+h] [+o *option*] [*argument...*]

set -- [*argument...*]

set -o

set +o

DESCRIPTION

If no *options* or *arguments* are specified, *set* shall write the names and values of all shell variables in the collation sequence of the current locale. Each *name* shall start on a separate line, using the format:

```
"%s=%s\n", <name>, <value>
```

The *value* string shall be written with appropriate quoting; see the description of shell quoting in [Section 2.2](#) (on page 2246). The output shall be suitable for reinput to the shell, setting or resetting, as far as possible, the variables that are currently set; read-only variables cannot be reset.

When options are specified, they shall set or unset attributes of the shell, as described below. When *arguments* are specified, they cause positional parameters to be set or unset, as described below. Setting or unsetting attributes and positional parameters are not necessarily related actions, but they can be combined in a single invocation of *set*.

The *set* special built-in shall support XBD [Section 12.2](#) (on page 201) except that options can be specified with either a leading hyphen (meaning enable the option) or plus sign (meaning disable it) unless otherwise specified.

Implementations shall support the options in the following list in both their hyphen and plus-sign forms. These options can also be specified as options to *sh*.

-a When this option is on, the *export* attribute shall be set for each variable to which an assignment is performed; see XBD [Section 4.22](#) (on page 106). If the assignment precedes a utility name in a command, the *export* attribute shall not persist in the current execution environment after the utility completes, with the exception that preceding one of the special built-in utilities causes the *export* attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the *getopts* or *read* utilities, the *export* attribute shall persist until the variable is unset.

-b This option shall be supported if the implementation supports the User Portability Utilities option. It shall cause the shell to notify the user asynchronously of background job completions. The following message is written to standard error:

```
"[%d]%c %s%s\n", <job-number>, <current>, <status>, <job-name>
```

where the fields shall be as follows:

<i><current></i>	The character '+' identifies the job that would be used as a default for the <i>fg</i> or <i>bg</i> utilities; this job can also be specified using the <i>job_id</i> "%+" or "%%". The character '-' identifies the job that would become the default if the current default job were to exit; this job can also be specified using the <i>job_id</i> "%-". For other jobs, this field is a <space>. At most one job
------------------------	---

- 73018 can be identified with '+' and at most one job can be identified with '-'.
 73019 If there is any suspended job, then the current job shall be a suspended
 73020 job. If there are at least two suspended jobs, then the previous job also
 73021 shall be a suspended job.
- 73022 <job-number> A number that can be used to identify the process group to the *wait*, *fg*, *bg*,
 73023 and *kill* utilities. Using these utilities, the job can be identified by prefixing
 73024 the job number with '% '.
- 73025 <status> Unspecified.
- 73026 <job-name> Unspecified.
- 73027 When the shell notifies the user a job has been completed, it may remove the job's process
 73028 ID from the list of those known in the current shell execution environment; see [Section](#)
 73029 [2.9.3.1](#) (on page 2266). Asynchronous notification shall not be enabled by default.
- 73030 -C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection
 73031 operator (see [Section 2.7.2](#), on page 2260); the ">|" redirection operator shall override this
 73032 *noclobber* option for an individual file.
- 73033 -e When this option is on, if a simple command fails for any of the reasons listed in [Section](#)
 73034 [2.8.1](#) (on page 2262) or returns an exit status value >0, and is not part of the compound list
 73035 following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a
 73036 pipeline preceded by the ! reserved word, then the shell shall immediately exit.
- 73037 -f The shell shall disable pathname expansion.
- 73038 XSI -h Locate and remember utilities invoked by functions as those functions are defined (the
 73039 utilities are normally located when the function is executed).
- 73040 -m This option shall be supported if the implementation supports the User Portability Utilities
 73041 option. All jobs shall be run in their own process groups. Immediately before the shell issues
 73042 a prompt after completion of the background job, a message reporting the exit status of the
 73043 background job shall be written to standard error. If a foreground job stops, the shell shall
 73044 write a message to standard error to that effect, formatted as described by the *jobs* utility. In
 73045 addition, if a job changes status other than exiting (for example, if it stops for input or
 73046 output or is stopped by a SIGSTOP signal), the shell shall write a similar message
 73047 immediately prior to writing the next prompt. This option is enabled by default for
 73048 interactive shells.
- 73049 -n The shell shall read commands but does not execute them; this can be used to check for
 73050 shell script syntax errors. An interactive shell may ignore this option.
- 73051 -o Write the current settings of the options to standard output in an unspecified format.
- 73052 +o Write the current option settings to standard output in a format that is suitable for reinput
 73053 to the shell as commands that achieve the same options settings.
- 73054 -o *option*
 73055 This option is supported if the system supports the User Portability Utilities option. It shall
 73056 set various options, many of which shall be equivalent to the single option letters. The
 73057 following values of *option* shall be supported:
- 73058 *allexport* Equivalent to -a.
- 73059 *errexit* Equivalent to -e.
- 73060 *ignoreeof* Prevent an interactive shell from exiting on end-of-file. This setting prevents
 73061 accidental logouts when <control>-D is entered. A user shall explicitly *exit* to
 73062 leave the interactive shell.

set*Shell Command Language*

73063	<i>monitor</i>	Equivalent to -m . This option is supported if the system supports the User Portability Utilities option.
73064		
73065	<i>noclobber</i>	Equivalent to -C (uppercase C).
73066	<i>noglob</i>	Equivalent to -f .
73067	<i>noexec</i>	Equivalent to -n .
73068	<i>nolog</i>	Prevent the entry of function definitions into the command history; see Command History List (on page 3078).
73069		
73070	<i>notify</i>	Equivalent to -b .
73071	<i>nounset</i>	Equivalent to -u .
73072	<i>verbose</i>	Equivalent to -v .
73073	<i>vi</i>	Allow shell command line editing using the built-in <i>vi</i> editor. Enabling <i>vi</i> mode shall disable any other command line editing mode provided as an implementation extension.
73074		
73075		
73076		It need not be possible to set <i>vi</i> mode on for certain block-mode terminals.
73077	<i>xtrace</i>	Equivalent to -x .
73078	-u	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.
73079		
73080	-v	The shell shall write its input to standard error as it is read.
73081	-x	The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.
73082		
73083		
73084		The default for all these options shall be off (unset) unless stated otherwise in the description of the option or unless the shell was invoked with them on; see <i>sh</i> .
73085		
73086		The remaining arguments shall be assigned in order to the positional parameters. The special parameter '# ' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.
73087		
73088		
73089		If the first argument is '- ', the results are unspecified.
73090		The special argument "--" immediately following the <i>set</i> command name can be used to delimit the arguments if the first argument begins with '+' or '-', or to prevent inadvertent listing of all shell variables when there are no arguments. The command <i>set --</i> without <i>argument</i> shall unset all positional parameters and set the special parameter '# ' to zero.
73091		
73092		
73093		

OPTIONS

See the DESCRIPTION.

OPERANDS

See the DESCRIPTION.

STDIN

Not used.

INPUT FILES

None.

73102 **ENVIRONMENT VARIABLES**

73103 None.

73104 **ASYNCHRONOUS EVENTS**

73105 Default.

73106 **STDOUT**

73107 See the DESCRIPTION.

73108 **STDERR**

73109 The standard error shall be used only for diagnostic messages.

73110 **OUTPUT FILES**

73111 None.

73112 **EXTENDED DESCRIPTION**

73113 None.

73114 **EXIT STATUS**

73115 Zero.

73116 **CONSEQUENCES OF ERRORS**

73117 Default.

73118 **APPLICATION USAGE**

73119 None.

73120 **EXAMPLES**

73121 Write out all variables and their values:

73122 `set`

73123 Set \$1, \$2, and \$3 and set "\$#" to 3:

73124 `set c a b`73125 Turn on the `-x` and `-v` options:73126 `set -xv`

73127 Unset all positional parameters:

73128 `set --`73129 Set \$1 to the value of `x`, even if it begins with `'-'` or `'+'`:73130 `set -- "$x"`73131 Set the positional parameters to the expansion of `x`, even if `x` expands with a leading `'-'` or `'+'`:73132 `set -- $x`73133 **RATIONALE**

73134 The `set --` form is listed specifically in the SYNOPSIS even though this usage is implied by the
 73135 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether
 73136 the `set --` form might be misinterpreted as being equivalent to `set` without any options or
 73137 arguments. The functionality of this form has been adopted from the KornShell. In System V, `set`
 73138 `--` only unsets parameters if there is at least one argument; the only way to unset all parameters
 73139 is to use `shift`. Using the KornShell version should not affect System V scripts because there
 73140 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

73141 `set -- "$@"`

73142 and there were in fact no arguments resulting from `"$@"`, unsetting the parameters would have
 73143 no result.

73144 The *set +* form in early proposals was omitted as being an unnecessary duplication of *set* alone
 73145 and not widespread historical practice.

73146 The *noclobber* option was changed to allow *set -C* as well as the *set -o noclobber* option. The
 73147 single-letter version was added so that the historical "\$-" paradigm would not be broken; see
 73148 [Section 2.5.2](#) (on page 2250).

73149 The *-h* flag is related to command name hashing and is only required on XSI-conformant
 73150 systems.

73151 The following *set* flags were omitted intentionally with the following rationale:

73152 **-k** The *-k* flag was originally added by the author of the Bourne shell to make it easier for
 73153 users of pre-release versions of the shell. In early versions of the Bourne shell the construct
 73154 *set name=value* had to be used to assign values to shell variables. The problem with *-k* is
 73155 that the behavior affects parsing, virtually precluding writing any compilers. To explain the
 73156 behavior of *-k*, it is necessary to describe the parsing algorithm, which is implementation-
 73157 defined. For example:

```
73158     set -k; echo name=value
```

73159 and:

```
73160     set -k
73161     echo name=value
```

73162 behave differently. The interaction with functions is even more complex. What is more, the
 73163 *-k* flag is never needed, since the command line could have been reordered.

73164 **-t** The *-t* flag is hard to specify and almost never used. The only known use could be done
 73165 with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page
 73166 says that it exits after reading and executing one command. What is one command? If the
 73167 input is *date;date*, *sh* executes both *date* commands while *ksh* does only the first.

73168 Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion
 73169 was that the *unset* utility should be used to unset options instead of using the non-*getopt*(-)-able
 73170 *+option* syntax. However, the conclusion was reached that the historical practice of using *+option*
 73171 was satisfactory and that there was no compelling reason to modify such widespread historical
 73172 practice.

73173 The *-o* option was adopted from the KornShell to address user needs. In addition to its
 73174 generally friendly interface, *-o* is needed to provide the *vi* command line editing mode, for
 73175 which historical practice yields no single-letter option name. (Although it might have been
 73176 possible to invent such a letter, it was recognized that other editing modes would be developed
 73177 and *-o* provides ample name space for describing such extensions.)

73178 Historical implementations are inconsistent in the format used for *-o* option status reporting.
 73179 The *+o* format without an option-argument was added to allow portable access to the options
 73180 that can be saved and then later restored using, for instance, a dot script.

73181 Historically, *sh* did trace the command *set +x*, but *ksh* did not.

73182 The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically
 73183 <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

73184 The *set -m* option was added to apply only to the UPE because it applies primarily to interactive
 73185 use, not shell script applications.

73186 The ability to do asynchronous notification became available in the 1988 version of the
 73187 KornShell. To have it occur, the user had to issue the command:

```
73188     trap "jobs -n" CLD
```

73189 The C shell provides two different levels of an asynchronous notification capability. The
 73190 environment variable *notify* is analogous to what is done in *set -b* or *set -o notify*. When *set*, it
 73191 notifies the user immediately of background job completions. When *unset*, this capability is
 73192 turned off.

73193 The other notification ability comes through the built-in utility *notify*. The syntax is:

```
73194 notify [%job ... ]
```

73195 By issuing *notify* with no operands, it causes the C shell to notify the user asynchronously when
 73196 the state of the current job changes. If given operands, *notify* asynchronously informs the user of
 73197 changes in the states of the specified jobs.

73198 To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*,
 73199 nor the C shell *notify* environment variable seemed appropriate (*notify* is not a proper POSIX
 73200 environment variable name).

73201 The *set -b* option was selected as a compromise.

73202 The *notify* built-in was considered to have more functionality than was required for simple
 73203 asynchronous notification.

73204 FUTURE DIRECTIONS

73205 None.

73206 SEE ALSO

73207 [Section 2.14](#) (on page 2280)

73208 XBD [Section 4.22](#) (on page 106), [Section 12.2](#) (on page 201) +

73209 CHANGE HISTORY

73210 Issue 6

73211 The obsolescent *set* command name followed by ‘-’ has been removed.

73212 The following new requirements on POSIX implementations derive from alignment with the
 73213 Single UNIX Specification:

- 73214 • The *nolog* option is added to *set -o*.

73215 IEEE PASC Interpretation 1003.2 #167 is applied, clarifying that the options default also takes
 73216 into account the description of the option.

73217 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
 73218 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
 73219 behavior is intended.

73220 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/8 is applied, changing the square
 73221 brackets in the example in RATIONALE to be in bold, which is the typeface used for optional
 73222 items.

73223 Issue 7

73224 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
 73225 argument is ‘-’.

73226 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

shift73227 **NAME**

73228 shift — shift positional parameters

73229 **SYNOPSIS**73230 shift [*n*]73231 **DESCRIPTION**

73232 The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of
 73233 parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The
 73234 parameters represented by the numbers "\$#" down to "\$#-*n*+1" shall be unset, and the
 73235 parameter '#' is updated to reflect the new number of positional parameters.

73236 The value *n* shall be an unsigned decimal integer less than or equal to the value of the special
 73237 parameter '#'. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special
 73238 parameters are not changed.

73239 **OPTIONS**

73240 None.

73241 **OPERANDS**

73242 See the DESCRIPTION.

73243 **STDIN**

73244 Not used.

73245 **INPUT FILES**

73246 None.

73247 **ENVIRONMENT VARIABLES**

73248 None.

73249 **ASYNCHRONOUS EVENTS**

73250 Default.

73251 **STDOUT**

73252 Not used.

73253 **STDERR**

73254 The standard error shall be used only for diagnostic messages.

73255 **OUTPUT FILES**

73256 None.

73257 **EXTENDED DESCRIPTION**

73258 None.

73259 **EXIT STATUS**73260 The exit status is >0 if *n*>\$#; otherwise, it is zero.73261 **CONSEQUENCES OF ERRORS**

73262 Default.

73263
73264
73265
73266
73267
73268
73269
73270
73271
73272
73273
73274
73275
73276
73277
73278
73279
73280

APPLICATION USAGE

None.

EXAMPLES

```
$ set a b c d e  
$ shift 2  
$ echo $*  
c d e
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.14](#) (on page 2280)

CHANGE HISTORY**Issue 6**

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in behavior is intended.

73281 **NAME**73282 `times` — write process times73283 **SYNOPSIS**73284 `times`73285 **DESCRIPTION**73286 The `times` utility shall write the accumulated user and system times for the shell and for all of its
73287 child processes, in the following POSIX locale format:

```
73288 "%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
73289 <shell user seconds>, <shell system minutes>,
73290 <shell system seconds>, <children user minutes>,
73291 <children user seconds>, <children system minutes>,
73292 <children system seconds>
```

73293 The four pairs of times shall correspond to the members of the `<sys/times.h>` `tms` structure |
73294 (defined in XBD [Chapter 13](#), on page 205) as returned by `times()`: `tms_utime`, `tms_stime`,
73295 `tms_cutime`, and `tms_cstime`, respectively.

73296 **OPTIONS**

73297 None.

73298 **OPERANDS**

73299 None.

73300 **STDIN**

73301 Not used.

73302 **INPUT FILES**

73303 None.

73304 **ENVIRONMENT VARIABLES**

73305 None.

73306 **ASYNCHRONOUS EVENTS**

73307 Default.

73308 **STDOUT**

73309 See the DESCRIPTION.

73310 **STDERR**

73311 The standard error shall be used only for diagnostic messages.

73312 **OUTPUT FILES**

73313 None.

73314 **EXTENDED DESCRIPTION**

73315 None.

73316 **EXIT STATUS**

73317 Zero.

73318 **CONSEQUENCES OF ERRORS**

73319 Default.

73320
73321
73322
73323
73324
73325
73326
73327
73328
73329
73330
73331
73332
73333
73334
73335
73336
73337
73338
73339

APPLICATION USAGE

None.

EXAMPLES

```
$ times
0m0.43s 0m1.11s
8m44.18s 1m43.23s
```

RATIONALE

The *times* special built-in from the Single UNIX Specification is now required for all conforming shells.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.14](#) (on page 2280)

XBD [<sys/times.h>](#)

+

CHANGE HISTORY**Issue 6**

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/9 is applied, changing text in the DESCRIPTION from: "Write the accumulated user and system times for the shell and for all of its child processes ..." to: "The *times* utility shall write the accumulated user and system times for the shell and for all of its child processes ...".

73340 **NAME**

73341 trap — trap signals

73342 **SYNOPSIS**73343 trap *n* [*condition...*] +73344 trap [*action condition...*]73345 **DESCRIPTION**

73346 If the first operand is an unsigned decimal integer, the shell shall treat all operands as +
 73347 conditions, and shall reset each condition to the default value. Otherwise, if there are operands, +
 73348 the first is treated as an action and the remaining as conditions. +

73349 If *action* is '-', the shell shall reset each *condition* to the default value. If *action* is null (" "), the
 73350 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read
 73351 and executed by the shell when one of the corresponding conditions arises. The action of *trap*
 73352 shall override a previous action (either default action or one explicitly set). The value of "\$?"
 73353 after the *trap* action completes shall be the value it had before *trap* was invoked.

73354 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,
 73355 without the SIG prefix, as listed in the tables of signal names in the <signal.h> header defined in |
 73356 XBD Chapter 13 (on page 205); for example, HUP, INT, QUIT, TERM. Implementations may |
 73357 permit names with the SIG prefix or ignore case in signal names as an extension. Setting a trap |
 73358 for SIGKILL or SIGSTOP produces undefined results.

73359 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment
 73360 immediately after the last command executed before the *trap* on EXIT was taken.

73361 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

73362 eval *action*

73363 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although
 73364 no error need be reported when attempting to do so. An interactive shell may reset or catch
 73365 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed
 73366 with another *trap* command.

73367 When a subshell is entered, traps that are not being ignored are set to the default actions. This
 73368 does not imply that the *trap* command cannot be used within the subshell to set new traps.

73369 The *trap* command with no arguments shall write to standard output a list of commands
 73370 associated with each condition. The format shall be:

73371 "trap -- %s %s ... \n", <action>, <condition> ...

73372 The shell shall format the output, including the proper use of quoting, so that it is suitable for
 73373 reinput to the shell as commands that achieve the same trapping results. For example:

73374 save_traps=\$(trap)

73375 ...

73376 eval "\$save_traps"

73377 XSI
 73378 XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to
 the following signal names:

73379 1 SIGHUP

73380 2 SIGINT

73381	3	SIGQUIT
73382	6	SIGABRT
73383	9	SIGKILL
73384	14	SIGALRM
73385	15	SIGTERM

73386 The *trap* special built-in shall conform to XBD [Section 12.2](#) (on page 201).

73387 OPTIONS

73388 None.

73389 OPERANDS

73390 See the DESCRIPTION.

73391 STDIN

73392 Not used.

73393 INPUT FILES

73394 None.

73395 ENVIRONMENT VARIABLES

73396 None.

73397 ASYNCHRONOUS EVENTS

73398 Default.

73399 STDOUT

73400 See the DESCRIPTION.

73401 STDERR

73402 The standard error shall be used only for diagnostic messages.

73403 OUTPUT FILES

73404 None.

73405 EXTENDED DESCRIPTION

73406 None.

73407 EXIT STATUS

73408 XSI If the trap name **or number** is invalid, a non-zero exit status shall be returned; otherwise, zero
 73409 XSI shall be returned. For both interactive and non-interactive shells, invalid signal names **or**
 73410 **numbers** shall not be considered a syntax error and do not cause the shell to abort.

73411 CONSEQUENCES OF ERRORS

73412 Default.

73413 APPLICATION USAGE

73414 None.

73415 EXAMPLES

73416 Write out a list of all traps and actions:

73417 trap

73418 Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable
 73419 executes when the shell terminates:

73420 trap '\$HOME/logout' EXIT

73421 or:

73422 trap '\$HOME/logout' 0

73423 Unset traps on INT, QUIT, TERM, and EXIT:

73424 trap - INT QUIT TERM EXIT

73425 RATIONALE

73426 Implementations may permit lowercase signal names as an extension. Implementations may
73427 also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill*
73428 utilities in this volume of POSIX.1-200x are now consistent in their omission of the SIG prefix for
73429 signal names. Some *kill* implementations do not allow the prefix, and *kill -l* lists the signals
73430 without prefixes.

73431 Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but
73432 it has no effect. Portable POSIX applications cannot attempt to trap these signals.

73433 The output format is not historical practice. Since the output of historical *trap* commands is not
73434 portable (because numeric signal values are not portable) and had to change to become so, an
73435 opportunity was taken to format the output in a way that a shell script could use to save and
73436 then later reuse a trap if it wanted.

73437 The KornShell uses an **ERR** trap that is triggered whenever *set -e* would cause an exit. This is
73438 allowable as an extension, but was not mandated, as other shells have not used it.

73439 The text about the environment for the EXIT trap invalidates the behavior of some historical
73440 versions of interactive shells which, for example, close the standard input before executing a
73441 trap on 0. For example, in some historical interactive shell sessions the following trap on 0
73442 would always print "--":

73443 trap 'read foo; echo "--\$foo--"' 0

73444 The command:

73445 trap '\$cmd' 0

73446 causes the contents of the shell variable *cmd* to be executed as a command when the shell exits. +
73447 Using double-quotes instead of single-quotes might have unexpected behavior, since in theory +
73448 the value of *cmd* might be a decimal integer which would be treated as a condition, not an +
73449 action; or *cmd* might begin with '-'. Also, using double-quotes will cause the value of *cmd* to be +
73450 expanded twice, once when *trap* is executed, and once when the condition arises.

73451 FUTURE DIRECTIONS

73452 None.

73453 SEE ALSO

73454 [Section 2.14](#) (on page 2280)

73455 [XBD Section 12.2](#) (on page 201), [<signal.h>](#)

73456 CHANGE HISTORY

73457 Issue 6

73458 XSI-conforming implementations provide the mapping of signal names to numbers given above
73459 (previously this had been marked obsolescent). Other implementations need not provide this
73460 optional mapping.

73461 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
73462 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
73463 behavior is intended.

73464

Issue 7

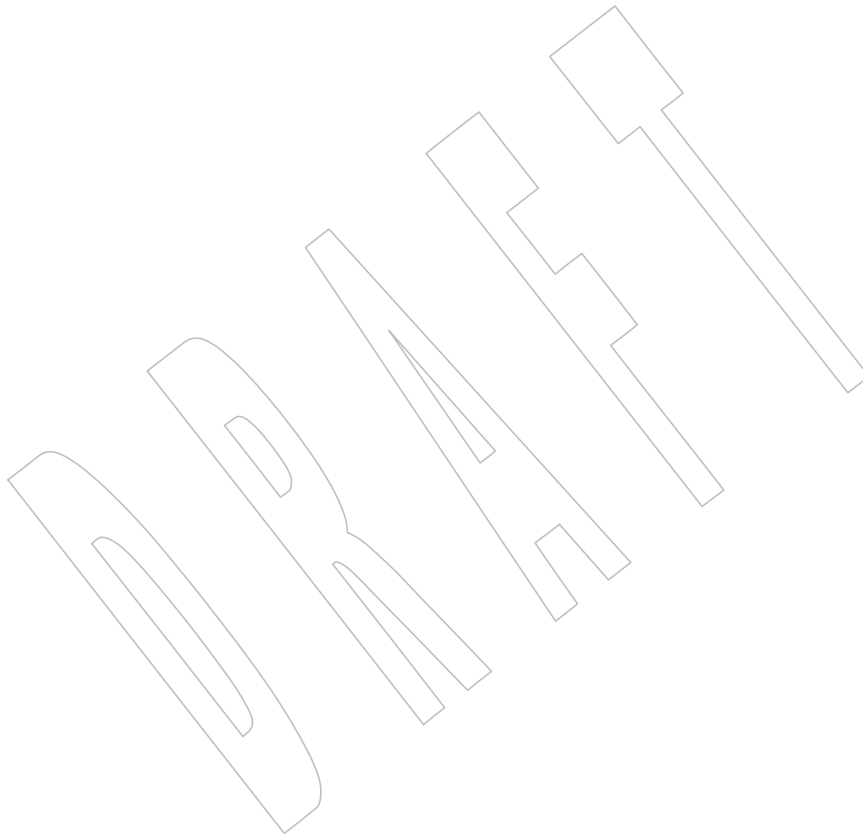
73465

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

73466

Austin Group Interpretation 1003.1-2001 #116 is applied.

+



73467 **NAME**
 73468 `unset` — unset values and attributes of variables and functions

73469 **SYNOPSIS**
 73470 `unset [-fv] name...`

73471 **DESCRIPTION**
 73472 Each variable or function specified by *name* shall be unset.
 73473 If `-v` is specified, *name* refers to a variable name and the shell shall unset it and remove it from
 73474 the environment. Read-only variables cannot be unset.
 73475 If `-f` is specified, *name* refers to a function and the shell shall unset the function definition.
 73476 If neither `-f` nor `-v` is specified, *name* refers to a variable; if a variable by that name does not
 73477 exist, it is unspecified whether a function by that name, if any, shall be unset.
 73478 Unsetting a variable or function that was not previously set shall not be considered an error and
 73479 does not cause the shell to abort.

73480 The *unset* special built-in shall support XBD [Section 12.2](#) (on page 201).

73481 Note that:

73482 `VARIABLE=`

73483 is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to " ". Also, the
 73484 variables that can be *unset* should not be misinterpreted to include the special parameters (see
 73485 [Section 2.5.2](#), on page 2250).

73486 **OPTIONS**
 73487 See the DESCRIPTION.

73488 **OPERANDS**
 73489 See the DESCRIPTION.

73490 **STDIN**
 73491 Not used.

73492 **INPUT FILES**
 73493 None.

73494 **ENVIRONMENT VARIABLES**
 73495 None.

73496 **ASYNCHRONOUS EVENTS**
 73497 Default.

73498 **STDOUT**
 73499 Not used.

73500 **STDERR**
 73501 The standard error shall be used only for diagnostic messages.

73502 **OUTPUT FILES**
 73503 None.

73504 **EXTENDED DESCRIPTION**
 73505 None.

73506 **EXIT STATUS**73507 0 All *name* operands were successfully unset.73508 >0 At least one *name* could not be unset.73509 **CONSEQUENCES OF ERRORS**

73510 Default.

73511 **APPLICATION USAGE**

73512 None.

73513 **EXAMPLES**73514 Unset *VISUAL* variable:

73515 unset -v VISUAL

73516 Unset the functions **foo** and **bar**:

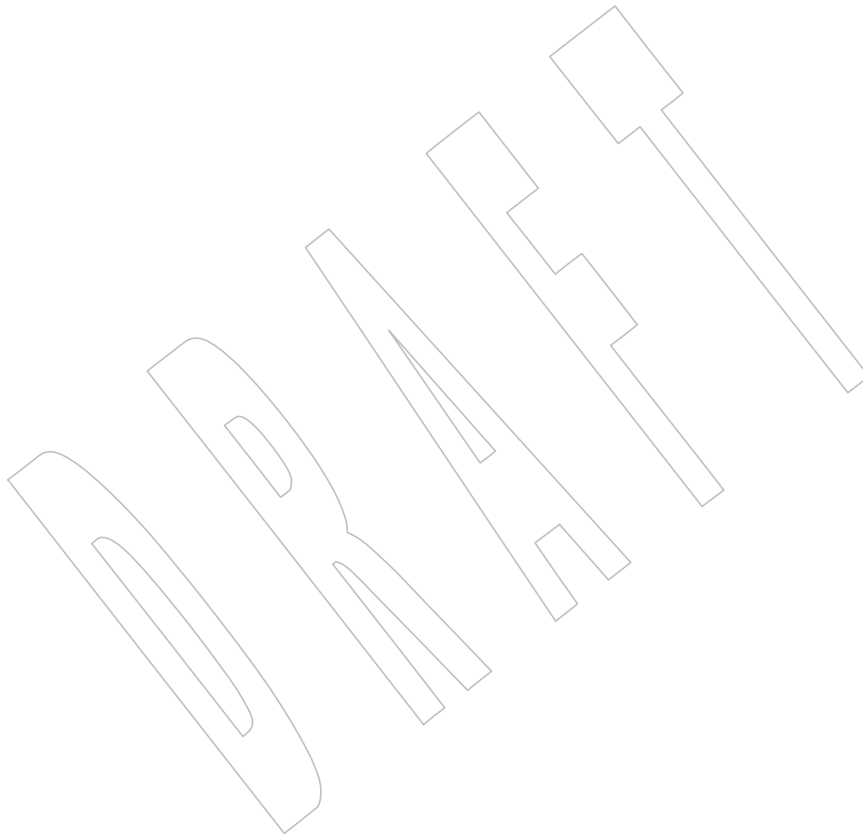
73517 unset -f foo bar

73518 **RATIONALE**73519 Consideration was given to omitting the `-f` option in favor of an *unfunction* utility, but the
73520 standard developers decided to retain historical practice.73521 The `-v` option was introduced because System V historically used one name space for both
73522 variables and functions. When *unset* is used without options, System V historically unset either a
73523 function or a variable, and there was no confusion about which one was intended. A portable
73524 POSIX application can use *unset* without an option to unset a variable, but not a function; the `-f`
73525 option must be used.73526 **FUTURE DIRECTIONS**

73527 None.

73528 **SEE ALSO**73529 [Section 2.14](#) (on page 2280)73530 XBD [Section 12.2](#) (on page 201)73531 **CHANGE HISTORY**73532 **Issue 6**73533 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/5 is applied so that the reference page
73534 sections use terms as described in the Utility Description Defaults ([Section 1.4](#)). No change in
73535 behavior is intended.73536 **Issue 7**

73537 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



Batch Environment Services

73538

73539

73540

OB BE

This chapter describes the services and utilities that shall be implemented on all systems that claim conformance to the Batch Environment Services and Utilities option. The functionality described in this section shall be provided on implementations that support the Batch Environment Services and Utilities option (and the rest of this section is not further shaded for this option).

73541

73542

73543

73544

73545

Note that the Batch Environment Services and Utilities option is marked obsolescent in Issue 7.

73546

3.1 General Concepts

73547

3.1.1 Batch Client-Server Interaction

73548

73549

73550

Batch jobs are created and managed by batch servers. A batch client interacts with a batch server to access batch services on behalf of the user. In order to use batch services, a user must have access to a batch client.

73551

73552

A batch server is a computational entity, such as a daemon process, that provides batch services. Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

73553

73554

73555

73556

73557

The batch utilities described in this volume of POSIX.1-200x (and listed in [Table 3-1](#)) are clients of batch services; they allow users to perform actions on the job such as creating, modifying, and deleting batch jobs from a shell command line. Although these batch utilities may be said to accomplish certain services, they actually obtain services on behalf of a user by means of requests to batch servers.

73558

Table 3-1 Batch Utilities

73559

73560

73561

<i>qalter</i>	<i>qmove</i>	<i>qrls</i>	<i>qstat</i>
<i>qdel</i>	<i>qmsg</i>	<i>qselect</i>	<i>qsub</i>
<i>qhold</i>	<i>qrerun</i>	<i>qsig</i>	

73562

73563

73564

73565

73566

Client-server interaction takes place by means of the batch requests defined in this chapter. Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

73567 3.1.2 Batch Queues

73568 Two types of batch queue are described: routing queues and execution queues. When a batch job
73569 is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing
73570 queues under the following conditions:

- 73571 • The batch job has been routed to another queue.
- 73572 • The batch job has been deleted from the batch queue.
- 73573 • The batch job has been aborted.

73574 When a batch job is placed in an execution queue, it is a candidate for execution.

73575 A batch job is removed from an execution queue under the following conditions:

- 73576 • The batch job has been executed and exited.
- 73577 • The batch job has been aborted.
- 73578 • The batch job has been deleted from the batch queue.
- 73579 • The batch job has been moved to another queue.

73580 Access to a batch queue is limited to the batch server that manages the batch queue. Clients
73581 never access a batch queue or a batch job directly, either to read or write information; all client
73582 access to batch queues or jobs takes place through batch servers.

73583 3.1.3 Batch Job Creation

73584 When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier
73585 to the job. A batch job identifier consists of both a sequence number that is unique among the
73586 sequence numbers issued by that server and the name of the server. Since the batch server name
73587 is unique within a name space, the job identifier is likewise unique within the name space.

73588 The batch server that creates a batch job shall return the batch server-assigned job identifier to
73589 the client that requested the job creation. If the batch server routes or moves the job to another
73590 server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job
73591 shall never change.

73592 3.1.4 Batch Job Tracking

73593 Since a batch job may be moved after creation, the batch server name component of the job
73594 identifier need not indicate the location of the job. An implementation may provide a batch job
73595 tracking mechanism, in which case the user generally does not need to know the location of the
73596 job. However, an implementation need not provide a batch job tracking mechanism, in which
73597 case the user must find routed jobs by probing the possible destinations.

73598 3.1.5 Batch Job Routing

73599 To route a batch job, a batch server either moves the job to some other queue that is managed by
73600 the batch server, or requests that some other batch server accept the job.

73601 Each routing queue has one or more queues to which it can route batch jobs. The batch server
73602 administrator creates routing queues.

73603 A batch server may route a batch job from a routing queue to another routing queue. Batch
73604 servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a
73605 batch server routes jobs from the routing queues that it manages. The algorithm by which a
73606 batch server selects a batch queue to which to route a batch job is implementation-defined.

73607 A batch job need not be eligible for routing to all the batch queues fed by the routing queue from
 73608 which it is routed. A batch server that has been asked to accept the job may reject the request if
 73609 the job requires resources that are unavailable to that batch server, or if the client is not
 73610 authorized to access the batch server.

73611 Batch servers may route high-priority jobs before low-priority jobs, but, on other than
 73612 overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a
 73613 routing queue reject requests to accept the job for reasons that are permanent, the batch server
 73614 that manages the job shall abort the job. If all or some rejections are temporary, the batch server
 73615 should try to route the job again at some later point.

73616 The reasons for rejecting a batch job are implementation-defined.

73617 The reasons for which the routing should be retried later and the reasons for which the job
 73618 should be aborted are also implementation-defined.

73619 3.1.6 Batch Job Execution

73620 To execute a batch job is to create a session leader (a process) that runs the shell program
 73621 indicated by the *Shell_Path* attribute of the job. The script shall be passed to the program as its
 73622 standard input. An implementation may pass the script to the program by other
 73623 implementation-defined means. At the time a batch job begins execution, it is defined to enter
 73624 the RUNNING state. The primary program that is executed by a batch job is typically, though
 73625 not necessarily, a shell program.

73626 A batch server shall execute eligible jobs as a deferred service—no client request is necessary
 73627 once the batch job is created and eligible. However, the attributes of a batch job, such as the job
 73628 hold type, may render the job ineligible. A batch server shall scan the execution queues that it
 73629 manages for jobs that are eligible for execution. The algorithm by which the batch server selects
 73630 eligible jobs for execution is implementation-defined.

73631 As part of creating the process for the batch job, the batch server shall open the standard output
 73632 and standard error streams of the session.

73633 The attributes of a batch job may indicate that the batch server executing the job shall send mail
 73634 to a list of users at the time it begins execution of the job.

73635 3.1.7 Batch Job Exit

73636 When the session leader of an executing job terminates, the job exits. As part of exiting a batch
 73637 job, the batch server that manages the job shall remove the job from the batch queue in which it
 73638 resides. The server shall transfer output files of the job to a location described by the attributes of
 73639 the job.

73640 The attributes of a batch job may indicate that the batch server managing the job shall send mail
 73641 to a list of users at the time the job exits.

73642 3.1.8 Batch Job Abort

73643 A batch server shall abort jobs for which a required deferred service cannot be performed. The
 73644 attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a
 73645 list of users at the time it aborts the job.

73646 3.1.9 Batch Authorization

73647 Clients, such as the batch environment utilities (marked BE), access batch services by means of
73648 requests to one or more batch servers. To acquire the services of any given batch server, the user
73649 identifier under which the client runs must be authorized to use that batch server.

73650 The user with an associated user name that creates a batch job shall own the job and can perform
73651 actions such as read, modify, delete, and move.

73652 A user identifier of the same value at a different host need not be the same user. For example,
73653 user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at
73654 host **beta**. Likewise, the same person may have access to different user names on different hosts.

73655 An implementation may optionally provide an authorization mechanism that permits one user
73656 name to access jobs under another user name.

73657 A process on a client host may be authorized to run processes under multiple user names at a
73658 batch server host. Where appropriate, the utilities defined in this volume of POSIX.1-200x
73659 provide a means for a user to choose from among such user names when creating or modifying
73660 a batch job.

73661 3.1.10 Batch Administration

73662 The processing of a batch job by a batch server is affected by the attributes of the job. The
73663 processing of a batch job may also be affected by the attributes of the batch queue in which the
73664 job resides and by the status of the batch server that manages the job. See also XBD [Chapter 3](#)
73665 (on page 33) for batch definitions.

73666 3.1.11 Batch Notification

73667 Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub*
73668 utility creates a batch job and exits. For this reason, batch servers notify users of batch job events
73669 by sending mail to the user that owns the job, or to other designated users.

73670 3.2 Batch Services

73671 The presence of Batch Environment Services and Utilities option services is indicated by the
73672 configuration variable `POSIX2_PBS`. A conforming batch server provides services as defined in
73673 this section.

73674 A batch server shall provide batch services in two ways:

- 73675 1. The batch server provides a service at the request of a client.
- 73676 2. The batch server provides a deferred service as a result of a change in conditions
73677 monitored by the batch server.

73678 If a batch server cannot complete a request, it shall reject the request. If a batch server cannot
73679 complete a deferred service for a batch job, the batch server shall abort the batch job. [Table 3-2](#)
73680 (on page 2323) is a summary of environment variables that shall be supported by an
73681 implementation of the batch server and utilities.

73682

Table 3-2 Environment Variable Summary

73683

Variable	Description
<i>PBS_DPREFIX</i>	Defines the directive prefix (see <i>qsub</i>)
<i>PBS_ENVIRONMENT</i>	Batch Job is batch or interactive (see Section 3.2.2.1)
<i>PBS_JOBID</i>	The <i>job_identifier</i> attribute of job (see Section 3.2.3.8)
<i>PBS_JOBNAME</i>	The <i>job_name</i> attribute of job (see Section 3.2.3.8)
<i>PBS_O_HOME</i>	Defines the <i>HOME</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_HOST</i>	Defines the host name of the batch client (see <i>qsub</i>)
<i>PBS_O_LANG</i>	Defines the <i>LANG</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_LOGNAME</i>	Defines the <i>LOGNAME</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_MAIL</i>	Defines the <i>MAIL</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_PATH</i>	Defines the <i>PATH</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_QUEUE</i>	Defines the submit queue of the batch client (see <i>qsub</i>)
<i>PBS_O_SHELL</i>	Defines the <i>SHELL</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_TZ</i>	Defines the <i>TZ</i> of the batch client (see <i>qsub</i>)
<i>PBS_O_WORKDIR</i>	Defines the working directory of the batch client (see <i>qsub</i>)
<i>PBS_QUEUE</i>	Defines the initial execution queue (see Section 3.2.2.1)

73684

73685

73686

73687

73688

73689

73690

73691

73692

73693

73694

73695

73696

73697

73698

73699

3.2.1 Batch Job States

73700

A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD, WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests that the batch server that manages the batch job can accept for the batch job. A batch server shall change the state of a batch job either in response to service requests from clients or as a result of deferred services, such as job execution or job routing.

73701

73702

73703

73704

73705

73706

A batch job that is in the QUEUED state resides in a queue but is still pending either execution or routing, depending on the queue type.

73707

73708

73709

73710

73711

A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED state. A batch server that puts a batch job in an execution queue, but has not yet executed the batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution queue and is executing is defined to be in the RUNNING state. While a batch job is in the RUNNING state, a session leader is associated with the batch job.

73712

73713

A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute, is defined to be in the HELD state.

73714

73715

A batch job that is not held, but must wait until a future date and time before executing, is defined to be in the WAITING state.

73716

73717

When the session leader associated with a running job exits, the batch job shall be placed in the EXITING state.

73718

73719

73720

73721

73722

A batch job for which the session leader has terminated is defined to be in the EXITING state, and the batch server that manages such a batch job cannot accept job modification requests that affect the batch job. While a batch job is in the EXITING state, the batch server that manages the batch job is staging output files and notifying clients of job completion. Once a batch job has exited, it no longer exists as an object managed by a batch server.

73723

73724

A batch job that is being moved from a routing queue to another queue is defined to be in the TRANSITING state.

73725

73726

When a batch job in a routing queue has been selected to be moved to a new destination, then the batch job shall be in either the QUEUED state or the TRANSITING state, depending on the

73727 batch server implementation.

73728 Batch jobs with either an *Execution_Time* attribute value set in the future or a *Hold_Types* attribute
73729 of value not equal to NO_HOLD, or both, may be routed or held in the routing queue. The
73730 treatment of jobs with the *Execution_Time* or *Hold_Types* attributes in a routing queue is
73731 implementation-defined.

73732 When a batch job in a routing queue has not been selected to be moved to a new destination and
73733 the batch job has a *Hold_Types* attribute value of other than NO_HOLD, then the job should be in
73734 the HELD state.

73735 **Note:** The effect of a hold upon a batch job in a routing queue is implementation-defined. The
73736 implementation should use the state that matches whether the batch job can route with a hold
73737 or not.

73738 When a batch job in a routing queue has not been selected to be moved to a new destination and
73739 the batch job has:

- 73740 • A *Hold_Types* attribute value of NO_HOLD
- 73741 • An *Execution_Time* attribute in the past

73742 then the batch job shall be in the QUEUED state.

73743 When a batch job in a routing queue has not been selected to be moved to a new destination and
73744 the batch job has:

- 73745 • A *Hold_Types* attribute value of NO_HOLD
- 73746 • An *Execution_Time* attribute in the future

73747 then the batch job may be in the WAITING state.

73748 **Note:** The effect of a future execution time upon a batch job in a routing queue is implementation-
73749 defined. The implementation should use the state that matches whether the batch job can route
73750 with a hold or not.

73751 [Table 3-3](#) (on page 2325) describes the next state of a batch job, given the current state of the
73752 batch job and the type of request. [Table 3-4](#) (on page 2326) describes the response of a batch
73753 server to a request, given the current state of the batch job and the type of request.

73754 3.2.2 Deferred Batch Services

73755 This section describes the deferred services performed by batch servers: job execution, job
73756 routing, job exit, job abort, and the rerunning of jobs after a restart.

73757 3.2.2.1 Batch Job Execution

73758 To execute a batch job is to create a session leader (a process) that runs the shell program
73759 indicated by the *Shell_Path_List* attribute of the batch job. The script is passed to the program as
73760 its standard input. An implementation may pass the script to the program by other
73761 implementation-defined means. At the time a batch job begins execution, it is defined to enter
73762 the RUNNING state.

73763

Table 3-3 Next State Table

73764

73765

73766

73767

73768

73769

73770

73771

73772

73773

73774

73775

73776

73777

73778

73779

73780

Request Type	Current State						
	X	Q	R	H	W	E	T
Queue Batch Job Request	Q	e	e	e	e	e	e
Modify Batch Job Request	e	Q	R	H	W	e	T
Delete Batch Job Request	e	X	E	X	X	E	X
Batch Job Message Request	e	Q	R	H	W	E	T
Rerun Batch Job Request	e	e	Q	e	e	e	e
Signal Batch Job Request	e	e	R	H	W	e	e
Batch Job Status Request	e	Q	R	H	W	E	T
Batch Queue Status Request	X	Q	R	H	W	E	T
Server Status Request	X	Q	R	H	W	E	T
Select Batch Jobs Request	X	Q	R	H	W	E	T
Move Batch Job Request	e	Q	R	H	W	e	T
Hold Batch Job Request	e	H	R/H	H	H	e	T
Release Batch Job Request	e	Q	R	Q/W/H	W	e	T
Server Shutdown Request	X	Q	Q	H	W	E	T
Locate Batch Job Request	e	Q	R	H	W	E	T

73781

Legend

73782

X Nonexistent

73783

Q QUEUED

73784

R RUNNING

73785

H HELD

73786

W WAITING

73787

E EXITING

73788

T TRANSITING

73789

e Error

73790

A batch server that has an execution queue containing jobs is said to own the queue and manage the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in the execution queues owned by the batch server. The batch server shall schedule for execution those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling jobs is implementation-defined.

73795

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_ENVIRONMENT*, the value of which is the string *PBS_BATCH* encoded in the portable character set.

73796

73797

73798

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_QUEUE*, the value of which is the name of the execution queue of the batch job encoded in the portable character set.

73799

73800

73801

To rerun a batch job is to requeue a batch job that is currently executing and then kill the session leader of the executing job by sending a SIGKILL prior to completion; see [Section 3.2.3.11](#) (on page 2338). A batch server that reruns a batch job shall append the standard output and standard error files of the batch job to the corresponding files of the previous execution, if they exist, with appropriate annotation. If either file does not exist, that file shall be created as in

73802

73803

73804

73805

73806 normal execution.

73807 **Table 3-4** Results/Output Table

Request Type	Current State						
	X	Q	R	H	W	E	T
73810 <i>Queue Batch Job Request</i>	O	e	e	e	e	e	e
73811 <i>Modify Batch Job Request</i>	e	O	e	O	O	e	e
73812 <i>Delete Batch Job Request</i>	e	O	O	O	O	e	O
73813 <i>Batch Job Message Request</i>	e	e	O	e	e	e	e
73814 <i>Rerun Batch Job Request</i>	e	e	O	e	e	e	e
73815 <i>Signal Batch Job Request</i>	e	e	O	e	e	e	e
73816 <i>Batch Job Status Request</i>	e	O	O	O	O	O	O
73817 <i>Batch Queue Status Request</i>	O	O	O	O	O	O	O
73818 <i>Server Status Request</i>	O	O	O	O	O	O	O
73819 <i>Select Batch Job Request</i>	e	O	O	O	O	O	O
73820 <i>Move Batch Job Request</i>	e	O	O	O	O	e	e
73821 <i>Hold Batch Job Request</i>	e	O	O	O	O	e	e
73822 <i>Release Batch Job Request</i>	e	O	e	O	O	e	e
73823 <i>Server Shutdown Request</i>	O	O	e	O	O	e	e
73824 <i>Locate Batch Job Request</i>	e	O	O	O	O	O	O

73825 Legend

73826 O OK

73827 e Error message

73828 The execution of a batch job by a batch server shall be controlled by job, queue, and server
73829 attributes, as defined in this section.

73830 Account_Name Attribute

73831 Batch accounting is an optional feature of batch servers. If a batch server implements
73832 accounting, the statements in this section apply and the configuration variable
73833 POSIX2_PBS_ACCOUNTING shall be set to 1.

73834 A batch server that executes a batch job shall charge the account named in the *Account_Name*
73835 attribute of the batch job for resources consumed by the batch job.

73836 If the *Account_Name* attribute of the batch job is absent from the batch job attribute list or is
73837 altered while the batch job is in execution, the batch server action is implementation-defined.

73838 Checkpoint Attribute

73839 Batch checkpointing is an optional feature of batch servers. If a batch server implements
73840 checkpointing, the statements in this section apply and the configuration variable
73841 POSIX2_PBS_CHECKPOINT shall be set to 1.

73842 There are two attributes associated with the checkpointing feature: *Checkpoint* and
73843 *Minimum_Cpu_Interval*. *Checkpoint* is a batch job attribute, while *Minimum_Cpu_Interval* is a
73844 queue attribute. An implementation that does not support checkpointing shall support the
73845 *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute
73846 to other servers.

73847 The behavior of a batch server that executes a batch job for which the value of the *Checkpoint*

73848 attribute is CHECKPOINT_UNSPECIFIED is implementation-defined. A batch server that
 73849 executes a batch job for which the value of the *Checkpoint* attribute is NO_CHECKPOINT shall
 73850 not checkpoint the batch job.

73851 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
 73852 CHECKPOINT_AT_SHUTDOWN shall checkpoint the batch job only when the batch server
 73853 accepts a request to shut down during the time when the batch job is in the RUNNING state.

73854 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is
 73855 CHECKPOINT_AT_MIN_CPU_INTERVAL shall checkpoint the batch job at the interval
 73856 specified by the *Minimum_Cpu_Interval* attribute of the queue for which the batch job has been
 73857 selected. The *Minimum_Cpu_Interval* attribute shall be specified in units of CPU minutes.

73858 A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an
 73859 unsigned integer shall checkpoint the batch job at an interval that is the value of either the
 73860 *Checkpoint* attribute, or the *Minimum_Cpu_Interval* attribute of the queue for which the batch job
 73861 has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When
 73862 the *Minimum_Cpu_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall
 73863 write a warning message to the standard error stream of the batch job.

73864 **Error_Path Attribute**

73865 The *Error_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When
 73866 the *Join_Path* attribute of the batch job is set to the value FALSE and the *Keep_Files* attribute of
 73867 the batch job does not contain the value KEEP_STD_ERROR, a batch server that executes a batch
 73868 job shall perform one of the following actions:

- 73869 • Set the standard error stream of the session leader of the batch job to the path described by
 73870 the value of the *Error_Path* attribute of the batch job.
- 73871 • Buffer the standard error of the session leader of the batch job until completion of the batch
 73872 job, and when the batch job exits return the contents to the destination described by the
 73873 value of the *Error_Path* attribute of the batch job.

73874 Applications shall not rely on having access to the standard error of a batch job prior to the
 73875 completion of the batch job.

73876 When the *Error_Path* attribute does not specify a host name, then the batch server shall retain the
 73877 standard error of the batch job on the host of execution.

73878 When the *Error_Path* attribute does specify a host name and the *Keep_Files* attribute does not
 73879 contain the value KEEP_STD_ERROR, then the final destination of the standard error of the
 73880 batch job shall be on the host whose host name is specified.

73881 If the path indicated by the value of the *Error_Path* attribute of the batch job is a relative path, the
 73882 batch server shall expand the path relative to the home directory of the user on the host to which
 73883 the file is being returned.

73884 When the batch server buffers the standard error of the batch job and the file cannot be opened
 73885 for write upon completion of the batch job, then the server shall place the standard error in an
 73886 implementation-defined location and notify the user of the location via mail. It shall be possible
 73887 for the user to process this mail using the *mailx* utility.

73888 If a batch server that does not buffer the standard error cannot open the standard error path of
 73889 the batch job for write access, then the batch server shall abort the batch job.

73890 **Execution_Time Attribute**

73891 A batch server shall not execute a batch job before the time represented by the value of the
 73892 *Execution_Time* attribute of the batch job. The *Execution_Time* attribute is defined in seconds since
 73893 the Epoch.

73894 **Hold_Types Attribute**

73895 A batch server shall support the following hold types:

- 73896 s Can be set or released by a user with at least a privilege level of batch administrator
 73897 (SYSTEM).
- 73898 o Can be set or released by a user with at least a privilege level of batch operator
 73899 (OPERATOR).
- 73900 u Can be set or released by the user with at least a privilege level of user, where the user is
 73901 defined in the *Job_Owner* attribute (USER).
- 73902 n Indicates that none of the *Hold_Types* attributes are set (NO_HOLD).

73903 An implementation may define other hold types. Any additional hold types, how they are
 73904 specified, their internal representation, their behavior, and how they affect the behavior of other
 73905 utilities are implementation-defined.

73906 The value of the *Hold_Types* attribute shall be the union of the valid hold types ('s', 'o', 'u',
 73907 and any implementation-defined hold types), or 'n'.

73908 A batch server shall not execute a batch job if the *Hold_Types* attribute of the batch job has a
 73909 value other than NO_HOLD. If the *Hold_Types* attribute of the batch job has a value other than
 73910 NO_HOLD, the batch job shall be in the HELD state.

73911 **Job_Owner Attribute**

73912 The *Job_Owner* attribute consists of a pair of user name and host name values of the form:

73913 `username@hostname`

73914 A batch server that accepts a *Queue Batch Job Request* shall set the *Job_Owner* attribute to a string
 73915 that is the `username@hostname` of the user who submitted the job.

73916 **Join_Path Attribute**

73917 A batch server that executes a batch job for which the value of the *Join_Path* attribute is TRUE
 73918 shall ignore the value of the *Error_Path* attribute and merge the standard error of the batch job
 73919 with the standard output of the batch job.

73920 **Keep_Files Attribute**

73921 A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes
 73922 the value KEEP_STD_OUTPUT shall retain the standard output of the batch job on the host
 73923 where execution occurs. The standard output shall be retained in the home directory of the user
 73924 under whose user ID the batch job is executed and the filename shall be the default filename for
 73925 the standard output as defined under the `-o` option of the *qsub* utility. The *Output_Path* attribute
 73926 is not modified.

73927 A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes
 73928 the value KEEP_STD_ERROR shall retain the standard error of the batch job on the host where
 73929 execution occurs. The standard error shall be retained in the home directory of the user under
 73930 whose user ID the batch job is executed and the filename shall be the default filename for

73931 standard error as defined under the `-e` option of the `qsub` utility. The `Error_Path` attribute is not
73932 modified.

73933 A batch server that executes a batch job for which the value of the `Keep_Files` attribute includes
73934 values other than `KEEP_STD_OUTPUT` and `KEEP_STD_ERROR` shall retain these other files on
73935 the host where execution occurs. These files (with implementation-defined names) shall be
73936 retained in the home directory of the user under whose user identifier the batch job is executed.

73937 **Mail_Points and Mail_Users Attributes**

73938 A batch server that executes a batch job for which one of the values of the `Mail_Points` attribute is
73939 the value `MAIL_AT_BEGINNING` shall send a mail message to each user account listed in the
73940 `Mail_Users` attribute of the batch job.

73941 The mail message shall contain at least the batch job identifier, queue, and server at which the
73942 batch job currently resides, and the `Job_Owner` attribute.

73943 **Output_Path Attribute**

73944 The `Output_Path` attribute of a running job cannot be changed by a `Modify Batch Job Request`.
73945 When the `Keep_Files` attribute of the batch job does not contain the value `KEEP_STD_OUTPUT`, a
73946 batch server that executes a batch job shall either:

- 73947 • Set the standard output stream of the session leader of the batch job to the destination
73948 described by the value of the `Output_Path` attribute of the batch job.

73949 or:

- 73950 • Buffer the standard output of the session leader of the batch job until completion of the
73951 batch job, and when the batch job exits return the contents to the destination described by
73952 the value of the `Output_Path` attribute of the batch job.

73953 When the `Output_Path` attribute does not specify a host name, then the batch server shall retain
73954 the standard output of the batch job on the host of execution.

73955 When the `Keep_Files` attribute does not contain the value `KEEP_STD_OUTPUT` and the
73956 `Output_Path` attribute does specify a host name, then the final destination of the standard output
73957 of the batch job shall be on the host specified.

73958 If the path specified in the `Output_Path` attribute of the batch job is a relative path, the batch
73959 server shall expand the path relative to the home directory of the user on the host to which the
73960 file is being returned.

73961 Whether or not the batch server buffers the standard output of the batch job until completion of
73962 the batch job is implementation-defined. Applications shall not rely on having access to the
73963 standard output of a batch job prior to the completion of the batch job.

73964 When the batch server does buffer the standard output of the batch job and the file cannot be
73965 opened for write upon completion of the batch job, then the batch server shall place the standard
73966 output in an implementation-defined location and notify the user of the location via mail. It shall
73967 be possible for the user to process this mail using the `mailx` utility.

73968 If a batch server that does not buffer the standard output cannot open the standard output path
73969 of the batch job for write access, then the batch server shall abort the batch job.

73970 **Priority Attribute**

73971 A batch server implementation may choose to preferentially execute a batch job based on the
 73972 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is
 73973 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger
 73974 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

73975 **Rerunable Attribute**

73976 A batch job that began execution but did not complete, because the batch server either shut
 73977 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has
 73978 the value TRUE.

73979 If a batch job, which was requeued after beginning execution but prior to completion, has a valid
 73980 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted
 73981 from the last valid checkpoint.

73982 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*
 73983 attribute value of TRUE and was requeued after beginning execution but prior to completion,
 73984 the batch server shall place the batch job into execution at the beginning of the job.

73985 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after
 73986 beginning execution but prior to completion, and the batch job cannot be restarted from a
 73987 checkpoint, then the batch server shall abort the batch job.

73988 **Resource_List Attribute**

73989 A batch server that executes a batch job shall establish the resource limits of the session leader of
 73990 the batch job according to the values of the *Resource_List* attribute of the batch job. Resource
 73991 limits shall be enforced by an implementation-defined method.

73992 **Shell_Path_List Attribute**

73993 The *Shell_Path_List* job attribute consists of a list of pairs of pathname and host name values. The
 73994 host name component can be omitted, in which case the pathname serves as the default
 73995 pathname when a batch server cannot find the name of the host on which it is running in the list.

73996 A batch server that executes a batch job shall select, from the value of the *Shell_Path_List*
 73997 attribute of the batch job, a pathname where the shell to execute the batch job shall be found.
 73998 The batch server shall select the pathname, in order of preference, according to the following
 73999 methods:

- 74000 • Select the pathname that contains the name of the host on which the batch server is
 74001 running.
- 74002 • Select the pathname for which the host name has been omitted.
- 74003 • Select the pathname for the login shell of the user under which the batch job is to execute.

74004 If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

74005 If the value of the selected pathname from the *Shell_Path_List* attribute of the batch job
 74006 represents a partial path, the batch server shall expand the path relative to a path that is
 74007 implementation-defined.

74008 The batch server that executes the batch job shall execute the program that was selected from the
 74009 *Shell_Path_List* attribute of the batch job. The batch server shall pass the path to the script of the
 74010 batch job as the first argument to the shell program.

74011 **User_List Attribute**

74012 The *User_List* job attribute consists of a list of pairs of user name and host name values. The host
 74013 name component can be omitted, in which case the user name serves as a default when a batch
 74014 server cannot find the name of the host on which it is running in the list.

74015 A batch server that executes a batch job shall select, from the value of the *User_List* attribute of
 74016 the batch job, a user name under which to create the session leader. The server shall select the
 74017 user name, in order of preference, according to the following methods:

- 74018 • Select the user name of a value that contains the name of the host on which the batch
 74019 server executes.
- 74020 • Select the user name of a value for which the host name has been omitted.
- 74021 • Select the user name from the *Job_Owner* attribute of the batch job.

74022 **Variable_List Attribute**

74023 A batch server that executes a batch job shall create, in the environment of the session leader of
 74024 the batch job, each environment variable listed in the *Variable_List* attribute of the batch job, and
 74025 set the value of each such environment variable to that of the corresponding variable in the
 74026 variable list.

74027 3.2.2.2 *Batch Job Routing*

74028 To route a batch job is to select a queue from a list and move the batch job to that queue.

74029 A batch server that has routing queues, which have been started, shall route the jobs in the
 74030 routing queues owned by the batch server. A batch server may delay the routing of a batch job.
 74031 The algorithm for selecting a batch job and the queue to which it will be routed is
 74032 implementation-defined.

74033 When a routing queue has multiple possible destinations specified, then the precedence of the
 74034 destinations is implementation-defined.

74035 A batch server that routes a batch job to a queue at another server shall move the batch job into
 74036 the target queue with a *Queue Batch Job Request*.

74037 If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the
 74038 batch job or abort the batch job. A batch server that retries failed routings shall provide a means
 74039 for the batch administrator to specify the number of retries and the minimum period of time
 74040 between retries. The means by which an administrator specifies the number of retries and the
 74041 delay between retries is implementation-defined. When the number of retries specified by the
 74042 batch administrator has been exhausted, the batch server shall abort the batch job and perform
 74043 the functions of *Batch Job Exit*; see [Section 3.2.2.3](#).

74044 3.2.2.3 *Batch Job Exit*

74045 For each job in the EXITING state, the batch server that exited the batch job shall perform the
 74046 following deferred services in the order specified:

- 74047 1. If buffering standard error, move that file into the location specified by the *Error_Path*
 74048 attribute of the batch job.
- 74049 2. If buffering standard output, move that file into the location specified by the *Output_Path*
 74050 attribute of the batch job.

- 74051 3. If the *Mail_Points* attribute of the batch job includes MAIL_AT_EXIT, send mail to the
 74052 users listed in the *Mail_Users* attribute of the batch job. The mail message shall contain at
 74053 least the batch job identifier, queue, and server at which the batch job currently resides,
 74054 and the *Job_Owner* attribute.
- 74055 4. Remove the batch job from the queue.

74056 If a batch server that buffers the standard error output cannot return the standard error file to
 74057 the standard error path at the time the batch job exits, the batch server shall do one of the
 74058 following:

- 74059 • Mail the standard error file to the batch job owner.
- 74060 • Save the standard error file and mail the location and name of the file where the standard
 74061 error is stored to the batch job owner.
- 74062 • Save the standard error file and notify the user by other implementation-defined means.

74063 If a batch server that buffers the standard output cannot return the standard output file to the
 74064 standard output path at the time the batch job exits, the batch server shall do one of the
 74065 following:

- 74066 • Mail the standard output file to the batch job owner.
- 74067 • Save the standard output file and mail the location and name of the file where the standard
 74068 output is stored to the batch job owner.
- 74069 • Save the standard output file and notify the user by other implementation-defined means.

74070 At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

74071 3.2.2.4 *Batch Server Restart*

74072 A batch server that has been either shutdown or terminated abnormally, and has returned to
 74073 operation, is said to have “restarted”.

74074 Upon restarting, a batch server shall requeue those jobs managed by the batch server that were
 74075 in the RUNNING state at the time the batch server shut down and for which the *Rerunable*
 74076 attribute of the batch job has the value TRUE.

74077 Queues are defined to be non-volatile. A batch server shall store the content of queues that it
 74078 controls in such a way that server and system shutdowns do not erase the content of the queues.

74079 3.2.2.5 *Batch Job Abort*

74080 A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

74081 A batch server that aborts a batch job shall perform the following services:

- 74082 • Delete the batch job from the queue in which it resides.
- 74083 • If the *Mail_Points* attribute of the batch job includes the value MAIL_AT_ABORT, send
 74084 mail to the users listed in the value of the *Mail_Users* attribute of the job. The mail message
 74085 shall contain at least the batch job identifier, queue, and server at which the batch job
 74086 currently resides, the *Job_Owner* attribute, and the reason for the abort.
- 74087 • If the batch job was in the RUNNING state, terminate the session leader of the executing
 74088 job by sending the session leader a SIGKILL, place the batch job in the EXITING state, and
 74089 perform the actions of *Batch Job Exit*.

3.2.3 Requested Batch Services

This section describes the services provided by batch servers in response to requests from clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type (deferred or not) and whether it is an optional function.

Table 3-5 Batch Services Summary

Batch Service	Deferred	Optional
<i>Batch Job Execution</i>	Yes	No
<i>Batch Job Routing</i>	Yes	No
<i>Batch Job Exit</i>	Yes	No
<i>Batch Server Restart</i>	Yes	No
<i>Batch Job Abort</i>	Yes	No
<i>Delete Batch Job Request</i>	No	No
<i>Hold Batch Job Request</i>	No	No
<i>Batch Job Message Request</i>	No	Yes
<i>Batch Job Status Request</i>	No	No
<i>Locate Batch Job Request</i>	No	Yes
<i>Modify Batch Job Request</i>	No	No
<i>Move Batch Job Request</i>	No	No
<i>Queue Batch Job Request</i>	No	No
<i>Batch Queue Status Request</i>	No	No
<i>Release Batch Job Request</i>	No	No
<i>Rerun Batch Job Request</i>	No	No
<i>Select Batch Jobs Request</i>	No	No
<i>Server Shutdown Request</i>	No	No
<i>Server Status Request</i>	No	No
<i>Signal Batch Job Request</i>	No	No
<i>Track Batch Job Request</i>	No	Yes

If a request is rejected because the batch client is not authorized to perform the action, the batch server shall return the same status as when the batch job does not exist.

3.2.3.1 Delete Batch Job Request

A batch job is defined to have been deleted when it has been removed from the queue in which it resides and not instantiated in another queue. A client requests that the server that manages a batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to delete the designated job.
- The designated job is not managed by the batch server.
- The designated job is in a state inconsistent with the delete request.

A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server requested to delete a batch job shall delete the batch job if the batch job exists and is not in the EXITING state.

A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the

74133 session leader of the batch job. It is implementation-defined whether additional signals are sent
74134 to the session leader of the job prior to sending the SIGKILL signal.

74135 A batch server that deletes a batch job in the RUNNING state shall place the batch job in the
74136 EXITING state after it has killed the session leader of the batch job and shall perform the actions
74137 of *Batch Job Exit*.

74138 3.2.3.2 *Hold Batch Job Request*

74139 A batch client can request that the batch server add one or more holds to a batch job. Such a
74140 request is called a *Hold Batch Job Request*.

74141 A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- 74142 • The batch server does not support one or more of the requested holds to be added to the
74143 batch job.
- 74144 • The user of the batch client is not authorized to add one or more of the requested holds to
74145 the batch job.
- 74146 • The batch server does not manage the specified job.
- 74147 • The designated job is in the EXITING state.

74148 A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The
74149 method used to determine whether the user of a client is authorized to perform the requested
74150 action is implementation-defined.

74151 A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall
74152 place a hold on the batch job. The effects, if any, the hold will have on a batch job in the
74153 RUNNING state are implementation-defined.

74154 A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold*
74155 *Batch Job Request*, that is not already present, to the value of the *Hold_Types* attribute of the batch
74156 job.

74157 3.2.3.3 *Batch Job Message Request*

74158 *Batch Job Message Request* is an optional feature of batch servers. If an implementation supports
74159 *Batch Job Message Request*, the statements in this section apply and the configuration variable
74160 POSIX2_PBS_MESSAGE shall be set to 1.

74161 A batch client can request that a batch server write a message into certain output files of a batch
74162 job. Such a request is called a *Batch Job Message Request*.

74163 A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- 74164 • The batch server does not support sending messages to jobs.
- 74165 • The user of the batch client is not authorized to post a message to the designated job.
- 74166 • The designated job does not exist on the batch server.
- 74167 • The designated job is not in the RUNNING state.

74168 A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons.
74169 The method used to determine whether the user of a client is authorized to perform the
74170 requested action is implementation-defined.

74171 A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch
74172 client into the files indicated by the batch client.

74173 3.2.3.4 *Batch Job Status Request*

74174 A batch client can request that a batch server respond with the status and attributes of a batch
74175 job. Such a request is called a *Batch Job Status Request*.

74176 A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- 74177 • The user of the batch client is not authorized to query the status of the designated job.
- 74178 • The designated job is not managed by the batch server.

74179 A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons.
74180 The method used to determine whether the user of a client is authorized to perform the
74181 requested action is implementation-defined.

74182 A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the
74183 batch client.

74184 A batch server may return other information in response to a *Batch Job Status Request*.

74185 3.2.3.5 *Locate Batch Job Request*

74186 *Locate Batch Job Request* is an optional feature of batch servers. If an implementation supports
74187 *Locate Batch Job Request*, the statements in this section apply and the configuration variable
74188 POSIX2_PBS_LOCATE shall be set to 1.

74189 A batch client can ask a batch server to respond with the location of a batch job that was created
74190 by the batch server. Such a request is called a *Locate Batch Job Request*.

74191 A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to
74192 the batch client.

74193 A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that
74194 server.

74195 A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by
74196 that server; that is, for a batch job that is not in a queue owned by that server.

74197 A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

74198 3.2.3.6 *Modify Batch Job Request*

74199 Batch clients modify (alter) the attributes of a batch job by making a request to the server that
74200 manages the batch job. Such a request is called a *Modify Batch Job Request*.

74201 A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- 74202 • The user of the batch client is not authorized to make the requested modification to the
74203 batch job.
- 74204 • The designated job is not managed by the batch server.
- 74205 • The requested modification is inconsistent with the state of the batch job.
- 74206 • An unrecognized resource is requested for a batch job in an execution queue.

74207 A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons.
74208 The method used to determine whether the user of a client is authorized to perform the
74209 requested action is implementation-defined.

74210 A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of
74211 the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the
74212 attributes of the batch job.

74213 If the servicing by a batch server of an otherwise valid request would result in no change, then
74214 the batch server shall indicate successful completion of the request.

74215 3.2.3.7 *Move Batch Job Request*

74216 A batch client can request that a batch server move a batch job to another destination. Such a
74217 request is called a *Move Batch Job Request*.

74218 A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- 74219 • The user of the batch client is not authorized to remove the designated job from the queue
74220 in which the batch job resides.
- 74221 • The user of the batch client is not authorized to move the designated job to the destination.
- 74222 • The designated job is not managed by the batch server.
- 74223 • The designated job is in the EXITING state.
- 74224 • The destination is inaccessible.

74225 A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The
74226 method used to determine whether the user of a client is authorized to perform the requested
74227 action is implementation-defined.

74228 A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- 74229 • Queue the designated job at the destination.
- 74230 • Remove the designated job from the queue in which the batch job resides.

74231 If the destination resides on another batch server, the batch server shall queue the batch job at
74232 the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job*
74233 *Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job*
74234 *Request* succeeds, the batch server shall remove the batch job from its queue.

74235 The batch server shall not modify any attributes of the batch job.

74236 3.2.3.8 *Queue Batch Job Request*

74237 A batch queue is controlled by one and only one batch server. A batch server is said to own the
74238 queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a
74239 request is called a *Queue Batch Job Request*.

74240 A batch server requested to queue a batch job for which the queue is not specified shall select an
74241 implementation-defined queue for the batch job. Such a queue is called the “default queue” of
74242 the batch server. The implementation shall provide the means for a batch administrator to
74243 specify the default queue. The queue, whether specified or defaulted, is called the “target
74244 queue”.

74245 A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- 74246 • The client is not authorized to create a batch job in the target queue.
- 74247 • The request specifies a queue that does not exist on the batch server.
- 74248 • The target queue is an execution queue and the batch server cannot satisfy a resource
74249 requirement of the batch job.
- 74250 • The target queue is an execution queue and an unrecognized resource is requested.

- 74251 • The target queue is an execution queue, the batch server does not support checkpointing,
74252 and the value of the *Checkpoint* attribute of the batch job is not NO_CHECKPOINT.
- 74253 • The job requires access to a user identifier that the batch client is not authorized to access.

74254 A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.

74255 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
74256 PBS_O_QUEUE value is missing from the value of the *Variable_List* attribute of the batch job
74257 shall add that variable to the list and set the value to the name of the target queue. Once set, no
74258 server shall change the value of PBS_O_QUEUE, even if the batch job is moved to another
74259 queue.

74260 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_JOBID
74261 value is missing from the value of the *Variable_List* attribute shall add that variable to the list and
74262 set the value to the batch job identifier assigned by the server in the format:

74263 `sequence_number.server`

74264 A batch server that accepts a *Queue Batch Job Request* for a batch job for which the
74265 PBS_JOBNAME value is missing from the value of the *Variable_List* attribute of the batch job
74266 shall add that variable to the list and set the value to the *Job_Name* attribute of the batch job.

74267 3.2.3.9 *Batch Queue Status Request*

74268 A batch client can request that a batch server respond with the status and attributes of a queue.
74269 Such a request is called a *Batch Queue Status Request*.

74270 A batch server shall reject a *Batch Queue Status Request* if any of the following statements are
74271 true:

- 74272 • The user of the batch client is not authorized to query the status of the designated queue.
- 74273 • The designated queue does not exist on the batch server.

74274 A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons.
74275 The method used to determine whether the user of a client is authorized to perform the
74276 requested action is implementation-defined.

74277 A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to
74278 the batch client.

74279 3.2.3.10 *Release Batch Job Request*

74280 A batch client can request that the server remove one or more holds from a batch job. Such a
74281 request is called a *Release Batch Job Request*.

74282 A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:

- 74283 • The user of the batch client is not authorized to remove one or more of the requested holds
74284 from the batch job.
- 74285 • The batch server does not manage the specified job.

74286 A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons.
74287 The method used to determine whether the user of a client is authorized to perform the
74288 requested action is implementation-defined.

74289 A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the
74290 *Release Batch Job Request*, that is present, from the value of the *Hold_Types* attribute of the batch
74291 job.

74292 3.2.3.11 *Rerun Batch Job Request*

74293 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible
74294 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request
74295 is called *Rerun Batch Job Request*.

74296 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

- 74297 • The user of the batch client is not authorized to rerun the designated job.
- 74298 • The *Rerunable* attribute of the designated job has the value FALSE.
- 74299 • The designated job is not in the RUNNING state.
- 74300 • The batch server does not manage the designated job.

74301 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons.
74302 The method used to determine whether the user of a client is authorized to perform the
74303 requested action is implementation-defined.

74304 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the
74305 batch job.

74306 A batch server that accepts a request to rerun a batch job shall perform the following services:

- 74307 • Requeue the batch job in the execution queue in which it was executing.
- 74308 • Send a SIGKILL signal to the process group of the session leader of the batch job.

74309 An implementation may indicate to the batch job owner that the batch job has been rerun.
74310 Whether and how the batch job owner is notified that a batch job is rerun is implementation-
74311 defined.

74312 A batch server that reruns a batch job may send other implementation-defined signals to the
74313 session leader of the batch job prior to sending the SIGKILL signal.

74314 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be
74315 selected for execution before other jobs is implementation-defined.

74316 3.2.3.12 *Select Batch Jobs Request*

74317 A batch client can request from a batch server a list of jobs managed by that server that match a
74318 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs
74319 managed by the batch server that receives the request are candidates for selection.

74320 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job
74321 identifiers that correspond to jobs that meet the selection criteria.

74322 If the batch client is not authorized to query the status of a batch job, the batch server shall not
74323 select the batch job.

74324 3.2.3.13 *Server Shutdown Request*

74325 A batch server is defined to have shut down when it does not respond to requests from clients
74326 and does not perform deferred services for jobs. A batch client can request that a batch server
74327 shut down. Such a request is called a *Server Shutdown Request*.

74328 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut
74329 down the batch server. The method used to determine whether the user of a client is authorized
74330 to perform the requested action is implementation-defined.

74331 A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons.
74332 The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

74333 At server shutdown, a batch server shall do, in order of preference, one of the following:

- 74334 • If checkpointing is implemented and the batch job is checkpointable, then checkpoint the
- 74335 batch job and requeue it.
- 74336 • If the batch job is rerunnable, then requeue the batch job to be rerun (restarted from the
- 74337 beginning).
- 74338 • Abort the batch job.

74339 3.2.3.14 *Server Status Request*

74340 A batch client can request that a batch server respond with the status and attributes of the batch
74341 server. Such a request is called a *Server Status Request*.

74342 A batch server shall reject a *Server Status Request* if the following statement is true:

- 74343 • The user of the batch client is not authorized to query the status of the designated server.

74344 A batch server may reject a *Server Status Request* for other implementation-defined reasons. The
74345 method used to determine whether the user of a client is authorized to perform the requested
74346 action is implementation-defined.

74347 A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch
74348 client.

74349 3.2.3.15 *Signal Batch Job Request*

74350 A batch client can request that a batch server signal the session leader of a batch job. Such a
74351 request is called a *Signal Batch Job Request*.

74352 A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- 74353 • The user of the batch client is not authorized to signal the batch job.
- 74354 • The job is not in the RUNNING state.
- 74355 • The batch server does not manage the designated job.
- 74356 • The requested signal is not supported by the implementation.

74357 A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons.
74358 The method used to determine whether the user of a client is authorized to perform the
74359 requested action is implementation-defined.

74360 A batch server that accepts a request to signal a batch job shall send the signal requested by the
74361 batch client to the process group of the session leader of the batch job.

74362 3.2.3.16 *Track Batch Job Request*

74363 *Track Batch Job Request* is an optional feature of batch servers. If an implementation supports
74364 *Track Batch Job Request*, the statements in this section apply and the configuration variable
74365 POSIX2_PBS_TRACK shall be set to 1.

74366 *Track Batch Job Request* provides a method for tracking the current location of a batch job. Clients
74367 may use the tracking information to determine the batch server that should receive a batch
74368 server request.

74369 If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a
74370 batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that
74371 created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that
74372 created the job.

74373 If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also
 74374 be sent to other servers as a backup to the primary server. The method by which backup servers
 74375 are specified is implementation-defined.

74376 If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*,
 74377 then the batch server shall record the current location of the batch job as contained in the
 74378 request.

74379 3.3 Common Behavior for Batch Environment Utilities

74380 3.3.1 Batch Job Identifier

74381 A utility shall recognize *job_identifiers* of the format:

74382 [*sequence_number*] [. *server_name*] [@*server*]

74383 where:

74384 *sequence_number* An integer that, when combined with *server_name*, provides a batch job
 74385 identifier that is unique within the batch system.

74386 *server_name* The name of the batch server to which the batch job was originally submitted.

74387 *server* The name of the batch server that is currently managing the batch job.

74388 If the application omits the batch *server_name* portion of a batch job identifier, a utility shall use
 74389 the name of a default batch server.

74390 If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- 74391 • The batch server indicated by *server_name*, if present
- 74392 • The name of the default batch server
- 74393 • The name of the batch server that is currently managing the batch job

74394 If only @*server* is specified, then the status of all jobs owned by the user on the requested server
 74395 is listed.

74396 The means by which a utility determines the default batch server is implementation-defined.

74397 If the application presents the batch *server* portion of a batch job identifier to a utility, the utility
 74398 shall send the request to the specified server.

74399 A strictly conforming application shall use the syntax described for the job identifier. Whenever
 74400 a batch job identifier is specified whose syntax is not recognized by an implementation, then a
 74401 message for each error that occurs shall be written to standard error and the utility shall exit
 74402 with an exit status greater than zero.

74403 When a batch job identifier is supplied as an argument to a batch utility and the *server_name*
 74404 portion of the batch job identifier is omitted, then the utility shall use the name of the default
 74405 batch server.

74406 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
 74407 portion of the batch job identifier is omitted, then the utility shall use either:

- 74408 • The name of the default batch server

74409 or:

- 74410
- The name of the batch server that is currently managing the batch job

74411 When a batch job identifier is supplied as an argument to a batch utility and the batch *server*
74412 portion of the batch job identifier is specified, then the utility shall send the required *Batch Server*
74413 *Request* to the specified server.

74414 3.3.2 Destination

74415 The utility shall recognize a *destination* of the format:

74416 [*queue*] [@*server*]

74417 where:

74418 *queue* The name of a valid execution or routing queue at the batch server denoted by
74419 @*server*, defined as a string of up to 15 alphanumeric characters in the portable
74420 character set (see XBD Section 6.1, on page 111) where the first character is
74421 alphabetic.

74422 *server* The name of a batch server, defined as a string of alphanumeric characters in the
74423 portable character set.

74424 If the application omits the batch *server* portion of a destination, then the utility shall use either:

- The name of the default batch server

74426 or:

- The name of the batch server that is currently managing the batch job

74428 The means by which a utility determines the default batch server is implementation-defined.

74429 If the application omits the *queue* portion of a destination, then the utility shall use the name of
74430 the default queue at the batch server chosen. The means by which a batch server determines its
74431 default queue is implementation-defined. If a destination is specified in the *queue@server* form,
74432 then the utility shall use the specified queue at the specified server.

74433 A strictly conforming application shall use the syntax described for a destination. Whenever a
74434 destination is specified whose syntax is not recognized by an implementation, then a message
74435 shall be written to standard error and the utility shall exit with an exit status greater than zero.

74436 3.3.3 Multiple Keyword-Value Pairs

74437 For each option that can have multiple keyword-value pair arguments, the following rules shall
74438 apply. Examples of options that can have list-oriented option-arguments are `-u value@keyword`
74439 and `-l keyword=value`.

1. If a batch utility is presented with a list-oriented option-argument for which a keyword
74440 has a corresponding value that begins with a single or double quote, then the utility shall
74441 stop interpreting the input stream for delimiters until a second single or double quote,
74442 respectively, is encountered. This feature allows some flexibility for a comma (',') or
74443 equals sign ('=') to be part of the value string for a particular keyword; for example:

74445 `keywd1='val1, val2', keywd2="val3, val4"`

74446 **Note:** This may require the user to escape the quotes as in the following command:

74447 `foo -xkeywd1='val1, val2\ ', keywd2=\"val3, val4\"`

2. If a batch server is presented with a list-oriented attribute that has a keyword that was
74448 encountered earlier in the list, then the later entry for that keyword shall replace the
74449 earlier entry.
74450

- 74451
- 74452
- 74453
- 74454
- 74455
- 74456
- 74457
- 74458
- 74459
- 74460
- 74461
- 74462
- 74463
- 74464
- 74465
- 74466
- 74467
- 74468
- 74469
- 74470
- 74471
- 74472
- 74473
- 74474
- 74475
3. If a batch server is presented with a list-oriented attribute that has a keyword without any corresponding value of the form *keyword=* or *@keyword* and the same keyword was encountered earlier in the list, then the prior entry for that keyword shall be ignored by the batch server.
 4. If a batch utility is expecting a list-oriented option-argument entry of the form *keyword=value*, but is presented with an entry of the form *keyword* without any corresponding *value*, then the entry shall be treated as though a default value of NULL was assigned (that is, *keyword=NULL*) for entry parsing purposes. The utility shall include only the keyword, not the NULL value, in the associated job attribute.
 5. If a batch utility is expecting a list-oriented option-argument entry of the form *value@keyword*, but is presented with an entry of the form *value* without any corresponding *keyword*, then the entry shall be treated as though a keyword of NULL was assigned (that is, *value@NULL*) for entry parsing purposes. The utility shall include only the value, not the NULL keyword, in the associated job attribute.
 6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the same keyword, interpreting the keywords, in order, with the last value encountered taking precedence over prior instances of the same keyword. This rule allows, but does not require, a batch utility to preprocess the attribute to remove duplicate keywords.
 7. If a batch utility is presented with multiple list-oriented option-arguments on the command line or in script directives, or both, for a single option, then the utility shall concatenate, in order, any command line keyword and value pairs to the end of any directive keyword and value pairs separated by a single comma to produce a single string that is an equivalent, valid option-argument. The resulting string shall be assigned to the associated attribute of the batch job (after optionally removing duplicate entries as described in item 6).

74476

Chapter 4

74477

Utilities

74478

This chapter contains the definitions of the utilities, as follows:

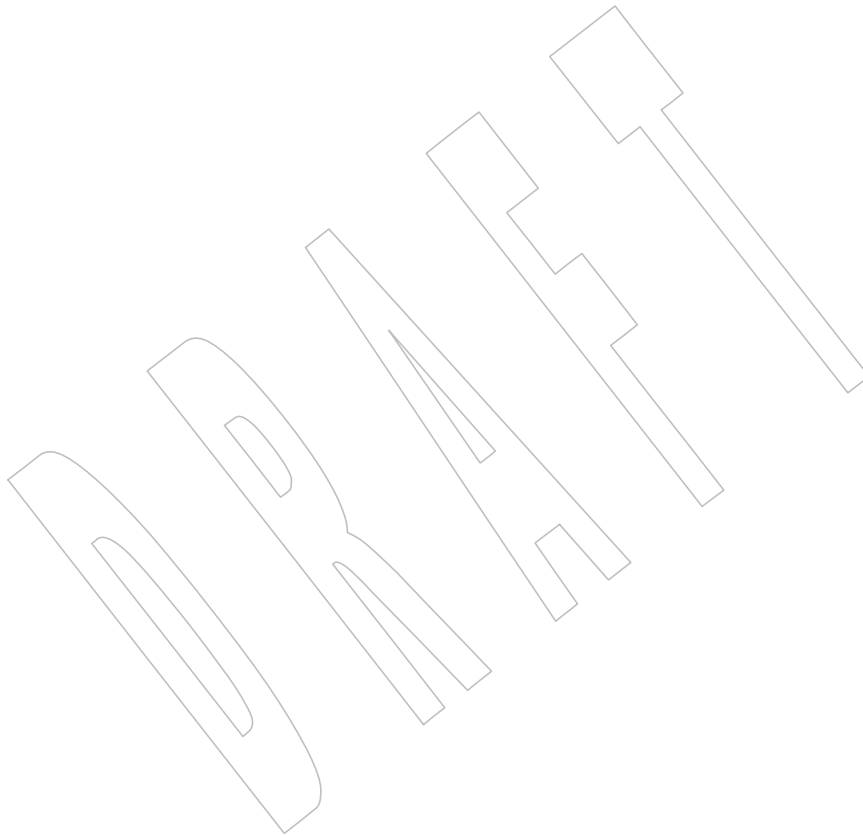
74479

- Mandatory utilities that are present on every conformant system

74480

- Optional utilities that are present only on systems supporting the associated option; see [Section 1.7.1](#) (on page 7) for information on the options in this volume of POSIX.1-200x

74481



74482 **NAME**74483 admin — create and administer SCCS files (**DEVELOPMENT**)74484 **SYNOPSIS**

```

74485 XSI admin -i[name] [-n] [-a login] [-d flag] [-e login] [-f flag]
74486         [-m mrlist] [-r rel] [-t[name] [-y[comment]]] newfile
74487
74487 admin -n [-a login] [-d flag] [-e login] [-f flag] [-m mrlist]
74488         [-t[name]] [-y[comment]] newfile...
74489
74489 admin [-a login] [-d flag] [-m mrlist] [-r rel] [-t[name]] file...
74490
74490 admin -h file...
74491
74491 admin -z file...

```

74492 **DESCRIPTION**

74493 The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named
 74494 file does not exist, it shall be created, and its parameters shall be initialized according to the
 74495 specified options. Parameters not initialized by an option shall be assigned a default value. If a
 74496 named file does exist, parameters corresponding to specified options shall be changed, and other
 74497 parameters shall be left as is.

74498 All SCCS filenames supplied by the application shall be of the form *s.filename*. New SCCS files
 74499 shall be given read-only permission mode. Write permission in the parent directory is required
 74500 to create a file. All writing done by *admin* shall be to a temporary *x-file*, named *x.filename* (see *get*)
 74501 created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode
 74502 as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file
 74503 shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This
 74504 ensures that changes are made to the SCCS file only if no errors occur.

74505 The *admin* utility shall also use a transient lock file (named *z.filename*), which is used to prevent
 74506 simultaneous updates to the SCCS file; see *get*.

74507 **OPTIONS**

74508 The *admin* utility shall conform to XBD [Section 12.2](#) (on page 201), except that the *-i*, *-t*, and *-y*
 74509 options have optional option-arguments. These optional option-arguments shall not be
 74510 presented as separate arguments. The following options are supported:

74511 **-n** Create a new SCCS file. When *-n* is used without *-i*, the SCCS file shall be created
 74512 with control information but without any file data.

74513 **-i[name]** Specify the *name* of a file from which the text for a new SCCS file shall be taken.
 74514 The text constitutes the first delta of the file (see the *-r* option for the delta
 74515 numbering scheme). If the *-i* option is used, but the *name* option-argument is
 74516 omitted, the text shall be obtained by reading the standard input. If this option is
 74517 omitted, the SCCS file shall be created with control information but without any
 74518 file data. The *-i* option implies the *-n* option.

74519 **-r SID** Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that
 74520 is, the branch and sequence numbers shall be zero or missing. The level number is
 74521 optional, and defaults to 1.

74522 **-t[name]** Specify the *name* of a file from which descriptive text for the SCCS file shall be
 74523 taken. In the case of existing SCCS files (neither *-i* nor *-n* is specified):

- 74524 • A **-t** option without a *name* option-argument shall cause the removal of
74525 descriptive text (if any) currently in the SCCS file.
- 74526 • A **-t** option with a *name* option-argument shall cause the text (if any) in the
74527 named file to replace the descriptive text (if any) currently in the SCCS file.
- 74528 **-f flag** Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file.
74529 Several **-f** options may be supplied on a single *admin* command line.
74530 Implementations shall recognize the following flags and associated values:
- 74531 **b** Allow use of the **-b** option on a *get* command to create branch deltas.
- 74532 **cceil** Specify the highest release (that is, ceiling), a number less than or equal to
74533 9999, which may be retrieved by a *get* command for editing. The default
74534 value for an unspecified **c** flag shall be 9999.
- 74535 **ffloor** Specify the lowest release (that is, floor), a number greater than 0 but less
74536 than 9999, which may be retrieved by a *get* command for editing. The
74537 default value for an unspecified **f** flag shall be 1.
- 74538 **dSID** Specify the default delta number (SID) to be used by a *get* command.
- 74539 **istr** Treat the “No ID keywords” message issued by *get* or *delta* as a fatal error.
74540 In the absence of this flag, the message is only a warning. The message is
74541 issued if no SCCS identification keywords (see *get*) are found in the text
74542 retrieved or stored in the SCCS file. If a value is supplied, the application
74543 shall ensure that the keywords exactly match the given string; however,
74544 the string shall contain a keyword, and no embedded <newline>s.
- 74545 **j** Allow concurrent *get* commands for editing on the same SID of an SCCS
74546 file. This allows multiple concurrent updates to the same version of the
74547 SCCS file.
- 74548 **l|list** Specify a *list* of releases to which deltas can no longer be made (that is, *get*
74549 **-e** against one of these locked releases fails). Conforming applications
74550 shall use the following syntax to specify a *list*. Implementations may
74551 accept additional forms as an extension:
- 74552 <list> ::= a | <range-list>
74553 <range-list> ::= <range> | <range-list>, <range>
74554 <range> ::= <SID>
- 74555 The character *a* in the *list* shall be equivalent to specifying all releases for
74556 the named SCCS file. The non-terminal <SID> in range shall be the delta
74557 number of an existing delta associated with the SCCS file.
- 74558 **n** Cause *delta* to create a null delta in each of those releases (if any) being
74559 skipped when a delta is made in a new release (for example, in making
74560 delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas
74561 shall serve as anchor points so that branch deltas may later be created
74562 from them. The absence of this flag shall cause skipped releases to be
74563 nonexistent in the SCCS file, preventing branch deltas from being created
74564 from them in the future. During the initial creation of an SCCS file, the **n**
74565 flag may be ignored; that is, if the **-r** option is used to set the release
74566 number of the initial SID to a value greater than 1, null deltas need not be
74567 created for the “skipped” releases.
- 74568 **qtext** Substitute user-definable *text* for all occurrences of the %Q% keyword in
74569 the SCCS file text retrieved by *get*.

74570	mmod	Specify the module name of the SCCS file substituted for all occurrences of the %M% keyword in the SCCS file text retrieved by <i>get</i> . If the m flag is not specified, the value assigned shall be the name of the SCCS file with the leading ' . ' removed.
74571		
74572		
74573		
74574	ttype	Specify the <i>type</i> of module in the SCCS file substituted for all occurrences of the %Y% keyword in the SCCS file text retrieved by <i>get</i> .
74575		
74576	vpgm	Cause <i>delta</i> to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the m option is also used even if its value is null.)
74577		
74578		
74579		
74580		
74581	-d flag	Remove (delete) the specified <i>flag</i> from an SCCS file. Several -d options may be supplied on a single <i>admin</i> command. See the -f option for allowable <i>flag</i> names. (The l ist flag gives a <i>list</i> of releases to be unlocked. See the -f option for further description of the l flag and the syntax of a <i>list</i> .)
74582		
74583		
74584		
74585	-a login	Specify a <i>login</i> name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all <i>login</i> names common to that group ID. Several -a options may be used on a single <i>admin</i> command line. As many <i>logins</i> , or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If <i>login</i> or group ID is preceded by a '!', the users so specified shall be denied permission to make deltas.
74586		
74587		
74588		
74589		
74590		
74591		
74592	-e login	Specify a <i>login</i> name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all <i>login</i> names common to that group ID. Several -e options may be used on a single <i>admin</i> command line.
74593		
74594		
74595		
74596	-y[comment]	Insert the <i>comment</i> text into the SCCS file as a comment for the initial delta in a manner identical to that of <i>delta</i> . In the POSIX locale, omission of the -y option shall result in a default comment line being inserted in the form:
74597		
74598		
74599		"date and time created %s %s by %s", <date>, <time>, <login>
74600		where <date> is expressed in the format of the <i>date</i> utility's %Y/%m/%d conversion specification, <time> in the format of the <i>date</i> utility's %T conversion specification format, and <login> is the login name of the user creating the file.
74601		
74602		
74603	-m mrlist	Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to <i>delta</i> . The application shall ensure that the v flag is set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the v flag is not set or MR validation fails.
74604		
74605		
74606		
74607		
74608	-h	Check the structure of the SCCS file and compare the newly computed checksum with the checksum that is stored in the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.
74609		
74610		
74611		
74612	-z	Recompute the SCCS file checksum and store it in the first line of the SCCS file (see the -h option above). Note that use of this option on a truly corrupted file may prevent future detection of the corruption.
74613		
74614		

74615 **OPERANDS**

74616 The following operands shall be supported:

74617 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin*
 74618 utility shall behave as though each file in the directory were specified as a named
 74619 file, except that non-SCCS files (last component of the pathname does not begin
 74620 with **s**.) and unreadable files shall be silently ignored.

74621 *newfile* A pathname of an SCCS file to be created.

74622 If exactly one *file* or *newfile* operand appears, and it is **'-'**, the standard input shall be read; each
 74623 line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-
 74624 SCCS files and unreadable files shall be silently ignored.

74625 **STDIN**

74626 The standard input shall be a text file used only if **-i** is specified without an option-argument or
 74627 if a *file* or *newfile* operand is specified as **'-'**. If the first character of any standard input line is
 74628 <SOH> in the POSIX locale, the results are unspecified.

74629 **INPUT FILES**

74630 The existing SCCS files shall be text files of an unspecified format.

74631 The application shall ensure that the file named by the **-i** option's *name* option-argument shall
 74632 be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results
 74633 are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the
 74634 header for this file shall be 99 999 for this delta.

74635 **ENVIRONMENT VARIABLES**74636 The following environment variables shall affect the execution of *admin*:

74637 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 74638 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 74639 variables used to determine the values of locale categories.)

74640 **LC_ALL** If set to a non-empty string value, override the values of all the other
 74641 internationalization variables.

74642 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 74643 characters (for example, single-byte as opposed to multi-byte characters in
 74644 arguments and input files).

74645 **LC_MESSAGES**
 74646 Determine the locale that should be used to affect the format and contents of
 74647 diagnostic messages written to standard error and the contents of the default **-y**
 74648 comment.

74649 **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

74650 **ASYNCHRONOUS EVENTS**

74651 Default.

74652 **STDOUT**

74653 Not used.

74654 **STDERR**

74655 The standard error shall be used only for diagnostic messages.

74656 **OUTPUT FILES**

74657 Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a
 74658 locking *z-file*, as described in *get* (on page 2693), may be created and deleted.

74659 **EXTENDED DESCRIPTION**

74660 None.

74661 **EXIT STATUS**

74662 The following exit values shall be returned:

74663 0 Successful completion.

74664 >0 An error occurred.

74665 **CONSEQUENCES OF ERRORS**

74666 Default.

74667 **APPLICATION USAGE**

74668 It is recommended that directories containing SCCS files be writable by the owner only, and that
 74669 SCCS files themselves be read-only. The mode of the directories should allow only the owner to
 74670 modify SCCS files contained in the directories. The mode of the SCCS files prevents any
 74671 modification at all except by SCCS commands.

74672 **EXAMPLES**

74673 None.

74674 **RATIONALE**

74675 None.

74676 **FUTURE DIRECTIONS**

74677 None.

74678 **SEE ALSO**74679 *delta, get, prs, what*74680 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

74681 **CHANGE HISTORY**

74682 First released in Issue 2.

74683 **Issue 6**

74684 The normative text is reworded to avoid use of the term “must” for application requirements,
 74685 and to emphasize the term “shall” for implementation requirements.

74686 The grammar is updated.

74687 The Open Group Base Resolution bwg2001-007 is applied, adding new text to the INPUT FILES
 74688 section warning that the maximum lines recorded in the file is 99 999.

74689 The Open Group Base Resolution bwg2001-009 is applied, amending the description of the **-h**
 74690 option.

74691 **NAME**
 74692 alias — define or display aliases

74693 **SYNOPSIS**
 74694 alias [*alias-name*[=*string*]. . .]

74695 **DESCRIPTION**
 74696 The *alias* utility shall create or redefine alias definitions or write the values of existing alias
 74697 definitions to standard output. An alias definition provides a string value that shall replace a
 74698 command name when it is encountered; see [Section 2.3.1](#) (on page 2248).

74699 An alias definition shall affect the current shell execution environment and the execution
 74700 environments of the subshells of the current shell. When used as specified by this volume of
 74701 POSIX.1-200x, the alias definition shall not affect the parent process of the current shell nor any
 74702 utility environment invoked by the shell; see [Section 2.12](#) (on page 2277).

74703 **OPTIONS**
 74704 None.

74705 **OPERANDS**
 74706 The following operands shall be supported:
 74707 *alias-name* Write the alias definition to standard output.
 74708 *alias-name=string*
 74709 Assign the value of *string* to the alias *alias-name*.
 74710 If no operands are given, all alias definitions shall be written to standard output.

74711 **STDIN**
 74712 Not used.

74713 **INPUT FILES**
 74714 None.

74715 **ENVIRONMENT VARIABLES**
 74716 The following environment variables shall affect the execution of *alias*:
 74717 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 74718 (See [XBD Section 8.2](#) (on page 160) for the precedence of internationalization |
 74719 variables used to determine the values of locale categories.)

74720 *LC_ALL* If set to a non-empty string value, override the values of all the other
 74721 internationalization variables.

74722 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 74723 characters (for example, single-byte as opposed to multi-byte characters in
 74724 arguments).

74725 *LC_MESSAGES*
 74726 Determine the locale that should be used to affect the format and contents of
 74727 diagnostic messages written to standard error.

74728 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

74729 **ASYNCHRONOUS EVENTS**
 74730 Default.

74731 **STDOUT**

74732 The format for displaying aliases (when no operands or only *name* operands are specified) shall
74733 be:

74734 `"%s=%s\n", name, value`

74735 The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the
74736 shell. See the description of shell quoting in [Section 2.2](#) (on page 2246).

74737 **STDERR**

74738 The standard error shall be used only for diagnostic messages.

74739 **OUTPUT FILES**

74740 None.

74741 **EXTENDED DESCRIPTION**

74742 None.

74743 **EXIT STATUS**

74744 The following exit values shall be returned:

74745 0 Successful completion.

74746 >0 One of the *name* operands specified did not have an alias definition, or an error occurred.

74747 **CONSEQUENCES OF ERRORS**

74748 Default.

74749 **APPLICATION USAGE**

74750 None.

74751 **EXAMPLES**

74752 1. Create a short alias for a commonly used *ls* command:

74753 `alias lf="ls -CF"`

74754 2. Create a simple "redo" command to repeat previous entries in the command history file:

74755 `alias r='fc -s'`

74756 3. Use 1K units for *du*:

74757 `alias du=du\ -k`

74758 4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

74759 `alias nohup="nohup "`

74760 **RATIONALE**

74761 The *alias* description is based on historical KornShell implementations. Known differences exist
74762 between that and the C shell. The KornShell version was adopted to be consistent with all the
74763 other KornShell features in this volume of POSIX.1-200x, such as command line editing.

74764 Since *alias* affects the current shell execution environment, it is generally provided as a shell
74765 regular built-in.

74766 Historical versions of the KornShell have allowed aliases to be exported to scripts that are
74767 invoked by the same shell. This is triggered by the *alias -x* flag; it is allowed by this volume of
74768 POSIX.1-200x only when an explicit extension such as *-x* is used. The standard developers
74769 considered that aliases were of use primarily to interactive users and that they should normally
74770 not affect shell scripts called by those users; functions are available to such scripts.

74771 Historical versions of the KornShell had not written aliases in a quoted manner suitable for
74772 reentry to the shell, but this volume of POSIX.1-200x has made this a requirement for all similar
74773 output. Therefore, consistency was chosen over this detail of historical practice.

74774

FUTURE DIRECTIONS

74775

None.

74776

SEE ALSO

74777

[Section 2.9.5](#) (on page 2270)

74778

XBD [Chapter 8](#) (on page 159) +

74779

CHANGE HISTORY

74780

First released in Issue 4.

74781

Issue 6

74782

This utility is marked as part of the User Portability Utilities option.

74783

The APPLICATION USAGE section is added.

74784

Issue 7

74785

The *alias* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

74786

74787

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

74788

The first example is changed to remove the creation of an alias for a standard utility that alters its behavior to be non-conforming. +

74789

DRAFT

74790 NAME

74791 ar — create and maintain library archives

74792 SYNOPSIS

74793 SD ar -d [-v] archive file...

74794 XSI ar -m [-v] archive file...

74795 ar -m -a [-v] posname archive file...

74796 ar -m -b [-v] posname archive file...

74797 ar -m -i [-v] posname archive file...

74798 XSI ar -p [-v] [-s] archive [file...]

74799 XSI ar -q [-cv] archive file...

74800 ar -r [-cuv] archive file...

74801 XSI ar -r -a [-cuv] posname archive file...

74802 ar -r -b [-cuv] posname archive file...

74803 ar -r -i [-cuv] posname archive file...

74804 XSI ar -t [-v] [-s] archive [file...]

74805 XSI ar -x [-v] [-sCT] archive [file...]

74806 DESCRIPTION

74807 The *ar* utility is part of the Software Development Utilities option.

74808 The *ar* utility can be used to create and maintain groups of files combined into an archive. Once
 74809 an archive has been created, new files can be added, and existing files in an archive can be
 74810 extracted, deleted, or replaced. When an archive consists entirely of valid object files, the
 74811 implementation shall format the archive so that it is usable as a library for link editing (see *c99*
 74812 and *fort77*). When some of the archived files are not valid object files, the suitability of the
 74813 XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire
 74814 archive shall be printable.

74815 When *ar* creates an archive, it creates administrative information indicating whether a symbol
 74816 table is present in the archive. When there is at least one object file that *ar* recognizes as such in
 74817 the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is
 74818 used by the link editor to search the archive. Whenever the *ar* utility is used to create or update
 74819 the contents of such an archive, the symbol table shall be rebuilt. The *-s* option shall force the
 74820 symbol table to be rebuilt.

74821 All *file* operands can be pathnames. However, files within archives shall be named by a filename,
 74822 which is the last component of the pathname used when the file was entered into the archive.
 74823 The comparison of *file* operands to the names of files in archives shall be performed by
 74824 comparing the last component of the operand to the name of the file in the archive.

74825 It is unspecified whether multiple files in the archive may be identically named. In the case of
 74826 XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive
 74827 having a name that is the same as the last component of the operand.

74828 OPTIONS

74829 The *ar* utility shall conform to XBD Section 12.2 (on page 201), except for Guideline 9.

74830 The following options shall be supported:

- 74831 XSI **-a** Position new files in the archive after the file named by the *posname* operand.
- 74832 XSI **-b** Position new files in the archive before the file named by the *posname* operand.
- 74833 **-c** Suppress the diagnostic message that is written to standard error by default when
74834 the archive *archive* is created.
- 74835 XSI **-C** Prevent extracted files from replacing like-named files in the file system. This
74836 option is useful when **-T** is also used, to prevent truncated filenames from
74837 replacing files with the same prefix.
- 74838 **-d** Delete one or more *files* from *archive*.
- 74839 XSI **-i** Position new files in the archive before the file in the archive named by the *posname*
74840 operand (equivalent to **-b**).
- 74841 XSI **-m** Move the named files in the archive. The **-a**, **-b**, or **-i** options with the *posname*
74842 operand indicate the position; otherwise, move the names files in the archive to the
74843 end of the archive.
- 74844 **-p** Write the contents of the *files* in the archive named by *file* operands from *archive* to
74845 the standard output. If no *file* operands are specified, the contents of all files in the
74846 archive shall be written in the order of the archive.
- 74847 XSI **-q** Append the named files to the end of the archive. In this case *ar* does not check
74848 whether the added files are already in the archive. This is useful to bypass the
74849 searching otherwise done when creating a large archive piece by piece.
- 74850 **-r** Replace or add *files* to *archive*. If the archive named by *archive* does not exist, a new
74851 archive shall be created and a diagnostic message shall be written to standard error
74852 (unless the **-c** option is specified). If no *files* are specified and the *archive* exists, the
74853 results are undefined. Files that replace existing files in the archive shall not change
74854 the order of the archive. Files that do not replace existing files in the archive shall
74855 be appended to the archive unless a **-a**, **-b**, or **-i** option specifies another position.
- 74856 XSI **-s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an
74857 option that modifies the archive contents. This option is useful to restore the
74858 archive symbol table after it has been stripped; see *strip*.
- 74859 **-t** Write a table of contents of *archive* to the standard output. Only the files specified
74860 by the *file* operands shall be included in the written list. If no *file* operands are
74861 specified, all files in *archive* shall be included in the order of the archive.
- 74862 XSI **-T** Allow filename truncation of extracted files whose archive names are longer than
74863 the file system can support. By default, extracting a file with a name that is too
74864 long shall be an error; a diagnostic message shall be written and the file shall not
74865 be extracted.
- 74866 **-u** Update older files in the archive. When used with the **-r** option, files in the archive
74867 shall be replaced only if the corresponding *file* has a modification time that is at
74868 least as new as the modification time of the file in the archive.
- 74869 **-v** Give verbose output. When used with the option characters **-d**, **-r**, or **-x**, write a
74870 detailed file-by-file description of the archive creation and maintenance activity, as
74871 described in the STDOUT section.
- 74872 When used with **-p**, write the name of the file in the archive to the standard output

74873 before writing the file in the archive itself to the standard output, as described in
 74874 the STDOUT section.

74875 When used with `-t`, include a long listing of information about the files in the
 74876 archive, as described in the STDOUT section.

74877 `-x` Extract the files in the archive named by the *file* operands from *archive*. The
 74878 contents of the archive shall not be changed. If no *file* operands are given, all files
 74879 in the archive shall be extracted. The modification time of each file extracted shall
 74880 be set to the time the file is extracted from the archive.

OPERANDS

74881 The following operands shall be supported:

74882 *archive* A pathname of the archive.

74883 *file* A pathname. Only the last component shall be used when comparing against the
 74884 names of files in the archive. If two or more *file* operands have the same last
 74885 pathname component (basename), the results are unspecified. The
 74886 implementation's archive format shall not truncate valid filenames of files added
 74887 to or replaced in the archive.

74888 XSI *posname* The name of a file in the archive, used for relative positioning; see options `-m` and
 74889 `-r`.

STDIN

74891 Not used.

INPUT FILES

74892 The archive named by *archive* shall be a file in the format created by `ar -r`.

ENVIRONMENT VARIABLES

74895 The following environment variables shall affect the execution of *ar*:

74897 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 74898 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 74899 variables used to determine the values of locale categories.)

74900 *LC_ALL* If set to a non-empty string value, override the values of all the other
 74901 internationalization variables.

74902 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 74903 characters (for example, single-byte as opposed to multi-byte characters in
 74904 arguments and input files).

74905 *LC_MESSAGES*
 74906 Determine the locale that should be used to affect the format and contents of
 74907 diagnostic messages written to standard error.

74908 *LC_TIME* Determine the format and content for date and time strings written by `ar -tv`.

74909 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

74910 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if
 74911 any.

74912 *TZ* Determine the timezone used to calculate date and time strings written by `ar -tv`.
 74913 If *TZ* is unset or null, an unspecified default timezone shall be used.

74914 **ASYNCHRONOUS EVENTS**

74915 Default.

74916 **STDOUT**74917 If the `-d` option is used with the `-v` option, the standard output format shall be:74918 `"d - %s\n", <file>`74919 where *file* is the operand specified on the command line.74920 If the `-p` option is used with the `-v` option, *ar* shall precede the contents of each file with:74921 `"\n<%s>\n\n", <file>`74922 where *file* is the operand specified on the command line, if *file* operands were specified, and the
74923 name of the file in the archive if they were not.74924 If the `-r` option is used with the `-v` option:74925 • If *file* is already in the archive, the standard output format shall be:74926 `"r - %s\n", <file>`74927 where *<file>* is the operand specified on the command line.74928 • If *file* is not already in the archive, the standard output format shall be:74929 `"a - %s\n", <file>`74930 where *<file>* is the operand specified on the command line.74931 If the `-t` option is used, *ar* shall write the names of the files in the archive to the standard output
74932 in the format:74933 `"%s\n", <file>`74934 where *file* is the operand specified on the command line, if *file* operands were specified, or the
74935 name of the file in the archive if they were not.74936 If the `-t` option is used with the `-v` option, the standard output format shall be:74937 `"%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,
74938 <group ID>, <number of bytes in member>,
74939 <abbreviated month>, <day-of-month>, <hour>,
74940 <minute>, <year>, <file>`

74941 where:

74942 *<file>* Shall be the operand specified on the command line, if *file* operands were specified,
74943 or the name of the file in the archive if they were not.74944 *<member mode>*74945 Shall be formatted the same as the *<file mode>* string defined in the STDOUT
74946 section of *ls*, except that the first character, the *<entry type>*, is not used; the string
74947 represents the file mode of the file in the archive at the time it was added to or
74948 replaced in the archive.74949 The following represent the last-modification time of a file when it was most recently added to
74950 or replaced in the archive:74951 *<abbreviated month>*74952 Equivalent to the format of the `%b` conversion specification format in *date*.74953 *<day-of-month>*74954 Equivalent to the format of the `%e` conversion specification format in *date*.

74955 <hour> Equivalent to the format of the %H conversion specification format in *date*.

74956 <minute> Equivalent to the format of the %M conversion specification format in *date*.

74957 <year> Equivalent to the format of the %Y conversion specification format in *date*.

74958 When *LC_TIME* does not specify the POSIX locale, a different format and order of presentation
74959 of these fields relative to each other may be used in a format appropriate in the specified locale.

74960 If the *-x* option is used with the *-v* option, the standard output format shall be:

74961 "x - %s\n", <file>

74962 where *file* is the operand specified on the command line, if *file* operands were specified, or the
74963 name of the file in the archive if they were not.

74964 **STDERR**

74965 The standard error shall be used only for diagnostic messages. The diagnostic message about
74966 creating a new archive when *-c* is not specified shall not modify the exit status.

74967 **OUTPUT FILES**

74968 Archives are files with unspecified formats.

74969 **EXTENDED DESCRIPTION**

74970 None.

74971 **EXIT STATUS**

74972 The following exit values shall be returned:

74973 0 Successful completion.

74974 >0 An error occurred.

74975 **CONSEQUENCES OF ERRORS**

74976 Default.

74977 **APPLICATION USAGE**

74978 None.

74979 **EXAMPLES**

74980 None.

74981 **RATIONALE**

74982 The archive format is not described. It is recognized that there are several known *ar* formats,
74983 which are not compatible. The *ar* utility is included, however, to allow creation of archives that
74984 are intended for use only on one machine. The archive is specified as a file, and it can be moved
74985 as a file. This does allow an archive to be moved from one machine to another machine that uses
74986 the same implementation of *ar*.

74987 Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable "archives". This is a not
74988 a duplication; the *ar* utility is included to provide an interface primarily for *make* and the
74989 compilers, based on a historical model.

74990 In historical implementations, the *-q* option (available on XSI-conforming systems) is known to
74991 execute quickly because *ar* does not check on whether the added members are already in the
74992 archive. This is useful to bypass the searching otherwise done when creating a large archive
74993 piece-by-piece. These remarks may but need not remain true for a brand new implementation of
74994 this utility; hence, these remarks have been moved into the RATIONALE.

74995 BSD implementations historically required applications to provide the *-s* option whenever the
74996 archive was supposed to contain a symbol table. As in this volume of POSIX.1-200x, System V
74997 historically creates or updates an archive symbol table whenever an object file is removed from,
74998 added to, or updated in the archive.

74999 The OPERANDS section requires what might seem to be true without specifying it: the archive
 75000 cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so,
 75001 however, causing unexpected results for the application. Therefore, this volume of POSIX.1-200x
 75002 makes the requirement explicit to avoid misunderstandings.

75003 According to the System V documentation, the options `-dmpqrtx` are not required to begin with
 75004 a hyphen ('-'). This volume of POSIX.1-200x requires that a conforming application use the
 75005 leading hyphen.

75006 The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as
 75007 an example:

75008 A file created by *ar* begins with the "magic" string "`!<arch>\n". The rest of the archive
 75009 is made up of objects, each of which is composed of a header for a file, a possible filename,
 75010 and the file contents. The header is portable between machine architectures, and, if the file
 75011 contents are printable, the archive is itself printable.`

75012 The header is made up of six ASCII fields, followed by a two-character trailer. The fields
 75013 are the object name (16 characters), the file last modification time (12 characters), the user
 75014 and group IDs (each 6 characters), the file mode (8 characters), and the file size (10
 75015 characters). All numeric fields are in decimal, except for the file mode, which is in octal.

75016 The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid`
 75017 and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field.
 75018 The two-byte trailer is the string "`\n`".

75019 Only the name field has any provision for overflow. If any filename is more than 16
 75020 characters in length or contains an embedded space, the string "`#1/`" followed by the
 75021 ASCII length of the name is written in the name field. The file size (stored in the archive
 75022 header) is incremented by the length of the name. The name is then written immediately
 75023 following the archive header.

75024 Any unused characters in any of these fields are written as `<space>s`. If any fields are their
 75025 particular maximum number of characters in length, there is no separation between the
 75026 fields.

75027 Objects in the archive are always an even number of bytes long; files that are an odd
 75028 number of bytes long are padded with a `<newline>`, although the size in the header does
 75029 not reflect this.

75030 The *ar* utility description requires that (when all its members are valid object files) *ar* produce an
 75031 object code library, which the linkage editor can use to extract object modules. If the linkage
 75032 editor needs a symbol table to permit random access to the archive, *ar* must provide it; however,
 75033 *ar* does not require a symbol table.

75034 The BSD `-o` option was omitted. It is a rare conforming application that uses *ar* to extract object
 75035 code from a library with concern for its modification time, since this can only be of importance
 75036 to *make*. Hence, since this functionality is not deemed important for applications portability, the
 75037 modification time of the extracted files is set to the current time.

75038 There is at least one known implementation (for a small computer) that can accommodate only
 75039 object files for that system, disallowing mixed object and other files. The ability to handle any
 75040 type of file is not only historical practice for most implementations, but is also a reasonable
 75041 expectation.

75042 Consideration was given to changing the output format of *ar* `-tv` to the same format as the
 75043 output of *ls* `-l`. This would have made parsing the output of *ar* the same as that of *ls*. This was
 75044 rejected in part because the current *ar* format is commonly used and changes would break
 75045 historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a
 75046 slash. Changing this to be the user name and group name would not be correct if the archive

75047 were moved to a machine that contained a different user database. Since *ar* cannot know
75048 whether the archive was generated on the same machine, it cannot tell what to report.

75049 The text on the **-ur** option combination is historical practice—since one filename can easily
75050 represent two different files (for example, **/a/foo** and **/b/foo**), it is reasonable to replace the file in
75051 the archive even when the modification time in the archive is identical to that in the file system.

75052 FUTURE DIRECTIONS

75053 None.

75054 SEE ALSO

75055 *c99, date, fort77, pax, strip*

75056 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [<unistd.h>](#), description of
75057 {POSIX_NO_TRUNC}

75058 CHANGE HISTORY

75059 First released in Issue 2.

75060 Issue 5

75061 The FUTURE DIRECTIONS section is added.

75062 Issue 6

75063 This utility is marked as part of the Software Development Utilities option.

75064 The STDOUT description is changed for the **-v** option to align with the IEEE P1003.2b draft
75065 standard.

75066 The normative text is reworded to avoid use of the term “must” for application requirements.

75067 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

75068 IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use
75069 “file” to refer to a file in the file system hierarchy, “archive” to refer to the archive being
75070 operated upon by the *ar* utility, and “file in the archive” to refer to a copy of a file that is
75071 contained in the archive.

75072 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/10 is applied, making corrections to the
75073 SYNOPSIS. The change was needed since the **-a**, **-b**, and **-i** options are mutually-exclusive, and
75074 *posname* is required if any of these options is specified.

75075 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/11 is applied, correcting the description
75076 of the two-byte trailer in RATIONALE which had missed out a backquote. The correct trailer is a
75077 backquote followed by a <newline>.

75078 Issue 7

75079 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
75080 apply.

75081 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

75082 The description of the **-t** option is changed to say “Only the files specified ...”. +

75083 **NAME**
 75084 `asa` — interpret carriage-control characters

75085 **SYNOPSIS**
 75086 FR `asa [file...]`

75087 **DESCRIPTION**
 75088 The *asa* utility shall write its input files to standard output, mapping carriage-control characters
 75089 from the text files to line-printer control sequences in an implementation-defined manner.
 75090 The first character of every line shall be removed from the input, and the following actions are
 75091 performed.
 75092 If the character removed is:
 75093 <space> The rest of the line is output without change.
 75094 0 A <newline> is output, then the rest of the input line.
 75095 1 One or more implementation-defined characters that causes an advance to the next
 75096 page shall be output, followed by the rest of the input line.
 75097 + The <newline> of the previous line shall be replaced with one or more implementation-
 75098 defined characters that causes printing to return to column position 1, followed by the
 75099 rest of the input line. If the '+' is the first character in the input, it shall be equivalent
 75100 to <space>.
 75101 The action of the *asa* utility is unspecified upon encountering any character other than those
 75102 listed above as the first character in a line.

75103 **OPTIONS**
 75104 None.

75105 **OPERANDS**
 75106 *file* A pathname of a text file used for input. If no *file* operands are specified, the
 75107 standard input shall be used.

75108 **STDIN**
 75109 The standard input shall be used only if no *file* operands are specified; see the INPUT FILES
 75110 section.

75111 **INPUT FILES**
 75112 The input files shall be text files.

75113 **ENVIRONMENT VARIABLES**
 75114 The following environment variables shall affect the execution of *asa*:
 75115 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 75116 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 75117 variables used to determine the values of locale categories.)
 75118 *LC_ALL* If set to a non-empty string value, override the values of all the other
 75119 internationalization variables.
 75120 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 75121 characters (for example, single-byte as opposed to multi-byte characters in
 75122 arguments and input files).

75123 *LC_MESSAGES*
 75124 Determine the locale that should be used to affect the format and contents of
 75125 diagnostic messages written to standard error.

75126 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

75127 ASYNCHRONOUS EVENTS

75128 Default.

75129 STDOUT

75130 The standard output shall be the text from the input file modified as described in the
 75131 DESCRIPTION section.

75132 STDERR

75133 None.

75134 OUTPUT FILES

75135 None.

75136 EXTENDED DESCRIPTION

75137 None.

75138 EXIT STATUS

75139 The following exit values shall be returned:

75140 0 All input files were output successfully.

75141 >0 An error occurred.

75142 CONSEQUENCES OF ERRORS

75143 Default.

75144 APPLICATION USAGE

75145 None.

75146 EXAMPLES

75147 1. The following command:

75148 `asa file`

75149 permits the viewing of *file* (created by a program using FORTRAN-style carriage-control
 75150 characters) on a terminal.

75151 2. The following command:

75152 `a.out | asa | lp`

75153 formats the FORTRAN output of **a.out** and directs it to the printer.

75154 RATIONALE

75155 The *asa* utility is needed to map “standard” FORTRAN 77 output into a form acceptable to
 75156 contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

75157 This utility is generally used only by FORTRAN programs. The standard developers decided to
 75158 retain *asa* to avoid breaking the historical large base of FORTRAN applications that put carriage-
 75159 control characters in their output files. There is no requirement that a system have a FORTRAN
 75160 compiler in order to run applications that need *asa*.

75161 Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII
 75162 <carriage-return> in response to a '+'. It is suggested that implementations treat characters
 75163 other than 0, 1, and '+' as <space> in the absence of any compelling reason to do otherwise.
 75164 However, the action is listed here as “unspecified”, permitting an implementation to provide
 75165 extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

75166

FUTURE DIRECTIONS

75167

None.

75168

SEE ALSO

75169

fort77, lp

75170

XBD [Chapter 8](#) (on page 159)

+

75171

CHANGE HISTORY

75172

First released in Issue 4.

75173

Issue 6

75174

This utility is marked as part of the FORTRAN Runtime Utilities option.

75175

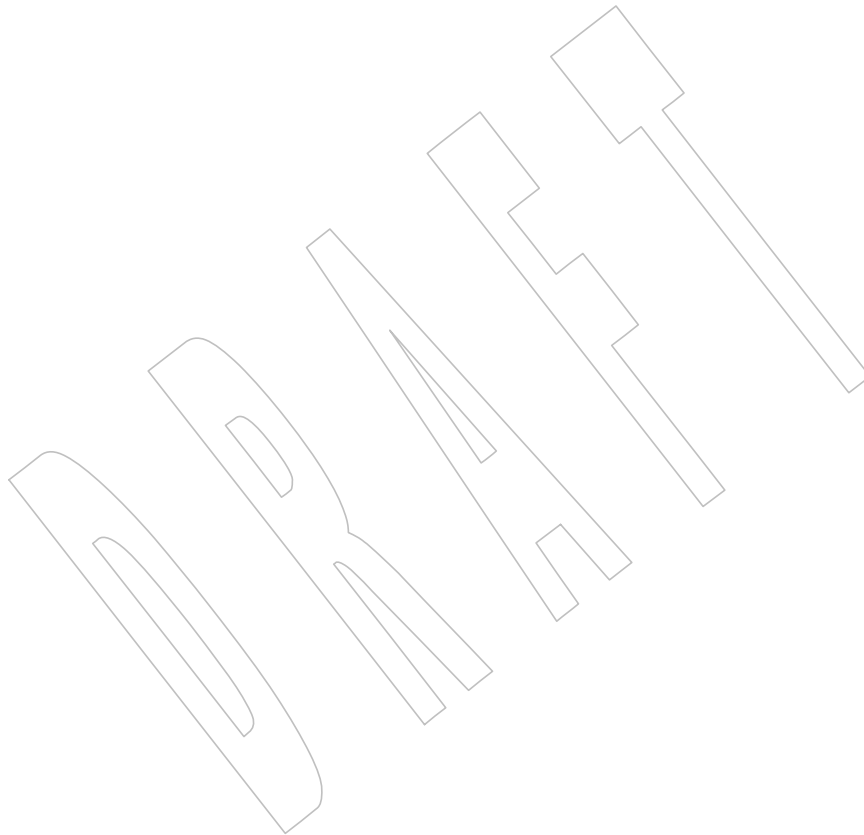
The normative text is reworded to avoid use of the term “must” for application requirements.

75176

Issue 7

75177

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



75178 **NAME**
 75179 `at` — execute commands at a later time

75180 **SYNOPSIS**
 75181 `at [-m] [-f file] [-q queueName] -t time_arg`
 75182 `at [-m] [-f file] [-q queueName] timespec...`
 75183 `at -r at_job_id...`
 75184 `at -l -q queueName`
 75185 `at -l [at_job_id...]`

75186 **DESCRIPTION**
 75187 The `at` utility shall read commands from standard input and group them together as an *at-job*, to
 75188 be executed at a later time.

75189 The *at-job* shall be executed in a separate invocation of the shell, running in a separate process
 75190 group with no controlling terminal, except that the environment variables, current working
 75191 directory, file creation mask, and other implementation-defined execution-time attributes in
 75192 effect when the `at` utility is executed shall be retained and used when the *at-job* is executed.

75193 When the *at-job* is submitted, the *at_job_id* and scheduled time shall be written to standard error.
 75194 The *at_job_id* is an identifier that shall be a string consisting solely of alphanumeric characters
 75195 and the period character. The *at_job_id* shall be assigned by the system when the job is scheduled
 75196 such that it uniquely identifies a particular job.

75197 User notification and the processing of the job's standard output and standard error are
 75198 described under the `-m` option.

75199 XSI Users shall be permitted to use `at` if their name appears in the file **at.allow** which is located in an
 75200 implementation-defined directory. If that file does not exist, the file **at.deny**, which is located in
 75201 an implementation-defined directory, shall be checked to determine whether the user shall be
 75202 denied access to `at`. If neither file exists, only a process with the appropriate privileges shall be
 75203 allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The
 75204 **at.allow** and **at.deny** files shall consist of one user name per line.

75205 **OPTIONS**
 75206 The `at` utility shall conform to XBD [Section 12.2](#) (on page 201).

75207 The following options shall be supported:

75208 `-f file` Specify the pathname of a file to be used as the source of the *at-job*, instead of
 75209 standard input.

75210 `-l` (The letter ell.) Report all jobs scheduled for the invoking user if no *at_job_id*
 75211 operands are specified. If *at_job_ids* are specified, report only information for these
 75212 jobs. The output shall be written to standard output.

75213 `-m` Send mail to the invoking user after the *at-job* has run, announcing its completion.
 75214 Standard output and standard error produced by the *at-job* shall be mailed to the
 75215 user as well, unless redirected elsewhere. Mail shall be sent even if the job
 75216 produces no output.

75217 If `-m` is not used, the job's standard output and standard error shall be provided to
 75218 the user by means of mail, unless they are redirected elsewhere; if there is no such
 75219 output to provide, the implementation need not notify the user of the job's
 75220 completion.

- 75221 **-q** *queuename*
- 75222 Specify in which queue to schedule a job for submission. When used with the **-l**
- 75223 option, limit the search to that particular queue. By default, at-jobs shall be
- 75224 scheduled in queue *a*. In contrast, queue *b* shall be reserved for batch jobs; see
- 75225 *batch*. The meanings of all other *queuenames* are implementation-defined. If **-q** is
- 75226 specified along with either of the **-t** *time_arg* or *timespec* arguments, the results are
- 75227 unspecified.
- 75228 **-r** Remove the jobs with the specified *at_job_id* operands that were previously
- 75229 scheduled by the *at* utility.
- 75230 **-t** *time_arg* Submit the job to be run at the time specified by the *time* option-argument, which
- 75231 the application shall ensure has the format as specified by the *touch -t time* utility.

OPERANDS

The following operands shall be supported:

- 75234 *at_job_id* The name reported by a previous invocation of the *at* utility at the time the job was
- 75235 scheduled.
- 75236 *timespec* Submit the job to be run at the date and time specified. All of the *timespec* operands
- 75237 are interpreted as if they were separated by <space>s and concatenated, and shall
- 75238 be parsed as described in the grammar at the end of this section. The date and time
- 75239 shall be interpreted as being in the timezone of the user (as determined by the *TZ*
- 75240 variable), unless a timezone name appears as part of *time*, below.
- 75241 In the POSIX locale, the following describes the three parts of the time specification
- 75242 string. All of the values from the *LC_TIME* categories in the POSIX locale shall be
- 75243 recognized in a case-insensitive manner.
- 75244 *time* The time can be specified as one, two, or four digits. One-digit and
- 75245 two-digit numbers shall be taken to be hours; four-digit numbers to
- 75246 be hours and minutes. The time can alternatively be specified as two
- 75247 numbers separated by a colon, meaning *hour:minute*. An AM/PM
- 75248 indication (one of the values from the **am_pm** keywords in the
- 75249 *LC_TIME* locale category) can follow the time; otherwise, a 24-hour
- 75250 clock time shall be understood. A timezone name can also follow to
- 75251 further qualify the time. The acceptable timezone names are
- 75252 implementation-defined, except that they shall be case-insensitive
- 75253 and the string **utc** is supported to indicate the time is in Coordinated
- 75254 Universal Time. In the POSIX locale, the *time* field can also be one of
- 75255 the following tokens:
- 75256 **midnight** Indicates the time 12:00 am (00:00).
- 75257 **noon** Indicates the time 12:00 pm.
- 75258 **now** Indicates the current day and time. Invoking *at* <**now**>
- 75259 shall submit an at-job for potentially immediate
- 75260 execution (that is, subject only to unspecified
- 75261 scheduling delays).
- 75262 *date* An optional *date* can be specified as either a month name (one of the
- 75263 values from the **mon** or **abmon** keywords in the *LC_TIME* locale
- 75264 category) followed by a day number (and possibly year number
- 75265 preceded by a comma), or a day of the week (one of the values from
- 75266 the **day** or **abday** keywords in the *LC_TIME* locale category). In the
- 75267 POSIX locale, two special days shall be recognized:

75268 **today** Indicates the current day.

75269 **tomorrow** Indicates the day following the current day.

75270 If no *date* is given, **today** shall be assumed if the given time is greater
75271 than the current time, and **tomorrow** shall be assumed if it is less. If
75272 the given month is less than the current month (and no year is given),
75273 next year shall be assumed.

75274 *increment* The optional *increment* shall be a number preceded by a plus sign
75275 ('+') and suffixed by one of the following: **minutes**, **hours**, **days**,
75276 **weeks**, **months**, or **years**. (The singular forms shall also be accepted.)
75277 The keyword **next** shall be equivalent to an increment number of +1.
75278 For example, the following are equivalent commands:

75279 at 2pm + 1 week
75280 at 2pm next week

75281 The following grammar describes the precise format of *timespec* in the POSIX locale. The general
75282 conventions for this style of grammar are described in [Section 1.3](#) (on page 2235). This formal
75283 syntax shall take precedence over the preceding text syntax description. The longest possible
75284 token or delimiter shall be recognized at a given point. When used in a *timespec*, white space
75285 shall also delimit tokens.

```
75286 %token hr24clock_hr_min
75287 %token hr24clock_hour
75288 /*
75289     An hr24clock_hr_min is a one, two, or four-digit number. A one-digit
75290     or two-digit number constitutes an hr24clock_hour. An hr24clock_hour
75291     may be any of the single digits [0,9], or may be double digits, ranging
75292     from [00,23]. If an hr24clock_hr_min is a four-digit number, the
75293     first two digits shall be a valid hr24clock_hour, while the last two
75294     represent the number of minutes, from [00,59].
75295 */
75296 %token wallclock_hr_min
75297 %token wallclock_hour
75298 /*
75299     A wallclock_hr_min is a one, two-digit, or four-digit number.
75300     A one-digit or two-digit number constitutes a wallclock_hour.
75301     A wallclock_hour may be any of the single digits [1,9], or may
75302     be double digits, ranging from [01,12]. If a wallclock_hr_min
75303     is a four-digit number, the first two digits shall be a valid
75304     wallclock_hour, while the last two represent the number of
75305     minutes, from [00,59].
75306 */
75307 %token minute
75308 /*
75309     A minute is a one or two-digit number whose value can be [0,9]
75310     or [00,59].
75311 */
75312 %token day_number
75313 /*
75314     A day_number is a number in the range appropriate for the particular
75315     month and year specified by month_name and year_number, respectively.
75316     If no year_number is given, the current year is assumed if the given
75317     date and time are later this year. If no year_number is given and
```

```

75318         the date and time have already occurred this year and the month is
75319         not the current month, next year is the assumed year.
75320     */

75321     %token year_number
75322     /*
75323         A year_number is a four-digit number representing the year A.D., in
75324         which the at_job is to be run.
75325     */

75326     %token inc_number
75327     /*
75328         The inc_number is the number of times the succeeding increment
75329         period is to be added to the specified date and time.
75330     */

75331     %token timezone_name
75332     /*
75333         The name of an optional timezone suffix to the time field, in an
75334         implementation-defined format.
75335     */

75336     %token month_name
75337     /*
75338         One of the values from the mon or abmon keywords in the LC_TIME
75339         locale category.
75340     */

75341     %token day_of_week
75342     /*
75343         One of the values from the day or abday keywords in the LC_TIME
75344         locale category.
75345     */

75346     %token am_pm
75347     /*
75348         One of the values from the am_pm keyword in the LC_TIME locale
75349         category.
75350     */

75351     %start timespec
75352     %%
75353     timespec      : time
75354                   | time date
75355                   | time increment
75356                   | time date increment
75357                   | nowspec
75358                   ;

75359     nowspec      : "now"
75360                   | "now" increment
75361                   ;

75362     time         : hr24clock_hr_min
75363                   | hr24clock_hr_min timezone_name
75364                   | hr24clock_hour ":" minute
75365                   | hr24clock_hour ":" minute timezone_name
75366                   | wallclock_hr_min am_pm
75367                   | wallclock_hr_min am_pm timezone_name

```

```

75368         | wallclock_hour ":" minute am_pm
75369         | wallclock_hour ":" minute am_pm timezone_name
75370         | "noon"
75371         | "midnight"
75372         ;

75373     date      : month_name day_number
75374         | month_name day_number "," year_number
75375         | day_of_week
75376         | "today"
75377         | "tomorrow"
75378         ;

75379     increment : "+" inc_number inc_period
75380         | "next" inc_period
75381         ;

75382     inc_period : "minute" | "minutes"
75383         | "hour" | "hours"
75384         | "day" | "days"
75385         | "week" | "weeks"
75386         | "month" | "months"
75387         | "year" | "years"
75388         ;

```

STDIN

75389 The standard input shall be a text file consisting of commands acceptable to the shell command
75390 language described in [Chapter 2](#) (on page 2245). The standard input shall only be used if no `-f`
75391 `file` option is specified.
75392

INPUT FILES

75393 See the STDIN section.
75394

75395 XSI The text files `at.allow` and `at.deny`, which are located in an implementation-defined directory,
75396 shall contain zero or more user names, one per line, of users who are, respectively, authorized or
75397 denied access to the `at` and `batch` utilities.

ENVIRONMENT VARIABLES

75398 The following environment variables shall affect the execution of `at`:
75399

75400 `LANG` Provide a default value for the internationalization variables that are unset or null. |
75401 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
75402 variables used to determine the values of locale categories.)

75403 `LC_ALL` If set to a non-empty string value, override the values of all the other
75404 internationalization variables.

75405 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
75406 characters (for example, single-byte as opposed to multi-byte characters in
75407 arguments and input files).

75408 `LC_MESSAGES`
75409 Determine the locale that should be used to affect the format and contents of
75410 diagnostic messages written to standard error and informative messages written to
75411 standard output.

75412 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

75413 `LC_TIME` Determine the format and contents for date and time strings written and accepted
75414 by `at`.

75415 *SHELL* Determine a name of a command interpreter to be used to invoke the *at*-job. If the
 75416 variable is unset or null, *sh* shall be used. If it is set to a value other than a name for
 75417 *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the
 75418 login shell from the user database; or any of the preceding accompanied by a
 75419 warning diagnostic about which was chosen.

75420 *TZ* Determine the timezone. The job shall be submitted for execution at the time
 75421 specified by *timespec* or *-t time* relative to the timezone specified by the *TZ*
 75422 variable. If *timespec* specifies a timezone, it shall override *TZ*. If *timespec* does not
 75423 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall
 75424 be used.

75425 **ASYNCHRONOUS EVENTS**

75426 Default.

75427 **STDOUT**

75428 When standard input is a terminal, prompts of unspecified format for each line of the user input
 75429 described in the *STDIN* section may be written to standard output.

75430 In the POSIX locale, the following shall be written to the standard output for each job when jobs
 75431 are listed in response to the *-l* option:

75432 `"%s\t%s\n", at_job_id, <date>`

75433 where *date* shall be equivalent in format to the output of:

75434 `date +"%a %b %e %T %Y"`

75435 The date and time written shall be adjusted so that they appear in the timezone of the user (as
 75436 determined by the *TZ* variable).

75437 **STDERR**

75438 In the POSIX locale, the following shall be written to standard error when a job has been
 75439 successfully submitted:

75440 `"job %s at %s\n", at_job_id, <date>`

75441 where *date* has the same format as that described in the *STDOUT* section. Neither this, nor
 75442 warning messages concerning the selection of the command interpreter, shall be considered a
 75443 diagnostic that changes the exit status.

75444 Diagnostic messages, if any, shall be written to standard error.

75445 **OUTPUT FILES**

75446 None.

75447 **EXTENDED DESCRIPTION**

75448 None.

75449 **EXIT STATUS**

75450 The following exit values shall be returned:

75451 0 The *at* utility successfully submitted, removed, or listed a job or jobs.

75452 >0 An error occurred.

75453 **CONSEQUENCES OF ERRORS**

75454 The job shall not be scheduled, removed, or listed.

APPLICATION USAGE

The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other cultures may be supported with substantially different interfaces, although implementations are encouraged to provide comparable levels of functionality.

Since the commands run in a separate shell invocation, running in a separate process group with no controlling terminal, open file descriptors, traps, and priority inherited from the invoking environment are lost.

Some implementations do not allow substitution of different shells using *SHELL*. System V systems, for example, have used the login shell value for the user in */etc/passwd*. To select reliably another command interpreter, the user must include it as part of the script, such as:

```
$ at 1800
myshell myscript
EOT
job ... at ...
$
```

EXAMPLES

1. This sequence can be used at a terminal:

```
at -m 0730 tomorrow
sort < file >outfile
EOT
```

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
at now + 1 hour <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

3. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this daily processing script named **my.daily** runs every day (although *crontab* is a more appropriate vehicle for such work):

```
# my.daily runs every day
daily processing
at now tomorrow < my.daily
```

4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as there are no ambiguities. Examples of various times and operand presentation include:

```
at 0815am Jan 24
at 8 :15amjan24
at now "+ 1day"
at 5 pm FRIday
at '17
    utc+
    30minutes'
```

RATIONALE

The *at* utility reads from standard input the commands to be executed at a later time. It may be useful to redirect standard output and standard error within the specified commands.

The *-t time* option was added as a new capability to support an internationalized way of specifying a time for execution of the submitted job.

Early proposals added a “jobname” concept as a way of giving submitted jobs names that are meaningful to the user submitting them. The historical, system-specified *at_job_id* gives no

75502 indication of what the job is. Upon further reflection, it was decided that the benefit of this was
 75503 not worth the change in historical interface. The *at* functionality is useful in simple
 75504 environments, but in large or complex situations, the functionality provided by the Batch
 75505 Services option is more suitable.

75506 The `-q` option historically has been an undocumented option, used mainly by the *batch* utility.

75507 The System V `-m` option was added to provide a method for informing users that an at-job had
 75508 completed. Otherwise, users are only informed when output to standard error or standard
 75509 output are not redirected.

75510 The behavior of *at* `<now>` was changed in an early proposal from being unspecified to
 75511 submitting a job for potentially immediate execution. Historical BSD *at* implementations support
 75512 this. Historical System V implementations give an error in that case, but a change to the System
 75513 V versions should have no backwards-compatibility ramifications.

75514 On BSD-based systems, a `-u user` option has allowed those with appropriate privileges to access
 75515 the work of other users. Since this is primarily a system administration feature and is not
 75516 universally implemented, it has been omitted. Similarly, a specification for the output format for
 75517 a user with appropriate privileges viewing the queues of other users has been omitted.

75518 The `-f file` option from System V is used instead of the BSD method of using the last operand as
 75519 the pathname. The BSD method is ambiguous—does:

75520 `at 1200 friday`

75521 mean the same thing if there is a file named **friday** in the current directory?

75522 The *at_job_id* is composed of a limited character set in historical practice, and it is mandated here
 75523 to invalidate systems that might try using characters that require shell quoting or that could not
 75524 be easily parsed by shell scripts.

75525 The *at* utility varies between System V and BSD systems in the way timezones are used. On
 75526 System V systems, the *TZ* variable affects the at-job submission times and the times displayed
 75527 for the user. On BSD systems, *TZ* is not taken into account. The BSD behavior is easily achieved
 75528 with the current specification. If the user wishes to have the timezone default to that of the
 75529 system, they merely need to issue the *at* command immediately following an unsetting or null
 75530 assignment to *TZ*. For example:

75531 `TZ= at noon ...`

75532 gives the desired BSD result.

75533 While the *yacc*-like grammar specified in the OPERANDS section is lexically unambiguous with
 75534 respect to the digit strings, a lexical analyzer would probably be written to look for and return
 75535 digit strings in those cases. The parser could then check whether the digit string returned is a
 75536 valid *day_number*, *year_number*, and so on, based on the context.

75537 FUTURE DIRECTIONS

75538 None.

75539 SEE ALSO

75540 *batch*, *crontab*

75541 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

+

75542 CHANGE HISTORY

75543 First released in Issue 2.

75544 **Issue 6**
75545 This utility is marked as part of the User Portability Utilities option.

75546 The following new requirements on POSIX implementations derive from alignment with the
75547 Single UNIX Specification:

- 75548 • If **-m** is not used, the job's standard output and standard error are provided to the user by
75549 mail.

75550 The effects of using the **-q** and **-t** options as defined in the IEEE P1003.2b draft standard are
75551 specified.

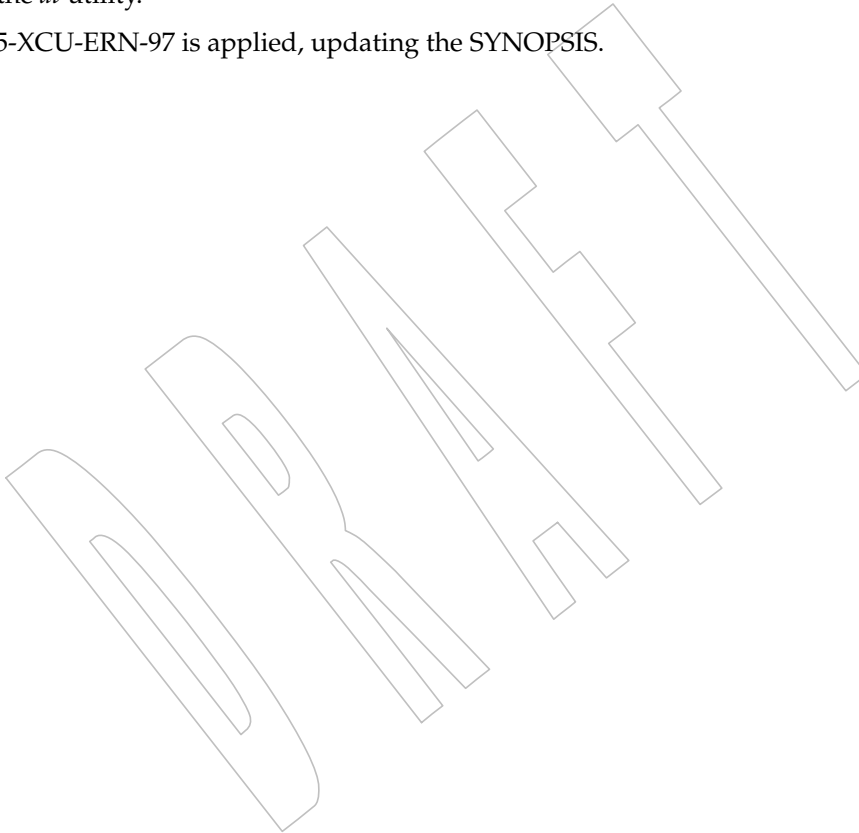
75552 The normative text is reworded to avoid use of the term "must" for application requirements.

75553 **Issue 7**

75554 The *at* utility is moved from the User Portability Utilities option to the Base. User Portability
75555 Utilities is now an option for interactive utilities.

75556 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
75557 by the *at* utility.

75558 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



75559 **NAME**
 75560 awk — pattern scanning and processing language

75561 **SYNOPSIS**
 75562 awk [-F *ERE*] [-v *assignment*...] *program* [*argument*...]

75563 awk [-F *ERE*] -f *progfile* [-f *progfile*]... [-v *assignment*]...
 75564 [*argument*...]

75565 DESCRIPTION

75566 The *awk* utility shall execute programs written in the *awk* programming language, which is
 75567 specialized for textual data manipulation. An *awk* program is a sequence of patterns and
 75568 corresponding actions. When input is read that matches a pattern, the action associated with that
 75569 pattern is carried out.

75570 Input shall be interpreted as a sequence of records. By default, a record is a line, less its
 75571 terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of
 75572 input shall be matched in turn against each pattern in the program. For each pattern matched,
 75573 the associated action shall be executed.

75574 The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field
 75575 is a string of non-<blank>s. This default white-space field delimiter can be changed by using the
 75576 **FS** built-in variable or **-F *ERE***. The *awk* utility shall denote the first field in a record \$1, the
 75577 second \$2, and so on. The symbol \$0 shall refer to the entire record; setting any other field causes
 75578 the re-evaluation of \$0. Assigning to \$0 shall reset the values of all other fields and the **NF** built-
 75579 in variable.

75580 OPTIONS

75581 The *awk* utility shall conform to XBD [Section 12.2](#) (on page 201).

75582 The following options shall be supported:

75583 **-F *ERE*** Define the input field separator to be the extended regular expression *ERE*, before
 75584 any input is read; see [Regular Expressions](#) (on page 2379).

75585 **-f *progfile*** Specify the pathname of the file *progfile* containing an *awk* program. If multiple
 75586 instances of this option are specified, the concatenation of the files specified as
 75587 *progfile* in the order specified shall be the *awk* program. The *awk* program can
 75588 alternatively be specified in the command line as a single argument.

75589 **-v *assignment***
 75590 The application shall ensure that the *assignment* argument is in the same form as an
 75591 *assignment* operand. The specified variable assignment shall occur prior to
 75592 executing the *awk* program, including the actions associated with **BEGIN** patterns
 75593 (if any). Multiple occurrences of this option can be specified.

75594 OPERANDS

75595 The following operands shall be supported:

75596 *program* If no **-f** option is specified, the first operand to *awk* shall be the text of the *awk*
 75597 program. The application shall supply the *program* operand as a single argument to
 75598 *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

75599 *argument* Either of the following two types of *argument* can be intermixed:

75600 *file* A pathname of a file that contains the input to be read, which is
 75601 matched against the set of patterns in the program. If no *file* operands
 75602 are specified, or if a *file* operand is '-', the standard input shall be
 75603 used.

75604 *assignment* An operand that begins with an underscore or alphabetic character
 75605 from the portable character set (see the table in XBD Section 6.1, on
 75606 page 111), followed by a sequence of underscores, digits, and
 75607 alphabets from the portable character set, followed by the '='
 75608 character, shall specify a variable assignment rather than a pathname.
 75609 The characters before the '=' represent the name of an *awk* variable;
 75610 if that name is an *awk* reserved word (see Grammar, on page 2386)
 75611 the behavior is undefined. The characters following the equal sign
 75612 shall be interpreted as if they appeared in the *awk* program preceded
 75613 and followed by a double-quote ('"') character, as a **STRING** token
 75614 (see Grammar, on page 2386), except that if the last character is an
 75615 unescaped backslash, it shall be interpreted as a literal backslash
 75616 rather than as the first character of the sequence "\". The variable
 75617 shall be assigned the value of that **STRING** token and, if appropriate,
 75618 shall be considered a *numeric string* (see Expressions in awk, on page
 75619 2374), the variable shall also be assigned its numeric value. Each such
 75620 variable assignment shall occur just prior to the processing of the
 75621 following *file*, if any. Thus, an assignment before the first *file*
 75622 argument shall be executed after the **BEGIN** actions (if any), while an
 75623 assignment after the last *file* argument shall occur before the **END**
 75624 actions (if any). If there are no *file* arguments, assignments shall be
 75625 executed before processing the standard input.

75626 STDIN

75627 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-';
 75628 see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is
 75629 otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk*
 75630 shall exit with a return status of zero.

75631 INPUT FILES

75632 Input files to the *awk* program from any of the following sources shall be text files:

- 75633 • Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV**
 75634 and **ARGC**
- 75635 • Standard input in the absence of any *file* operands
- 75636 • Arguments to the **getline** function

75637 Whether the variable **RS** is set to a value other than a <newline> or not, for these files,
 75638 implementations shall support records terminated with the specified separator up to
 75639 {LINE_MAX} bytes and may support longer records.

75640 If **-f progfile** is specified, the application shall ensure that the files named by each of the *progfile*
 75641 option-arguments are text files and their concatenation, in the same order as they appear in the
 75642 arguments, is an *awk* program.

75643 ENVIRONMENT VARIABLES

75644 The following environment variables shall affect the execution of *awk*:

75645 **LANG** Provide a default value for the internationalization variables that are unset or null.
 75646 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 75647 variables used to determine the values of locale categories.)

75648 **LC_ALL** If set to a non-empty string value, override the values of all the other
 75649 internationalization variables.

75650 **LC_COLLATE**

75651 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 75652 character collating elements within regular expressions and in comparisons of

75653 string values.

75654 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 75655 characters (for example, single-byte as opposed to multi-byte characters in
 75656 arguments and input files), the behavior of character classes within regular
 75657 expressions, the identification of characters as letters, and the mapping of
 75658 uppercase and lowercase characters for the **toupper** and **tolower** functions.

75659 **LC_MESSAGES**
 75660 Determine the locale that should be used to affect the format and contents of
 75661 diagnostic messages written to standard error.

75662 **LC_NUMERIC**
 75663 Determine the radix character used when interpreting numeric input, performing
 75664 conversions between numeric and string values, and formatting numeric output.
 75665 Regardless of locale, the period character (the decimal-point character of the POSIX
 75666 locale) is the decimal-point character recognized in processing *awk* programs
 75667 (including assignments in command line arguments).

75668 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

75669 **PATH** Determine the search path when looking for commands executed by *system(expr)*,
 75670 or input and output pipes; see XBD [Chapter 8](#) (on page 159).

75671 In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.

75672 ASYNCHRONOUS EVENTS

75673 Default.

75674 STDOUT

75675 The nature of the output files depends on the *awk* program.

75676 STDERR

75677 The standard error shall be used only for diagnostic messages.

75678 OUTPUT FILES

75679 The nature of the output files depends on the *awk* program.

75680 EXTENDED DESCRIPTION

75681 Overall Program Structure

75682 An *awk* program is composed of pairs of the form:

75683 *pattern* { *action* }

75684 Either the pattern or the action (including the enclosing brace characters) can be omitted.

75685 A missing pattern shall match any record of input, and a missing action shall be equivalent to:

75686 { *print* }

75687 Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN**
 75688 patterns in the order they occur in the program. Then each *file* operand (or standard input if no
 75689 files were specified) shall be processed in turn by reading data from the file until a record
 75690 separator is seen (<newline> by default). Before the first reference to a field in the record is
 75691 evaluated, the record shall be split into fields, according to the rules in [Regular Expressions](#) (on
 75692 page 2379), using the value of **FS** that was current at the time the record was read. Each pattern
 75693 in the program then shall be evaluated in the order of occurrence, and the action associated with
 75694 each pattern that matches the current record executed. The action for a matching pattern shall be
 75695 executed before evaluating subsequent patterns. Finally, the actions associated with all **END**
 75696 patterns shall be executed in the order they occur in the program.

75697

Expressions in awk

75698

75699

75700

75701

75702

75703

75704

75705

Expressions describe computations used in *patterns* and *actions*. In the following table, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, where the grammar is formally ambiguous, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while *lvalue* represents any entity that can be assigned to (that is, on the left side of an assignment operator). The precise syntax of expressions is given in [Grammar](#) (on page 2386).

75706

Table 4-1 Expressions in Decreasing Precedence in *awk*

75707

75708

75709

75710

75711

75712

75713

75714

75715

75716

75717

75718

75719

75720

75721

75722

75723

75724

75725

75726

75727

75728

75729

75730

75731

75732

75733

75734

75735

75736

75737

75738

75739

75740

75741

75742

75743

Syntax	Name	Type of Result	Associativity
(<i>expr</i>)	Grouping	Type of <i>expr</i>	N/A
<i>\$expr</i>	Field reference	String	N/A
<i>lvalue</i> ++	Post-increment	Numeric	N/A
<i>lvalue</i> --	Post-decrement	Numeric	N/A
++ <i>lvalue</i>	Pre-increment	Numeric	N/A
-- <i>lvalue</i>	Pre-decrement	Numeric	N/A
<i>expr</i> ^ <i>expr</i>	Exponentiation	Numeric	Right
! <i>expr</i>	Logical not	Numeric	N/A
+ <i>expr</i>	Unary plus	Numeric	N/A
- <i>expr</i>	Unary minus	Numeric	N/A
<i>expr</i> * <i>expr</i>	Multiplication	Numeric	Left
<i>expr</i> / <i>expr</i>	Division	Numeric	Left
<i>expr</i> % <i>expr</i>	Modulus	Numeric	Left
<i>expr</i> + <i>expr</i>	Addition	Numeric	Left
<i>expr</i> - <i>expr</i>	Subtraction	Numeric	Left
<i>expr</i> <i>expr</i>	String concatenation	String	Left
<i>expr</i> < <i>expr</i>	Less than	Numeric	None
<i>expr</i> <= <i>expr</i>	Less than or equal to	Numeric	None
<i>expr</i> != <i>expr</i>	Not equal to	Numeric	None
<i>expr</i> == <i>expr</i>	Equal to	Numeric	None
<i>expr</i> > <i>expr</i>	Greater than	Numeric	None
<i>expr</i> >= <i>expr</i>	Greater than or equal to	Numeric	None
<i>expr</i> ~ <i>expr</i>	ERE match	Numeric	None
<i>expr</i> !~ <i>expr</i>	ERE non-match	Numeric	None
<i>expr</i> in <i>array</i>	Array membership	Numeric	Left
(<i>index</i>) in <i>array</i>	Multi-dimension array membership	Numeric	Left
<i>expr</i> && <i>expr</i>	Logical AND	Numeric	Left
<i>expr</i> <i>expr</i>	Logical OR	Numeric	Left
<i>expr1</i> ? <i>expr2</i> : <i>expr3</i>	Conditional expression	Type of selected <i>expr2</i> or <i>expr3</i>	Right
<i>lvalue</i> ^= <i>expr</i>	Exponentiation assignment	Numeric	Right
<i>lvalue</i> %= <i>expr</i>	Modulus assignment	Numeric	Right
<i>lvalue</i> *= <i>expr</i>	Multiplication assignment	Numeric	Right
<i>lvalue</i> /= <i>expr</i>	Division assignment	Numeric	Right
<i>lvalue</i> += <i>expr</i>	Addition assignment	Numeric	Right

75744

75745

75746

Syntax	Name	Type of Result	Associativity
<code>lvalue -= expr</code>	Subtraction assignment	Numeric	Right
<code>lvalue = expr</code>	Assignment	Type of <i>expr</i>	Right

75747

75748

75749

75750

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the ISO C standard:

75751

```
setlocale(LC_NUMERIC, "");
```

75752

```
numeric_value = atof(string_value);
```

75753

75754

75755

75756

75757

75758

75759

75760

75761

75762

75763

A numeric value that is exactly equal to the value of an integer (see [Section 1.1.2](#), on page 2231) shall be converted to a string by the equivalent of a call to the `sprintf` function (see [String Functions](#), on page 2384) with the string `"%d"` as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the `sprintf` function with the value of the variable `CONVFMT` as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of `CONVFMT` is not a floating-point format specification. This volume of POSIX.1-200x specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (`" "`) to it.

75764

A string value shall be considered a *numeric string* if it comes from one of the following:

75765

1. Field variables

75766

2. Input from the `getline()` function

75767

3. `FILENAME`

75768

4. `ARGV` array elements

75769

5. `ENVIRON` array elements

75770

6. Array elements created by the `split()` function

75771

7. A command line variable assignment

75772

8. Variable assignment from another numeric string variable

75773

75774

75775

and after all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in [Grammar](#) (on page 2386):

75776

- All leading and trailing `<blank>`s are discarded.

75777

- If the first non-`<blank>` is `'+'` or `'-'`, it is discarded.

75778

- Changing each occurrence of the decimal point character from the current locale to a period.

75779

75780

75781

75782

75783

75784

If a `'-'` character is ignored in the preceding description, the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token. Otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

75785

75786

75787

75788

When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean context shall be one of the following:

- 75789 • The first subexpression of a conditional expression
- 75790 • An expression operated on by logical NOT, logical AND, or logical OR
- 75791 • The second expression of a **for** statement
- 75792 • The expression of an **if** statement
- 75793 • The expression of the **while** clause in either a **while** or **do...while** statement
- 75794 • An expression used as a pattern (as in Overall Program Structure)

75795 All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C
75796 standard (see [Section 1.1.2](#), on page 2231).

75797 The value of the expression:

75798 `expr1 ^ expr2`

75799 shall be equivalent to the value returned by the ISO C standard function call:

75800 `pow(expr1, expr2)`

75801 The expression:

75802 `lvalue ^= expr`

75803 shall be equivalent to the ISO C standard expression:

75804 `lvalue = pow(lvalue, expr)`

75805 except that `lvalue` shall be evaluated only once. The value of the expression:

75806 `expr1 % expr2`

75807 shall be equivalent to the value returned by the ISO C standard function call:

75808 `fmod(expr1, expr2)`

75809 The expression:

75810 `lvalue %= expr`

75811 shall be equivalent to the ISO C standard expression:

75812 `lvalue = fmod(lvalue, expr)`

75813 except that `lvalue` shall be evaluated only once.

75814 Variables and fields shall be set by the assignment statement:

75815 `lvalue = expression`

75816 and the type of *expression* shall determine the resulting variable type. The assignment includes
75817 the arithmetic assignments ("`+=`", "`-=`", "`*=`", "`/=`", "`%=`", "`^=`", "`++`", "`--`") all of which
75818 shall produce a numeric result. The left-hand side of an assignment and the target of increment
75819 and decrement operators can be one of a variable, an array with index, or a field selector.

75820 The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not
75821 be declared. They shall initially be empty, and their sizes shall change dynamically. The
75822 subscripts, or element identifiers, are strings, providing a type of associative array capability. An
75823 array name followed by a subscript within square brackets can be used as an lvalue and thus as
75824 an expression, as described in the grammar; see [Grammar](#) (on page 2386). Unsubscripted array
75825 names can be used in only the following contexts:

- 75826 • A parameter in a function definition or function call

- 75827 • The **NAME** token following any use of the keyword **in** as specified in the grammar (see
75828 [Grammar](#), on page 2386); if the name used in this context is not an array name, the
75829 behavior is undefined

75830 A valid array *index* shall consist of one or more comma-separated expressions, similar to the way
75831 in which multi-dimensional arrays are indexed in some programming languages. Because *awk*
75832 arrays are really one-dimensional, such a comma-separated list shall be converted to a single
75833 string by concatenating the string values of the separate expressions, each separated from the
75834 other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be
75835 equivalent:

```
75836 var[expr1, expr2, ... exprn]
```

```
75837 var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

75838 The application shall ensure that a multi-dimensioned *index* used with the **in** operator is
75839 parenthesized. The **in** operator, which tests for the existence of a particular array element, shall
75840 not cause that element to exist. Any other reference to a nonexistent array element shall
75841 automatically create it.

75842 Comparisons (with the '<', '<=', '!=', '==', '>', and '>=' operators) shall be made
75843 numerically if both operands are numeric, if one is numeric and the other has a string value that
75844 is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise,
75845 operands shall be converted to strings as required and a string comparison shall be made using
75846 the locale-specific collation sequence. The value of the comparison expression shall be 1 if the
75847 relation is true, or 0 if the relation is false.

75848 Variables and Special Variables

75849 Variables can be used in an *awk* program by referencing them. With the exception of function
75850 parameters (see [User-Defined Functions](#), on page 2386), they are not explicitly declared.
75851 Function parameter names shall be local to the function; all other variable names shall be global.
75852 The same name shall not be used as both a function parameter name and as the name of a
75853 function or a special *awk* variable. The same name shall not be used both as a variable name with
75854 global scope and as the name of a function. The same name shall not be used within the same
75855 scope both as a scalar variable and as an array. Uninitialized variables, including scalar
75856 variables, array elements, and field variables, shall have an uninitialized value. An uninitialized
75857 value shall have both a numeric value of zero and a string value of the empty string. Evaluation
75858 of variables with an uninitialized value, to either string or numeric, shall be determined by the
75859 context in which they are used.

75860 Field variables shall be designated by a '\$' followed by a number or numerical expression. The
75861 effect of the field number *expression* evaluating to anything other than a non-negative integer is
75862 unspecified; uninitialized variables or string values need not be converted to numeric values in
75863 this context. New field variables can be created by assigning a value to them. References to
75864 nonexistent fields (that is, fields after **\$NF**), shall evaluate to the uninitialized value. Such
75865 references shall not create new fields. However, assigning to a nonexistent field (for example,
75866 $\$(NF+2)=5$) shall increase the value of **NF**; create any intervening fields with the uninitialized
75867 value; and cause the value of **\$0** to be recomputed, with the fields being separated by the value
75868 of **OFS**. Each field variable shall have a string value or an uninitialized value when created.
75869 Field variables shall have the uninitialized value when created from **\$0** using **FS** and the variable
75870 does not contain any characters. If appropriate, the field variable shall be considered a numeric
75871 string (see [Expressions in awk](#), on page 2374).

75872 Implementations shall support the following other special variables that are set by *awk*:

75873 **ARGC** The number of elements in the **ARGV** array.

75874 75875	ARGV	An array of command line arguments, excluding options and the <i>program</i> argument, numbered from zero to ARGC -1.
75876 75877 75878 75879 75880 75881 75882		The arguments in ARGV can be modified or added to; ARGC can be altered. As each input file ends, <i>awk</i> shall treat the next non-null element of ARGV , up to the current value of ARGC -1, inclusive, as the name of the next input file. Thus, setting an element of ARGV to null means that it shall not be treated as an input file. The name '-' indicates the standard input. If an argument matches the format of an <i>assignment</i> operand, this argument shall be treated as an <i>assignment</i> rather than a <i>file</i> argument.
75883 75884	CONVFMT	The printf format for converting numbers to strings (except for output statements, where OFMT is used); "% . 6g" by default.
75885 75886 75887 75888 75889 75890 75891	ENVIRON	An array representing the value of the environment, as described in the <i>exec</i> functions defined in the System Interfaces volume of POSIX.1-200x. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a <i>numeric string</i> (see Expressions in awk , on page 2374); the array element shall also have its numeric value.
75892 75893 75894 75895 75896 75897		In all cases where the behavior of <i>awk</i> is affected by environment variables (including the environment of any commands that <i>awk</i> executes via the system function or via pipeline redirections with the print statement, the printf statement, or the getline function), the environment used shall be the environment at the time <i>awk</i> began executing; it is implementation-defined whether any modification of ENVIRON affects this environment.
75898 75899 75900	FILENAME	A pathname of the current input file. Inside a BEGIN action the value is undefined. Inside an END action the value shall be the name of the last input file processed.
75901 75902 75903	FNR	The ordinal number of the current record in the current file. Inside a BEGIN action the value shall be zero. Inside an END action the value shall be the number of the last record processed in the last file processed.
75904	FS	Input field separator regular expression; a <space> by default.
75905 75906 75907 75908 75909	NF	The number of fields in the current record. Inside a BEGIN action, the use of NF is undefined unless a getline function without a <i>var</i> argument is executed previously. Inside an END action, NF shall retain the value it had for the last record read, unless a subsequent, redirected, getline function without a <i>var</i> argument is performed prior to entering the END action.
75910 75911 75912	NR	The ordinal number of the current record from the start of input. Inside a BEGIN action the value shall be zero. Inside an END action the value shall be the number of the last record processed.
75913 75914 75915	OFMT	The printf format for converting numbers to strings in output statements (see Output Statements , on page 2382); "% . 6g" by default. The result of the conversion is unspecified if the value of OFMT is not a floating-point format specification.
75916	OFS	The print statement output field separator; <space> by default.
75917	ORS	The print statement output record separator; a <newline> by default.
75918	RLENGTH	The length of the string matched by the match function.

- 75919 **RS** The first character of the string value of **RS** shall be the input record separator; a
 75920 <newline> by default. If **RS** contains more than one character, the results are
 75921 unspecified. If **RS** is null, then records are separated by sequences consisting of a
 75922 <newline> plus one or more blank lines, leading or trailing blank lines shall not
 75923 result in empty records at the beginning or end of the input, and a <newline> shall
 75924 always be a field separator, no matter what the value of **FS** is.
- 75925 **RSTART** The starting position of the string matched by the **match** function, numbering from
 75926 1. This shall always be equivalent to the return value of the **match** function.
- 75927 **SUBSEP** The subscript separator string for multi-dimensional arrays; the default value is
 75928 implementation-defined.

75929 Regular Expressions

75930 The *awk* utility shall make use of the extended regular expression notation (see XBD Section 9.4,
 75931 on page 174) except that it shall allow the use of C-language conventions for escaping special
 75932 characters within the EREs, as specified in the table in XBD Chapter 5 (on page 107) ('\\',
 75933 '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v') and the following table; these escape sequences
 75934 shall be recognized both inside and outside bracket expressions. Note that records need not be
 75935 separated by <newline>s and string constants can contain <newline>s, so even the "\\n"
 75936 sequence is valid in *awk* EREs. Using a slash character within an ERE requires the escaping
 75937 shown in the following table.

75938 **Table 4-2** Escape Sequences in *awk*

Escape Sequence	Description	Meaning
\"	Backslash quotation-mark	Quotation-mark character
\/	Backslash slash	Slash character
\\ddd	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading '\\' for each byte.
\\c	A backslash character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 107) ('\\', '\\a', '\\b', '\\f', '\\n', '\\r', '\\t', '\\v').	Undefined

75955 A regular expression can be matched against a specific field or string by using one of the two
 75956 regular expression matching operators, '~' and '!~'. These operators shall interpret their
 75957 right-hand operand as a regular expression and their left-hand operand as a string. If the regular
 75958 expression matches the string, the '~' expression shall evaluate to a value of 1, and the '!~'
 75959 expression shall evaluate to a value of 0. (The regular expression matching operation is as
 75960 defined by the term matched in XBD Section 9.1 (on page 167), where a match occurs on any part
 75961 of the string unless the regular expression is limited with the circumflex or dollar sign special
 75962 characters.) If the regular expression does not match the string, the '~' expression shall evaluate
 75963 to a value of 0, and the '!~' expression shall evaluate to a value of 1. If the right-hand operand
 75964 is any expression other than the lexical token **ERE**, the string value of the expression shall be
 75965 interpreted as an extended regular expression, including the escape conventions described
 75966 above. Note that these same escape conventions shall also be applied in determining the value

75967 of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a
75968 string literal is used in this context.

75969 When an **ERE** token appears as an expression in any context other than as the right-hand of the
75970 '`~`' or '`!~`' operator or as one of the built-in function arguments described below, the value of
75971 the resulting expression shall be the equivalent of:

75972 `$0 ~ /ere/`

75973 The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function
75974 (see [String Functions](#), on page 2384) shall be interpreted as extended regular expressions. These
75975 can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner
75976 as the right-hand side of the '`~`' or '`!~`' operator.

75977 An extended regular expression can be used to separate fields by using the `-F ERE` option or by
75978 assigning a string containing the expression to the built-in variable **FS**. The default value of the
75979 **FS** variable shall be a single `<space>`. The following describes **FS** behavior:

- 75980 1. If **FS** is a null string, the behavior is unspecified.
- 75981 2. If **FS** is a single character:
 - 75982 a. If **FS** is `<space>`, skip leading and trailing `<blank>`s; fields shall be delimited by
75983 sets of one or more `<blank>`s.
 - 75984 b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single
75985 occurrence of *c*.
- 75986 3. Otherwise, the string value of **FS** shall be considered to be an extended regular
75987 expression. Each occurrence of a sequence matching the extended regular expression shall
75988 delimit fields.

75989 Except for the '`~`' and '`!~`' operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions,
75990 ERE matching shall be based on input records; that is, record separator characters (the first
75991 character of the value of the variable **RS**, `<newline>` by default) cannot be embedded in the
75992 expression, and no expression shall match the record separator character. If the record separator
75993 is not `<newline>`, `<newline>`s embedded in the expression can be matched. For the '`~`' and
75994 '`!~`' operators, and in those four built-in functions, ERE matching shall be based on text
75995 strings; that is, any character (including `<newline>` and the record separator) can be embedded
75996 in the pattern, and an appropriate pattern shall match any character. However, in all *awk* ERE
75997 matching, the use of one or more NUL characters in the pattern, input record, or text string
75998 produces undefined results.

75999 **Patterns**

76000 A *pattern* is any valid *expression*, a range specified by two expressions separated by a comma, or
76001 one of the two special patterns **BEGIN** or **END**.

76002 **Special Patterns**

76003 The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern
76004 shall be matched once and its associated action executed before the first record of input is read—
76005 except possibly by use of the **getline** function (see [Input/Output and General Functions](#), on
76006 page 2385) in a prior **BEGIN** action—and before command line assignment is done. Each **END**
76007 pattern shall be matched once and its associated action executed after the last record of input has
76008 been read. These two patterns shall have associated actions.

76009 **BEGIN** and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns
76010 shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order
76011 specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern
76012 in a program.

76013 If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action
 76014 contains no **getline** function, *awk* shall exit without reading its input when the last statement in
 76015 the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern
 76016 **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the
 76017 statements in the **END** actions are executed.

76018 Expression Patterns

76019 An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the
 76020 result is true, the pattern shall be considered to match, and the associated action (if any) shall be
 76021 executed. If the result is false, the action shall not be executed.

76022 Pattern Ranges

76023 A pattern range consists of two expressions separated by a comma; in this case, the action shall
 76024 be performed for all records between a match of the first expression and the following match of
 76025 the second expression, inclusive. At this point, the pattern range can be repeated starting at
 76026 input records subsequent to the end of the matched range.

76027 Actions

76028 An action is a sequence of statements as shown in the grammar in [Grammar](#) (on page 2386).
 76029 Any single statement can be replaced by a statement list enclosed in braces. The application shall
 76030 ensure that statements in a statement list are separated by <newline>s or semicolons. Statements
 76031 in a statement list shall be executed sequentially in the order that they appear.

76032 The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero
 76033 or non-null, the following statement shall be executed; otherwise, if **else** is present, the statement
 76034 following the **else** shall be executed.

76035 The **if**, **while**, **do...while**, **for**, **break**, and **continue** statements are based on the ISO C standard
 76036 (see [Section 1.1.2](#), on page 2231), except that the Boolean expressions shall be treated as
 76037 described in [Expressions in awk](#) (on page 2374), and except in the case of:

```
76038 for (variable in array)
```

76039 which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of
 76040 adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue**
 76041 statement occurs outside of a loop, the behavior is undefined.

76042 The **delete** statement shall remove an individual array element. Thus, the following code deletes
 76043 an entire array:

```
76044 for (index in array)
76045     delete array[index]
```

76046 The **next** statement shall cause all further processing of the current input record to be
 76047 abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or
 76048 **END** action.

76049 The **exit** statement shall invoke all **END** actions in the order in which they occur in the program
 76050 source and then terminate the program without reading further input. An **exit** statement inside
 76051 an **END** action shall terminate the program without further execution of **END** actions. If an
 76052 expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*,
 76053 unless subsequent errors are encountered or a subsequent **exit** statement with an expression is
 76054 executed.

76055

Output Statements

76056

Both **print** and **printf** statements shall write to standard output by default. The output shall be written to the location specified by *output_redirection* if one is supplied, as follows:

76057

76058

```
> expression
```

76059

```
>> expression
```

76060

```
| expression
```

76061

In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into which to write (for '**>**' or "**>>**") or as a command to be executed (for '**|**'). Using the first two forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and using the first form, truncating the file. The output then shall be appended to the file. As long as the file remains open, subsequent calls in which *expression* evaluates to the same string value shall simply append output to the file. The file remains open until the **close** function (see [Input/Output and General Functions](#), on page 2385) is called with an expression that evaluates to the same string value.

76062

76063

76064

76065

76066

76067

76068

The third form shall write output onto a stream piped to the input of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the *popen()* function defined in the System Interfaces volume of POSIX.1-200x with the value of *expression* as the *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall write output to the existing stream. The stream shall remain open until the **close** function (see [Input/Output and General Functions](#), on page 2385) is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the *pclose()* function defined in the System Interfaces volume of POSIX.1-200x.

76069

76070

76071

76072

76073

76074

76075

76076

76077

76078

As described in detail by the grammar in [Grammar](#) (on page 2386), these output statements shall take a comma-separated list of *expressions* referred to in the grammar by the non-terminal symbols **expr_list**, **print_expr_list**, or **print_expr_list_opt**. This list is referred to here as the *expression list*, and each member is referred to as an *expression argument*.

76083

76084

76085

76086

76087

76088

76089

The **print** statement shall write the value of each expression argument onto the indicated output stream separated by the current output field separator (see variable **OFS** above), and terminated by the output record separator (see variable **ORS** above). All expression arguments shall be taken as strings, being converted if necessary; this conversion shall be as described in [Expressions in awk](#) (on page 2374), with the exception that the **printf** format in **OFMT** shall be used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole input record (**\$0**).

76090

76091

76092

76093

76094

The **printf** statement shall produce output based on a notation similar to the File Format Notation used to describe file formats in this volume of POSIX.1-200x (see [XBD Chapter 5](#), on page 107). Output shall be produced as specified with the first *expression* argument as the string *format* and subsequent *expression* arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

76095

76096

76097

76098

1. The *format* shall be an actual character string rather than a graphical representation. Therefore, it cannot contain empty character positions. The **<space>** in the *format* string, in any context other than a *flag* of a conversion specification, shall be treated as an ordinary character that is copied to the output.

76099

76100

2. If the character set contains a '**Δ**' character and that character appears in the *format* string, it shall be treated as an ordinary character that is copied to the output.

76101

76102

76103

3. The *escape sequences* beginning with a backslash character shall be treated as sequences of ordinary characters that are copied to the output. Note that these same sequences shall be interpreted lexically by *awk* when they appear in literal strings, but they shall not be

- 76104 treated specially by the **printf** statement.
- 76105 4. A *field width* or *precision* can be specified as the '*' character instead of a digit string. In
76106 this case the next argument from the expression list shall be fetched and its numeric value
76107 taken as the field width or precision.
- 76108 5. The implementation shall not precede or follow output from the d or u conversion
76109 specifier characters with <blank>s not specified by the *format* string.
- 76110 6. The implementation shall not precede output from the o conversion specifier character
76111 with leading zeros not specified by the *format* string.
- 76112 7. For the c conversion specifier character: if the argument has a numeric value, the
76113 character whose encoding is that value shall be output. If the value is zero or is not the
76114 encoding of any character in the character set, the behavior is undefined. If the argument
76115 does not have a numeric value, the first character of the string value shall be output; if the
76116 string does not contain any characters, the behavior is undefined.
- 76117 8. For each conversion specification that consumes an argument, the next expression
76118 argument shall be evaluated. With the exception of the c conversion specifier character,
76119 the value shall be converted (according to the rules specified in [Expressions in awk](#), on
76120 page 2374) to the appropriate type for the conversion specification.
- 76121 9. If there are insufficient expression arguments to satisfy all the conversion specifications in
76122 the *format* string, the behavior is undefined.
- 76123 10. If any character sequence in the *format* string begins with a '%' character, but does not
76124 form a valid conversion specification, the behavior is unspecified.

76125 Both **print** and **printf** can output at least {LINE_MAX} bytes.

76126 Functions

76127 The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and
76128 general.

76129 Arithmetic Functions

76130 The arithmetic functions, except for **int**, shall be based on the ISO C standard (see [Section 1.1.2](#),
76131 on page 2231). The behavior is undefined in cases where the ISO C standard specifies that an
76132 error be returned or that the behavior is undefined. Although the grammar (see [Grammar](#), on
76133 page 2386) permits built-in functions to appear with no arguments or parentheses, unless the
76134 argument or parentheses are indicated as optional in the following list (by displaying them
76135 within the "[]" brackets), such use is undefined.

- 76136 **atan2**(*y*,*x*) Return arctangent of *y*/*x* in radians in the range $[-\pi, \pi]$.
- 76137 **cos**(*x*) Return cosine of *x*, where *x* is in radians.
- 76138 **sin**(*x*) Return sine of *x*, where *x* is in radians.
- 76139 **exp**(*x*) Return the exponential function of *x*.
- 76140 **log**(*x*) Return the natural logarithm of *x*.
- 76141 **sqrt**(*x*) Return the square root of *x*.
- 76142 **int**(*x*) Return the argument truncated to an integer. Truncation shall be toward 0 when
76143 *x*>0.
- 76144 **rand**() Return a random number *n*, such that $0 \leq n < 1$.

76145 **srand**(*expr*) Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The
76146 previous seed value shall be returned.

76147 String Functions

76148 The string functions in the following list shall be supported. Although the grammar (see
76149 [Grammar](#), on page 2386) permits built-in functions to appear with no arguments or parentheses,
76150 unless the argument or parentheses are indicated as optional in the following list (by displaying
76151 them within the "[]" brackets), such use is undefined.

76152 **gsub**(*ere, repl*[, *in*])
76153 Behave like **sub** (see below), except that it shall replace all occurrences of the
76154 regular expression (like the *ed* utility global substitute) in *\$0* or in the *in* argument,
76155 when specified.

76156 **index**(*s, t*) Return the position, in characters, numbering from 1, in string *s* where string *t* first
76157 occurs, or zero if it does not occur at all.

76158 **length**[(*s*)] Return the length, in characters, of its argument taken as a string, or of the whole
76159 record, *\$0*, if there is no argument.

76160 **match**(*s, ere*) Return the position, in characters, numbering from 1, in string *s* where the
76161 extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART
76162 shall be set to the starting position (which is the same as the returned value), zero
76163 if no match is found; RLENGTH shall be set to the length of the matched string, -1
76164 if no match is found.

76165 **split**(*s, a*[, *fs*])
76166 Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements
76167 of the array shall be deleted before the split is performed. The separation shall be
76168 done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array
76169 element shall have a string value when created and, if appropriate, the array
76170 element shall be considered a numeric string (see [Expressions in awk](#), on page
76171 2374). The effect of a null string as the value of *fs* is unspecified.

76172 **sprintf**(*fmt, expr, expr, ...*)
76173 Format the expressions according to the **printf** format given by *fmt* and return the
76174 resulting string.

76175 **sub**(*ere, repl*[, *in*])
76176 Substitute the string *repl* in place of the first instance of the extended regular
76177 expression *ERE* in string *in* and return the number of substitutions. An ampersand
76178 ('&') appearing in the string *repl* shall be replaced by the string from *in* that
76179 matches the ERE. An ampersand preceded with a backslash ('\&') shall be
76180 interpreted as the literal ampersand character. An occurrence of two consecutive
76181 backslashes shall be interpreted as just a single literal backslash character. Any
76182 other occurrence of a backslash (for example, preceding any other character) shall
76183 be treated as a literal backslash character. Note that if *repl* is a string literal (the
76184 lexical token **STRING**; see [Grammar](#), on page 2386), the handling of the
76185 ampersand character occurs after any lexical processing, including any lexical
76186 backslash escape sequence processing. If *in* is specified and it is not an lvalue (see
76187 [Expressions in awk](#), on page 2374), the behavior is undefined. If *in* is omitted, *awk*
76188 shall use the current record (*\$0*) in its place.

76189 **substr**(*s, m*[, *n*])
76190 Return the at most *n*-character substring of *s* that begins at position *m*, numbering
76191 from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string,
76192 the length of the substring shall be limited by the length of the string *s*.

- 76193 **tolower(s)** Return a string based on the string *s*. Each character in *s* that is an uppercase letter
76194 specified to have a **tolower** mapping by the *LC_CTYPE* category of the current
76195 locale shall be replaced in the returned string by the lowercase letter specified by
76196 the mapping. Other characters in *s* shall be unchanged in the returned string.
- 76197 **toupper(s)** Return a string based on the string *s*. Each character in *s* that is a lowercase letter
76198 specified to have a **toupper** mapping by the *LC_CTYPE* category of the current
76199 locale is replaced in the returned string by the uppercase letter specified by the
76200 mapping. Other characters in *s* are unchanged in the returned string.

76201 All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued
76202 expression that is a regular expression as defined in [Regular Expressions](#) (on page 2379).

76203 **Input/Output and General Functions**

76204 The input/output and general functions are:

76205 **close(expression)**
76206 Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with
76207 the same string-valued *expression*. The limit on the number of open *expression*
76208 arguments is implementation-defined. If the close was successful, the function
76209 shall return zero; otherwise, it shall return non-zero.

76210 *expression* | **getline** [*var*]
76211 Read a record of input from a stream piped from the output of a command. The
76212 stream shall be created if no stream is currently open with the value of *expression* as
76213 its command name. The stream created shall be equivalent to one created by a call
76214 to the *popen()* function with the value of *expression* as the *command* argument and a
76215 value of *r* as the *mode* argument. As long as the stream remains open, subsequent
76216 calls in which *expression* evaluates to the same string value shall read subsequent
76217 records from the stream. The stream shall remain open until the **close** function is
76218 called with an expression that evaluates to the same string value. At that time, the
76219 stream shall be closed as if by a call to the *pclose()* function. If *var* is omitted, \$0 and
76220 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered
76221 a numeric string (see [Expressions in awk](#), on page 2374).

76222 The **getline** operator can form ambiguous constructs when there are
76223 unparenthesized operators (including concatenate) to the left of the ' | ' (to the
76224 beginning of the expression containing **getline**). In the context of the '\$' operator,
76225 ' | ' shall behave as if it had a lower precedence than '\$'. The result of evaluating
76226 other operators is unspecified, and conforming applications shall parenthesize
76227 properly all such usages.

76228 **getline** Set \$0 to the next input record from the current input file. This form of **getline** shall
76229 set the **NF**, **NR**, and **FNR** variables.

76230 **getline** *var* Set variable *var* to the next input record from the current input file and, if
76231 appropriate, *var* shall be considered a numeric string (see [Expressions in awk](#), on
76232 page 2374). This form of **getline** shall set the **FNR** and **NR** variables.

76233 **getline** [*var*] < *expression*
76234 Read the next record of input from a named file. The *expression* shall be evaluated
76235 to produce a string that is used as a pathname. If the file of that name is not
76236 currently open, it shall be opened. As long as the stream remains open, subsequent
76237 calls in which *expression* evaluates to the same string value shall read subsequent
76238 records from the file. The file shall remain open until the **close** function is called
76239 with an expression that evaluates to the same string value. If *var* is omitted, \$0 and
76240 **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered
76241 a numeric string (see [Expressions in awk](#), on page 2374).

The **getline** operator can form ambiguous constructs when there are unparenthesized binary operators (including concatenate) to the right of the '<' (up to the end of the expression containing the **getline**). The result of evaluating such a construct is unspecified, and conforming applications shall parenthesize properly all such usages.

system(*expression*)

Execute the command given by *expression* in a manner equivalent to the *system*() function defined in the System Interfaces volume of POSIX.1-200x and return the exit status of the command.

All forms of **getline** shall return 1 for successful input, zero for end-of-file, and -1 for an error.

Where strings are used as the name of a file or pipeline, the application shall ensure that the strings are textually identical. The terminology "same string value" implies that "equivalent strings", even those that differ only by <space>s, represent different files.

User-Defined Functions

The *awk* language also provides user-defined functions. Such functions can be defined as:

```
function name([parameter, ...]) { statements }
```

A function can be referred to anywhere in an *awk* program; in particular, its use can precede its definition. The scope of a function is global.

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name.

The number of parameters in the function definition need not match the number of parameters in the function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall evaluate to the uninitialized value until they are otherwise initialized, and the extra parameters that are used in the function body as arrays shall be treated as uninitialized arrays where each element evaluates to the uninitialized value until otherwise initialized.

When invoking a function, no white space can be placed between the function name and the opening parenthesis. Function calls can be nested and recursive calls can be made upon functions. Upon return from any nested or recursive function call, the values of all of the calling function's parameters shall be unchanged, except for array parameters passed by reference. The **return** statement can be used to return a value. If a **return** statement appears outside of a function definition, the behavior is undefined.

In the function definition, <newline>s shall be optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a *pattern-action* pair is allowed.

Grammar

The grammar in this section and the lexical conventions in the following section shall together describe the syntax for *awk* programs. The general conventions for this style of grammar are described in [Section 1.3](#) (on page 2235). A valid program can be represented as the non-terminal symbol *program* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

```
%token NAME NUMBER STRING ERE
%token FUNC_NAME /* Name followed by '(' without white space. */
```

```

76288      /* Keywords */
76289      %token      Begin      End
76290      /*          'BEGIN' 'END'                                     */
76291
76291      %token      Break      Continue      Delete      Do      Else
76292      /*          'break' 'continue' 'delete' 'do' 'else' */
76293
76293      %token      Exit      For      Function      If      In
76294      /*          'exit' 'for' 'function' 'if' 'in'         */
76295
76295      %token      Next      Print      Printf      Return      While
76296      /*          'next' 'print' 'printf' 'return' 'while' */
76297
76297      /* Reserved function names */
76298      %token BUILTIN_FUNC_NAME
76299          /* One token for the following:
76300          * atan2 cos sin exp log sqrt int rand srand
76301          * gsub index length match split sprintf sub
76302          * substr tolower toupper close system
76303          */
76304      %token GETLINE
76305          /* Syntactically different from other built-ins. */
76306
76306      /* Two-character tokens. */
76307      %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
76308      /*      '+='      '-='      '*='      '/='      '%='      '^=' */
76309
76309      %token OR      AND      NO_MATCH      EQ      LE      GE      NE      INCR      DECR      APPEND
76310      /*      '||' ' &&' '!~' '==' '<=' '>=' '!=' '++' '--' '>>' */
76311
76311      /* One-character tokens. */
76312      %token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
76313      %token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='
76314
76314      %start program
76315      %%
76316
76316      program      : item_list
76317                  | actionless_item_list
76318                  ;
76319
76319      item_list    : newline_opt
76320                  | actionless_item_list item terminator
76321                  | item_list item terminator
76322                  | item_list action terminator
76323                  ;
76324
76324      actionless_item_list : item_list pattern terminator
76325                          | actionless_item_list pattern terminator
76326                          ;
76327
76327      item         : pattern action
76328                  | Function NAME      '(' param_list_opt ')'
76329                  | newline_opt action
76330                  | Function FUNC_NAME '(' param_list_opt ')'
76331                  | newline_opt action
76332                  ;
76333
76333      param_list_opt : /* empty */
76334                  | param_list
76335                  ;

```

```

76336 param_list      : NAME
76337                   | param_list ',' NAME
76338                   ;
76339 pattern          : Begin
76340                   | End
76341                   | expr
76342                   | expr ',' newline_opt expr
76343                   ;
76344 action           : '{' newline_opt                '}'
76345                   | '{' newline_opt terminated_statement_list '}'
76346                   | '{' newline_opt unterminated_statement_list '}'
76347                   ;
76348 terminator       : terminator ';'
76349                   | terminator NEWLINE
76350                   | ';'
76351                   | NEWLINE
76352                   ;
76353 terminated_statement_list : terminated_statement
76354                   | terminated_statement_list terminated_statement
76355                   ;
76356 unterminated_statement_list : unterminated_statement
76357                   | terminated_statement_list unterminated_statement
76358                   ;
76359 terminated_statement : action newline_opt
76360                   | If '(' expr ')' newline_opt terminated_statement
76361                   | If '(' expr ')' newline_opt terminated_statement
76362                   | Else newline_opt terminated_statement
76363                   | While '(' expr ')' newline_opt terminated_statement
76364                   | For '(' simple_statement_opt ';'
76365                   |   expr_opt ';' simple_statement_opt ')' newline_opt
76366                   |   terminated_statement
76367                   | For '(' NAME In NAME ')' newline_opt
76368                   |   terminated_statement
76369                   | ';' newline_opt
76370                   | terminatable_statement NEWLINE newline_opt
76371                   | terminatable_statement ';' newline_opt
76372                   ;
76373 unterminated_statement : terminatable_statement
76374                   | If '(' expr ')' newline_opt unterminated_statement
76375                   | If '(' expr ')' newline_opt terminated_statement
76376                   |   Else newline_opt unterminated_statement
76377                   | While '(' expr ')' newline_opt unterminated_statement
76378                   | For '(' simple_statement_opt ';'
76379                   |   expr_opt ';' simple_statement_opt ')' newline_opt
76380                   |   unterminated_statement
76381                   | For '(' NAME In NAME ')' newline_opt
76382                   |   unterminated_statement
76383                   ;
76384 terminatable_statement : simple_statement
76385                   | Break

```

```

76386         | Continue
76387         | Next
76388         | Exit expr_opt
76389         | Return expr_opt
76390         | Do newline_opt terminated_statement While '(' expr ')'
76391         ;

76392 simple_statement_opt : /* empty */
76393         | simple_statement
76394         ;

76395 simple_statement : Delete NAME '[' expr_list ']'
76396         | expr
76397         | print_statement
76398         ;

76399 print_statement : simple_print_statement
76400         | simple_print_statement output_redirection
76401         ;

76402 simple_print_statement : Print print_expr_list_opt
76403         | Print '(' multiple_expr_list ')'
76404         | Printf print_expr_list
76405         | Printf '(' multiple_expr_list ')'
76406         ;

76407 output_redirection : '>' expr
76408         | APPEND expr
76409         | '|' expr
76410         ;

76411 expr_list_opt : /* empty */
76412         | expr_list
76413         ;

76414 expr_list : expr
76415         | multiple_expr_list
76416         ;

76417 multiple_expr_list : expr ',' newline_opt expr
76418         | multiple_expr_list ',' newline_opt expr
76419         ;

76420 expr_opt : /* empty */
76421         | expr
76422         ;

76423 expr : unary_expr
76424         | non_unary_expr
76425         ;

76426 unary_expr : '+' expr
76427         | '-' expr
76428         | unary_expr '^' expr
76429         | unary_expr '*' expr
76430         | unary_expr '/' expr
76431         | unary_expr '%' expr
76432         | unary_expr '+' expr
76433         | unary_expr '-' expr
76434         | unary_expr non_unary_expr

```

```

76435 | unary_expr '<'      expr
76436 | unary_expr LE      expr
76437 | unary_expr NE      expr
76438 | unary_expr EQ      expr
76439 | unary_expr '>'      expr
76440 | unary_expr GE      expr
76441 | unary_expr '~'     expr
76442 | unary_expr NO_MATCH expr
76443 | unary_expr In NAME
76444 | unary_expr AND newline_opt expr
76445 | unary_expr OR  newline_opt expr
76446 | unary_expr '?' expr ':' expr
76447 | unary_input_function
76448 | ;
76449 | non_unary_expr : '(' expr ')'
76450 |                '!' expr
76451 |                non_unary_expr '^'      expr
76452 |                non_unary_expr '*'      expr
76453 |                non_unary_expr '/'      expr
76454 |                non_unary_expr '%'      expr
76455 |                non_unary_expr '+'      expr
76456 |                non_unary_expr '-'      expr
76457 |                non_unary_expr '~'      non_unary_expr
76458 |                non_unary_expr '<'      expr
76459 |                non_unary_expr LE      expr
76460 |                non_unary_expr NE      expr
76461 |                non_unary_expr EQ      expr
76462 |                non_unary_expr '>'      expr
76463 |                non_unary_expr GE      expr
76464 |                non_unary_expr '~'     expr
76465 |                non_unary_expr NO_MATCH expr
76466 |                non_unary_expr In NAME
76467 |                '(' multiple_expr_list ')' In NAME
76468 |                non_unary_expr AND newline_opt expr
76469 |                non_unary_expr OR  newline_opt expr
76470 |                non_unary_expr '?' expr ':' expr
76471 |                NUMBER
76472 |                STRING
76473 |                lvalue
76474 |                ERE
76475 |                lvalue INCR
76476 |                lvalue DECR
76477 |                INCR lvalue
76478 |                DECR lvalue
76479 |                lvalue POW_ASSIGN expr
76480 |                lvalue MOD_ASSIGN expr
76481 |                lvalue MUL_ASSIGN expr
76482 |                lvalue DIV_ASSIGN expr
76483 |                lvalue ADD_ASSIGN expr
76484 |                lvalue SUB_ASSIGN expr
76485 |                lvalue '=' expr
76486 |                FUNC_NAME '(' expr_list_opt ')'
76487 |                /* no white space allowed before '(' */
76488 |                BUILTIN_FUNC_NAME '(' expr_list_opt ')'

```

```

76489         | BUILTIN_FUNC_NAME
76490         | non_unary_input_function
76491         ;

76492     print_expr_list_opt : /* empty */
76493         | print_expr_list
76494         ;

76495     print_expr_list : print_expr
76496         | print_expr_list ',' newline_opt print_expr
76497         ;

76498     print_expr      : unary_print_expr
76499         | non_unary_print_expr
76500         ;

76501     unary_print_expr : '+' print_expr
76502         | '-' print_expr
76503         | unary_print_expr '^'      print_expr
76504         | unary_print_expr '*'      print_expr
76505         | unary_print_expr '/'      print_expr
76506         | unary_print_expr '%'      print_expr
76507         | unary_print_expr '+'      print_expr
76508         | unary_print_expr '-'      print_expr
76509         | unary_print_expr          non_unary_print_expr
76510         | unary_print_expr '~'      print_expr
76511         | unary_print_expr NO_MATCH print_expr
76512         | unary_print_expr In NAME
76513         | unary_print_expr AND newline_opt print_expr
76514         | unary_print_expr OR  newline_opt print_expr
76515         | unary_print_expr '?' print_expr ':' print_expr
76516         ;

76517     non_unary_print_expr : '(' expr ')'
76518         | '!' print_expr
76519         | non_unary_print_expr '^'      print_expr
76520         | non_unary_print_expr '*'      print_expr
76521         | non_unary_print_expr '/'      print_expr
76522         | non_unary_print_expr '%'      print_expr
76523         | non_unary_print_expr '+'      print_expr
76524         | non_unary_print_expr '-'      print_expr
76525         | non_unary_print_expr          non_unary_print_expr
76526         | non_unary_print_expr '~'      print_expr
76527         | non_unary_print_expr NO_MATCH print_expr
76528         | non_unary_print_expr In NAME
76529         | '(' multiple_expr_list ')' In NAME
76530         | non_unary_print_expr AND newline_opt print_expr
76531         | non_unary_print_expr OR  newline_opt print_expr
76532         | non_unary_print_expr '?' print_expr ':' print_expr
76533         | NUMBER
76534         | STRING
76535         | lvalue
76536         | ERE
76537         | lvalue INCR
76538         | lvalue DECR
76539         | INCR lvalue
76540         | DECR lvalue

```

```

76541         | lvalue POW_ASSIGN print_expr
76542         | lvalue MOD_ASSIGN print_expr
76543         | lvalue MUL_ASSIGN print_expr
76544         | lvalue DIV_ASSIGN print_expr
76545         | lvalue ADD_ASSIGN print_expr
76546         | lvalue SUB_ASSIGN print_expr
76547         | lvalue '=' print_expr
76548         | FUNC_NAME '(' expr_list_opt ')'
76549         | /* no white space allowed before '(' */
76550         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
76551         | BUILTIN_FUNC_NAME
76552         ;

76553     lvalue      : NAME
76554                 | NAME '[' expr_list '['
76555                 | '$' expr
76556                 ;

76557     non_unary_input_function : simple_get
76558                             | simple_get '<' expr
76559                             | non_unary_expr '|' simple_get
76560                             ;

76561     unary_input_function : unary_expr '|' simple_get
76562                         ;

76563     simple_get          : GETLINE
76564                         | GETLINE lvalue
76565                         ;

76566     newline_opt        : /* empty */
76567                         | newline_opt NEWLINE
76568                         ;

```

This grammar has several ambiguities that shall be resolved as follows:

- Operator precedence and associativity shall be as described in [Table 4-1](#) (on page 2374).
- In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that would satisfy the grammar.
- In some contexts, a slash ('/') that is used to surround an ERE could also be the division operator. This shall be resolved in such a way that wherever the division operator could appear, a slash is assumed to be the division operator. (There is no unary division operator.)

Each expression in an *awk* program shall conform to the precedence and associativity rules, even when this is not needed to resolve an ambiguity. For example, because '\$' has higher precedence than '++', the string "\$x++--" is not a valid *awk* expression, even though it is unambiguously parsed by the grammar as "\$ (x++) --".

One convention that might not be obvious from the formal grammar is where <newline>s are acceptable. There are several obvious placements such as terminating a statement, and a backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s without backslashes can follow a comma, an open brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```

76587     { print $1,
76588         $2 }

```


76589

Lexical Conventions

76590

The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as follows:

76591

76592

1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a given point.

76593

76594

2. A comment shall consist of any characters beginning with the number sign character and terminated by, but excluding the next occurrence of, a <newline>. Comments shall have no effect, except to delimit lexical tokens.

76595

76596

76597

3. The <newline> shall be recognized as the token **NEWLINE**.

76598

4. A backslash character immediately followed by a <newline> shall have no effect.

76599

76600

76601

76602

76603

76604

76605

76606

76607

5. The token **STRING** shall represent a string constant. A string constant shall begin with the character ' '. Within a string constant, a backslash character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 107) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequences in Table 4-2 (on page 2379) shall be recognized. A <newline> shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character ' ' after the one that begins the string constant. The value of the string shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting ' ' characters.

76608

76609

76610

76611

76612

76613

76614

76615

76616

76617

6. The token **ERE** represents an extended regular expression constant. An ERE constant shall begin with the slash character. Within an ERE constant, a backslash character shall be considered to begin an escape sequence as specified in the table in XBD Chapter 5 (on page 107). In addition, the escape sequences in Table 4-2 (on page 2379) shall be recognized. The application shall ensure that a <newline> does not occur within an ERE constant. An ERE constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the ERE constant. The extended regular expression represented by the ERE constant shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting slash characters.

76618

76619

7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE** tokens.

76620

76621

76622

8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall be equivalent to either of the tokens **floating-constant** or **integer-constant** as specified by the ISO C standard, with the following exceptions:

76623

76624

- a. An integer constant cannot begin with 0x or include the hexadecimal digits 'a', 'b', 'c', 'd', 'e', 'f', 'A', 'B', 'C', 'D', 'E', or 'F'.

76625

76626

- b. The value of an integer constant beginning with 0 shall be taken in decimal rather than octal.

76627

- c. An integer constant cannot include a suffix ('u', 'U', 'l', or 'L').

76628

- d. A floating constant cannot include a suffix ('f', 'F', 'l', or 'L').

76629

76630

If the value is too large or too small to be representable (see Section 1.1.2, on page 2231), the behavior is undefined.

76631

76632

76633

9. A sequence of underscores, digits, and alphabetic characters from the portable character set (see XBD Section 6.1, on page 111), beginning with an underscore or alphabetic, shall be considered a word.

76634 10. The following words are keywords that shall be recognized as individual tokens; the
76635 name of the token is the same as the keyword:

76636 **BEGIN** **delete** **END** **function** **in** **printf**
76637 **break** **do** **exit** **getline** **next** **return**
76638 **continue** **else** **for** **if** **print** **while**

76639 11. The following words are names of built-in functions and shall be recognized as the token
76640 **BUILTIN_FUNC_NAME**:

76641 **atan2** **gsub** **log** **split** **sub** **toupper**
76642 **close** **index** **match** **sprintf** **substr**
76643 **cos** **int** **rand** **sqrt** **system**
76644 **exp** **length** **sin** **srand** **tolower**

76645 The above-listed keywords and names of built-in functions are considered reserved
76646 words.

76647 12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in
76648 function and is not followed immediately (without any delimiters) by the ' (' character.

76649 13. The token **FUNC_NAME** shall consist of a word that is not a keyword or a name of a
76650 built-in function, followed immediately (without any delimiters) by the ' (' character.
76651 The ' (' character shall not be included as part of the token.

76652 14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	NO_MATCH	!~
SUB_ASSIGN	-=	EQ	==
MUL_ASSIGN	*=	LE	<=
DIV_ASSIGN	/=	GE	>=
MOD_ASSIGN	%=	NE	!=
POW_ASSIGN	^=	INCR	++
OR	 	DECR	--
AND	&&	APPEND	>>

76662 15. The following single characters shall be recognized as tokens whose names are the
76663 character:

76664 <newline> { } () [] , ; + - * % ^ ! > < | ? : ~ \$ =

76665 There is a lexical ambiguity between the token **ERE** and the tokens **/'/'** and **DIV_ASSIGN**.
76666 When an input sequence begins with a slash character in any syntactic context where the token
76667 **/'/'** or **DIV_ASSIGN** could appear as the next token in a valid program, the longer of those two
76668 tokens that can be recognized shall be recognized. In any other syntactic context where the token
76669 **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

76670 EXIT STATUS

76671 The following exit values shall be returned:

76672 0 All input files were processed successfully.

76673 >0 An error occurred.

76674 The exit status can be altered within the program by using an **exit** expression.

76675
76676
76677
76678
76679
76680
76681
76682
76683
76684
76685
76686
76687
76688
76689
76690
76691
76692
76693
76694
76695
76696
76697
76698
76699
76700
76701
76702
76703
76704
76705
76706
76707
76708
76709
76710
76711
76712
76713
76714
76715

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a diagnostic message to standard error and terminate without any further action.

If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk* program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

APPLICATION USAGE

The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with bytes.

Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, it is often necessary to use parentheses to enforce the proper evaluation precedence.

EXAMPLES

The *awk* program specified in the command line is most easily specified within single-quotes (for example, '*program*') for applications using *sh*, because *awk* programs commonly contain characters that are special to the shell, including double-quotes. In the cases where an *awk* program contains single-quote characters, it is usually easiest to specify most of the program as strings within single-quotes concatenated by the shell with quoted single-quote characters. For example:

```
awk '/'\''/ { print "quote:", $0 }'
```

prints all lines from the standard input containing a single-quote character, prefixed with *quote:*.

The following are examples of simple *awk* programs:

1. Write to the standard output all input lines for which field 3 is greater than 5:

```
$3 > 5
```

2. Write every tenth line:

```
(NR % 10) == 0
```

3. Write any line with a substring matching the regular expression:

```
/(G|D)(2[0-9][[:alpha:]]*)/
```

4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits and characters. This example uses character classes **digit** and **alpha** to match language-independent digit and alphabetic characters respectively:

```
/(G|D)([:digit:][:alpha:]]*)/
```

5. Write any line in which the second field matches the regular expression and the fourth field does not:

```
$2 ~ /xyz/ && $4 !~ /xyz/
```

6. Write any line in which the second field contains a backslash:

```
$2 ~ /\\"/>

```

7. Write any line in which the second field contains a backslash. Note that backslash escapes are interpreted twice; once in lexical processing of the string and once in processing the regular expression:

```
$2 ~ "\\\"/>

```

- 76716 8. Write the second to the last and the last field in each line. Separate the fields by a colon:
- 76717 `{OFS=":";print $(NF-1), $NF}`
- 76718 9. Write the line number and number of fields in each line. The three strings representing
- 76719 the line number, the colon, and the number of fields are concatenated and that string is
- 76720 written to standard output:
- 76721 `{print NR ":" NF}`
- 76722 10. Write lines longer than 72 characters:
- 76723 `length($0) > 72`
- 76724 11. Write the first two fields in opposite order separated by **OFS**:
- 76725 `{ print $2, $1 }`
- 76726 12. Same, with input fields separated by a comma or <space>s and <tab>s, or both:
- 76727 `BEGIN { FS = ", [\t]* | [\t]+" }`
- 76728 `{ print $2, $1 }`
- 76729 13. Add up the first column, print sum, and average:
- 76730 `{s += $1 }`
- 76731 `END {print "sum is ", s, " average is", s/NR}`
- 76732 14. Write fields in reverse order, one per line (many lines out for each line in):
- 76733 `{ for (i = NF; i > 0; --i) print $i }`
- 76734 15. Write all lines between occurrences of the strings **start** and **stop**:
- 76735 `/start/, /stop/`
- 76736 16. Write all lines whose first field is different from the previous one:
- 76737 `$1 != prev { print; prev = $1 }`
- 76738 17. Simulate *echo*:
- 76739 `BEGIN {`
- 76740 `for (i = 1; i < ARGV; ++i)`
- 76741 `printf("%s%s", ARGV[i], i==ARGV-1?"\n":" ")`
- 76742 `}`
- 76743 18. Write the path prefixes contained in the *PATH* environment variable, one per line:
- 76744 `BEGIN {`
- 76745 `n = split (ENVIRON["PATH"], path, ":")`
- 76746 `for (i = 1; i <= n; ++i)`
- 76747 `print path[i]`
- 76748 `}`
- 76749 19. If there is a file named **input** containing page headers of the form:
- 76750 `Page #`
- 76751 and a file named **program** that contains:
- 76752 `/Page/ { $2 = n++; }`
- 76753 `{ print }`
- 76754 then the command line:
- 76755 `awk -f program n=5 input`

76756 prints the file **input**, filling in page numbers starting at 5.

76757 RATIONALE

76758 This description is based on the new *awk*, “nawk”, (see the referenced *The AWK Programming*
76759 *Language*), which introduced a number of new features to the historical *awk*:

- 76760 1. New keywords: **delete**, **do**, **function**, **return**
- 76761 2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
- 76762 3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
- 76763 4. New expression operators: **?**, **:**, **„**, **^**
- 76764 5. The **FS** variable and the third argument to **split**, now treated as extended regular
76765 expressions.
- 76766 6. The operator precedence, changed to more closely match the C language. Two examples
76767 of code that operate differently are:

```
76768 while ( n /= 10 > 1 ) ...
76769 if (!"wk" ~ /bwk/) ...
```

76770 Several features have been added based on newer implementations of *awk*:

- 76771 • Multiple instances of **-f *progfile*** are permitted.
- 76772 • The new option **-v *assignment***.
- 76773 • The new predefined variable **ENVIRON**.
- 76774 • New built-in functions **toupper** and **tolower**.
- 76775 • More formatting capabilities are added to **printf** to match the ISO C standard.

76776 The overall *awk* syntax has always been based on the C language, with a few features from the
76777 shell command language and other sources. Because of this, it is not completely compatible with
76778 any other language, which has caused confusion for some users. It is not the intent of the
76779 standard developers to address such issues. A few relatively minor changes toward making the
76780 language more compatible with the ISO C standard were made; most of these changes are based
76781 on similar changes in recent implementations, as described above. There remain several C-
76782 language conventions that are not in *awk*. One of the notable ones is the comma operator, which
76783 is commonly used to specify multiple expressions in the C language **for** statement. Also, there
76784 are various places where *awk* is more restrictive than the C language regarding the type of
76785 expression that can be used in a given context. These limitations are due to the different features
76786 that the *awk* language does provide.

76787 Regular expressions in *awk* have been extended somewhat from historical implementations to
76788 make them a pure superset of extended regular expressions, as defined by POSIX.1-200x (see
76789 XBD Section 9.4, on page 174). The main extensions are internationalization features and
76790 interval expressions. Historical implementations of *awk* have long supported backslash escape
76791 sequences as an extension to extended regular expressions, and this extension has been retained
76792 despite inconsistency with other utilities. The number of escape sequences recognized in both
76793 extended regular expressions and strings has varied (generally increasing with time) among
76794 implementations. The set specified by POSIX.1-200x includes most sequences known to be
76795 supported by popular implementations and by the ISO C standard. One sequence that is not
76796 supported is hexadecimal value escapes beginning with ‘\x’. This would allow values
76797 expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because
76798 this syntax has a non-deterministic length, it does not permit the subsequent character to be a
76799 hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string
76800 concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for
76801 extended regular expressions (either lexical ERE tokens or strings used dynamically as regular

76802 expressions). Because of this limitation, the feature has not been added to POSIX.1-200x.

76803 When a string variable is used in a context where an extended regular expression normally
76804 appears (where the lexical token ERE is used in the grammar) the string does not contain the
76805 literal slashes.

76806 Some versions of *awk* allow the form:

```
76807 func name(args, ... ) { statements }
```

76808 This has been deprecated by the authors of the language, who asked that it not be specified.

76809 Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN**
76810 action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This
76811 behavior has not been documented, and it was not believed that it was necessary to standardize
76812 it.

76813 The specification of conversions between string and numeric values is much more detailed than
76814 in the documentation of historical implementations or in the referenced *The AWK Programming*
76815 *Language*. Although most of the behavior is designed to be intuitive, the details are necessary to
76816 ensure compatible behavior from different implementations. This is especially important in
76817 relational expressions since the types of the operands determine whether a string or numeric
76818 comparison is performed. From the perspective of an application developer, it is usually
76819 sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating
76820 a null string) when the type of an expression does not obviously match what is needed. The
76821 intent has been to specify historical practice in almost all cases. The one exception is that, in
76822 historical implementations, variables and constants maintain both string and numeric values
76823 after their original value is converted by any use. This means that referencing a variable or
76824 constant can have unexpected side effects. For example, with historical implementations the
76825 following program:

```
76826 {  
76827     a = "+2"  
76828     b = 2  
76829     if (NR % 2)  
76830         c = a + b  
76831     if (a == b)  
76832         print "numeric comparison"  
76833     else  
76834         print "string comparison"  
76835 }
```

76836 would perform a numeric comparison (and output numeric comparison) for each odd-
76837 numbered line, but perform a string comparison (and output string comparison) for each even-
76838 numbered line. POSIX.1-200x ensures that comparisons will be numeric if necessary. With
76839 historical implementations, the following program:

```
76840 BEGIN {  
76841     OFMT = "%e"  
76842     print 3.14  
76843     OFMT = "%f"  
76844     print 3.14  
76845 }
```

76846 would output "3.140000e+00" twice, because in the second **print** statement the constant
76847 "3.14" would have a string value from the previous conversion. POSIX.1-200x requires that the
76848 output of the second **print** statement be "3.140000". The behavior of historical
76849 implementations was seen as too unintuitive and unpredictable.

76850 It was pointed out that with the rules contained in early drafts, the following script would print

```

76851     nothing:
76852     BEGIN {
76853         y[1.5] = 1
76854         OFMT = "%e"
76855         print y[1.5]
76856     }

```

76857 Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to
 76858 affecting output conversions of numbers to strings and **CONVFMT** is used for internal
 76859 conversions, such as comparisons or array indexing. The default value is the same as that for
 76860 **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it
 76861 will receive the historical behavior associated with internal string conversions.

76862 The POSIX *awk* lexical and syntactic conventions are specified more formally than in other
 76863 sources. Again the intent has been to specify historical practice. One convention that may not be
 76864 obvious from the formal grammar as in other verbal descriptions is where <newline>s are
 76865 acceptable. There are several obvious placements such as terminating a statement, and a
 76866 backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s
 76867 without backslashes can follow a comma, an open brace, a logical AND operator ("&&"), a
 76868 logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an
 76869 **if**, **for**, or **while** statement. For example:

```

76870     { print $1,
76871         $2 }

```

76872 The requirement that *awk* add a trailing <newline> to the program argument text is to simplify
 76873 the grammar, making it match a text file in form. There is no way for an application or test suite
 76874 to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

76875 POSIX.1-200x requires several changes from historical implementations in order to support
 76876 internationalization. Probably the most subtle of these is the use of the decimal-point character,
 76877 defined by the *LC_NUMERIC* category of the locale, in representations of floating-point
 76878 numbers. This locale-specific character is used in recognizing numeric input, in converting
 76879 between strings and numeric values, and in formatting output. However, regardless of locale,
 76880 the period character (the decimal-point character of the POSIX locale) is the decimal-point
 76881 character recognized in processing *awk* programs (including assignments in command line
 76882 arguments). This is essentially the same convention as the one used in the ISO C standard. The
 76883 difference is that the C language includes the *setlocale()* function, which permits an application
 76884 to modify its locale. Because of this capability, a C application begins executing with its locale set
 76885 to the C locale, and only executes in the environment-specified locale after an explicit call to
 76886 *setlocale()*. However, adding such an elaborate new feature to the *awk* language was seen as
 76887 inappropriate for POSIX.1-200x. It is possible to execute an *awk* program explicitly in any desired
 76888 locale by setting the environment in the shell.

76889 The undefined behavior resulting from NULs in extended regular expressions allows future
 76890 extensions for the GNU *gawk* program to process binary data.

76891 The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic
 76892 errors) is undefined because it was considered overly limiting on implementations to specify. In
 76893 most cases such errors can be expected to produce a diagnostic and a non-zero exit status.
 76894 However, some implementations may choose to extend the language in ways that make use of
 76895 certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but
 76896 otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect
 76897 in some implementations. Also, different implementations might detect a given error during an
 76898 initial parsing of the program (before reading any input files) while others might detect it when
 76899 executing the program after reading some input. Implementors should be aware that diagnosing
 76900 errors as early as possible and producing useful diagnostics can ease debugging of applications,

76901 and thus make an implementation more usable.

76902 The unspecified behavior from using multi-character **RS** values is to allow possible future
76903 extensions based on extended regular expressions used for record separators. Historical
76904 implementations take the first character of the string and ignore the others.

76905 Unspecified behavior when *split(string,array,<null>)* is used is to allow a proposed future
76906 extension that would split up a string into an array of individual characters.

76907 In the context of the **getline** function, equally good arguments for different precedences of the |
76908 and < operators can be made. Historical practice has been that:

```
76909 getline < "a" "b"
```

76910 is parsed as:

```
76911 ( getline < "a" ) "b"
```

76912 although many would argue that the intent was that the file **ab** should be read. However:

```
76913 getline < "x" + 1
```

76914 parses as:

```
76915 getline < ( "x" + 1 )
```

76916 Similar problems occur with the | version of **getline**, particularly in combination with \$. For
76917 example:

```
76918 $"echo hi" | getline
```

76919 (This situation is particularly problematic when used in a **print** statement, where the |**getline**
76920 part might be a redirection of the **print**.)

76921 Since in most cases such constructs are not (or at least should not) be used (because they have a
76922 natural ambiguity for which there is no conventional parsing), the meaning of these constructs
76923 has been made explicitly unspecified. (The effect is that a conforming application that runs into
76924 the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual
76925 uses of such constructs.

76926 Grammars can be written that would cause an error under these circumstances. Where
76927 backwards-compatibility is not a large consideration, implementors may wish to use such
76928 grammars.

76929 Some historical implementations have allowed some built-in functions to be called without an
76930 argument list, the result being a default argument list chosen in some “reasonable” way. Use of
76931 **length** as a synonym for **length(\$0)** is the only one of these forms that is thought to be widely
76932 known or widely used; this particular form is documented in various places (for example, most
76933 historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as
76934 legitimate practice. With this exception, default argument lists have always been undocumented
76935 and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-
76936 defined functions. They add no useful functionality and preclude possible future extensions that
76937 might need to name functions without calling them. Not standardizing them seems the simplest
76938 course. The standard developers considered that **length** merited special treatment, however,
76939 since it has been documented in the past and sees possibly substantial use in historical
76940 programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the
76941 obsolescent marking for XSI-conforming implementations and many otherwise conforming
76942 applications depend on this feature.

76943 In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive
76944 backslash characters should be used in the string to ensure a single backslash will precede the
76945 ampersand when the resultant string is passed to the function. (For example, to specify one
76946 literal ampersand in the replacement string, use **gsub(ERE, "\\&")**.)

76947 Historically the only special character in the *repl* argument of **sub** and **gsub** string functions was
 76948 the ampersand ('&') character and preceding it with the backslash character was used to turn
 76949 off its special meaning.

76950 The description in the ISO POSIX-2:1993 standard introduced behavior such that the backslash
 76951 character was another special character and it was unspecified whether there were any other
 76952 special characters. This description introduced several portability problems, some of which are
 76953 described below, and so it has been replaced with the more historical description. Some of the
 76954 problems include:

- 76955 • Historically, to create the replacement string, a script could use **gsub(ERE, "\\&")**, but
 76956 with the ISO POSIX-2:1993 standard wording, it was necessary to use **gsub(ERE,**
 76957 **"\\\\&")**. Backslash characters are doubled here because all string literals are subject to
 76958 lexical analysis, which would reduce each pair of backslash characters to a single backslash
 76959 before being passed to **gsub**.
- 76960 • Since it was unspecified what the special characters were, for portable scripts to guarantee
 76961 that characters are printed literally, each character had to be preceded with a backslash.
 76962 (For example, a portable script had to use **gsub(ERE, "\\h\\i")** to produce a replacement
 76963 string of "hi".)

76964 The description for comparisons in the ISO POSIX-2:1993 standard did not properly describe
 76965 historical practice because of the way numeric strings are compared as numbers. The current
 76966 rules cause the following code:

```
76967 if (0 == "000")
76968     print "strange, but true"
76969 else
76970     print "not true"
```

76971 to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this
 76972 is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

76973 To fix this problem, the definition of *numeric string* was enhanced to include only those values
 76974 obtained from specific circumstances (mostly external sources) where it is not possible to
 76975 determine unambiguously whether the value is intended to be a string or a numeric.

76976 Variables that are assigned to a numeric string shall also be treated as a numeric string. (For
 76977 example, the notion of a numeric string can be propagated across assignments.) In comparisons,
 76978 all variables having the uninitialized value are to be treated as a numeric operand evaluating to
 76979 the numeric value zero.

76980 Uninitialized variables include all types of variables including scalars, array elements, and
 76981 fields. The definition of an uninitialized value in [Variables and Special Variables](#) (on page 2377)
 76982 is necessary to describe the value placed on uninitialized variables and on fields that are valid
 76983 (for example, < \$NF) but have no characters in them and to describe how these variables are to
 76984 be used in comparisons. A valid field, such as \$1, that has no characters in it can be obtained
 76985 from an input line of "\t\t" when FS='\t'. Historically, the comparison (\$1<10) was done
 76986 numerically after evaluating \$1 to the value zero.

76987 The phrase "... also shall have the numeric value of the numeric string" was removed from
 76988 several sections of the ISO POSIX-2:1993 standard because it specifies an unnecessary
 76989 implementation detail. It is not necessary for POSIX.1-200x to specify that these objects be
 76990 assigned two different values. It is only necessary to specify that these objects may evaluate to
 76991 two different values depending on context.

76992 The description of numeric string processing is based on the behavior of the *atof()* function in
 76993 the ISO C standard. While it is not a requirement for an implementation to use this function,
 76994 many historical implementations of *awk* do. In the ISO C standard, floating-point constants use a
 76995 period as a decimal point character for the language itself, independent of the current locale, but

76996 the *atof()* function and the associated *strtod()* function use the decimal point character of the
 76997 current locale when converting strings to numeric values. Similarly in *awk*, floating-point
 76998 constants in an *awk* script use a period independent of the locale, but input strings use the
 76999 decimal point character of the locale.

77000 FUTURE DIRECTIONS

77001 None.

77002 SEE ALSO

77003 [Section 1.3](#) (on page 2235), *grep*, *lex*, *sed*

77004 XBD [Chapter 5](#) (on page 107), [Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Chapter 9](#) (on
 77005 page 167), [Section 12.2](#) (on page 201)

77006 XSH *atof()*, *exec*, *popen()*, *setlocale()*, *strtod()*

77007 CHANGE HISTORY

77008 First released in Issue 2.

77009 Issue 5

77010 The FUTURE DIRECTIONS section is added.

77011 Issue 6

77012 The *awk* utility is aligned with the IEEE P1003.2b draft standard.

77013 The normative text is reworded to avoid use of the term “must” for application requirements.

77014 IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence “An occurrence of two
 77015 consecutive backslashes shall be interpreted as just a single literal backslash character.” into the
 77016 description of the **sub** string function.

77017 Issue 7

77018 PASC Interpretation 1003.2-1992 #107 (SD5-XCU-ERN-73) is applied, updating the description of
 77019 the **OFS** variable.

77020 SD5-XCU-ERN-79 is applied, restoring the horizontal lines to [Table 4-1](#) (on page 2374), and
 77021 SD5-XCU-ERN-80 is applied, changing the order of some table entries.

77022 SD5-XCU-ERN-87 is applied, updating the descriptive text of the Grammar.

77023 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

77024 **NAME**
 77025 `basename` — return non-directory portion of a pathname

77026 **SYNOPSIS**
 77027 `basename string [suffix]`

77028 **DESCRIPTION**
 77029 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.265](#) (on page 70).
 77030 The string *string* shall be converted to the filename corresponding to the last pathname
 77031 component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be
 77032 done by performing actions equivalent to the following steps in order:

- 77033 1. If *string* is a null string, it is unspecified whether the resulting string is `.'` or a null
 77034 string. In either case, skip steps 2 through 6.
- 77035 2. If *string* is `"/"`, it is implementation-defined whether steps 3 to 6 are skipped or
 77036 processed.
- 77037 3. If *string* consists entirely of slash characters, *string* shall be set to a single slash character.
 77038 In this case, skip steps 4 to 6.
- 77039 4. If there are any trailing slash characters in *string*, they shall be removed.
- 77040 5. If there are any slash characters remaining in *string*, the prefix of *string* up to and
 77041 including the last slash character in *string* shall be removed.
- 77042 6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is
 77043 identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed
 77044 from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an
 77045 error if *suffix* is not found in *string*.

77046 The resulting string shall be written to standard output.

77047 **OPTIONS**
 77048 None.

77049 **OPERANDS**
 77050 The following operands shall be supported:

77051 *string* A string.
 77052 *suffix* A string.

77053 **STDIN**
 77054 Not used.

77055 **INPUT FILES**
 77056 None.

77057 **ENVIRONMENT VARIABLES**
 77058 The following environment variables shall affect the execution of *basename*:

77059 *LANG* Provide a default value for the internationalization variables that are unset or null.
 77060 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 77061 variables used to determine the values of locale categories.)

77062 *LC_ALL* If set to a non-empty string value, override the values of all the other
 77063 internationalization variables.

77064 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
77065 characters (for example, single-byte as opposed to multi-byte characters in
77066 arguments).

77067 *LC_MESSAGES*
77068 Determine the locale that should be used to affect the format and contents of
77069 diagnostic messages written to standard error.

77070 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

77071 **ASYNCHRONOUS EVENTS**

77072 Default.

77073 **STDOUT**

77074 The *basename* utility shall write a line to the standard output in the following format:

77075 "%s\n", <resulting string>

77076 **STDERR**

77077 The standard error shall be used only for diagnostic messages.

77078 **OUTPUT FILES**

77079 None.

77080 **EXTENDED DESCRIPTION**

77081 None.

77082 **EXIT STATUS**

77083 The following exit values shall be returned:

77084 0 Successful completion.

77085 >0 An error occurred.

77086 **CONSEQUENCES OF ERRORS**

77087 Default.

77088 **APPLICATION USAGE**

77089 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
77090 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the
77091 beginning of a pathname unless they can ensure that there are more or less than two or are
77092 prepared to deal with the implementation-defined consequences.

77093 **EXAMPLES**

77094 If the string *string* is a valid pathname:

77095 \$(basename "*string*")

77096 produces a filename that could be used to open the file named by *string* in the directory returned
77097 by:

77098 \$(dirname "*string*")

77099 If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be
77100 a valid filename. The *basename* utility is not expected to make any judgements about the validity
77101 of *string* as a pathname; it just follows the specified algorithm to produce a result string.

77102 The following shell script compiles */usr/src/cmd/cat.c* and moves the output to a file named **cat**
77103 in the current directory when invoked with the argument */usr/src/cmd/cat* or with the argument
77104 */usr/src/cmd/cat.c*:

77105 c99 \$(dirname "\$1")/\$(basename "\$1" .c).c

77106 mv a.out \$(basename "\$1" .c)

77107
77108
77109
77110
77111
77112
77113
77114
77115
77116
77117
77118
77119
77120
77121
77122
77123
77124
77125
77126
77127

RATIONALE

The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid pathname:

```
$(basename "string")
```

would be a valid filename for the file in the directory:

```
$(dirname "string")
```

This would not work for the early proposal versions of these utilities due to the way it specified handling of trailing slashes.

Since the definition of *pathname* specifies implementation-defined behavior for pathnames starting with two slash characters, this volume of POSIX.1-200x specifies similar implementation-defined behavior for the *basename* and *dirname* utilities.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.5](#) (on page 2249), *dirname*

[XBD Section 3.265](#) (on page 70), [Chapter 8](#) (on page 159)

+

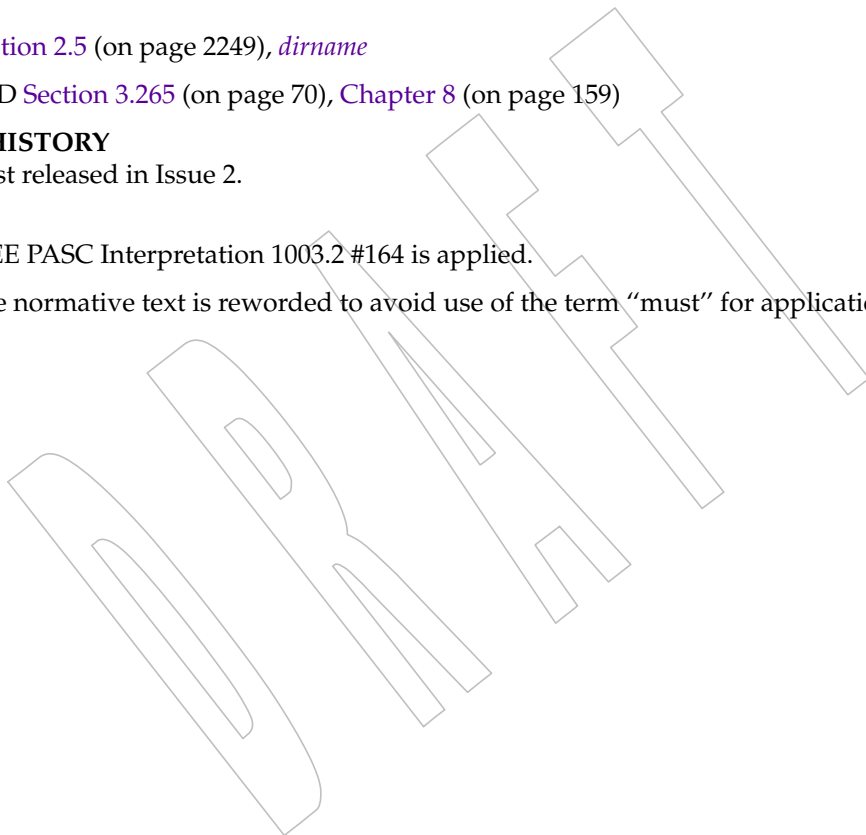
CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE PASC Interpretation 1003.2 #164 is applied.

The normative text is reworded to avoid use of the term “must” for application requirements.



77128 **NAME**77129 `batch` — schedule commands to be executed in a batch queue77130 **SYNOPSIS**77131 `batch`77132 **DESCRIPTION**77133 The *batch* utility shall read commands from standard input and schedule them for execution in a
77134 batch queue. It shall be the equivalent of the command:77135 `at -q b -m now`77136 where queue *b* is a special *at* queue, specifically for batch jobs. Batch jobs shall be submitted to
77137 the batch queue with no time constraints and shall be run by the system using algorithms, based
77138 on unspecified factors, that may vary with each invocation of *batch*.77139 XSI Users shall be permitted to use *batch* if their name appears in the file **at.allow** which is located in
77140 an implementation-defined directory. If that file does not exist, the file **at.deny**, which is located
77141 in an implementation-defined directory, shall be checked to determine whether the user shall be
77142 denied access to *batch*. If neither file exists, only a process with the appropriate privileges shall
77143 be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted.
77144 The **at.allow** and **at.deny** files shall consist of one user name per line.77145 **OPTIONS**

77146 None.

77147 **OPERANDS**

77148 None.

77149 **STDIN**77150 The standard input shall be a text file consisting of commands acceptable to the shell command
77151 language described in [Chapter 2](#) (on page 2245).77152 **INPUT FILES**77153 XSI The text files **at.allow** and **at.deny**, which are located in an implementation-defined directory,
77154 shall contain zero or more user names, one per line, of users who are, respectively, authorized or
77155 denied access to the *at* and *batch* utilities.77156 **ENVIRONMENT VARIABLES**77157 The following environment variables shall affect the execution of *batch*:77158 **LANG** Provide a default value for the internationalization variables that are unset or null. |
77159 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
77160 variables used to determine the values of locale categories.)77161 **LC_ALL** If set to a non-empty string value, override the values of all the other
77162 internationalization variables.77163 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
77164 characters (for example, single-byte as opposed to multi-byte characters in
77165 arguments and input files).77166 **LC_MESSAGES**77167 Determine the locale that should be used to affect the format and contents of
77168 diagnostic messages written to standard error and informative messages written to
77169 standard output.

77170		<i>LC_TIME</i>	Determine the format and contents for date and time strings written by <i>batch</i> .
77171	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
77172		<i>SHELL</i>	Determine the name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, <i>sh</i> shall be used. If it is set to a value other than a name for <i>sh</i> , the implementation shall do one of the following: use that shell; use <i>sh</i> ; use the login shell from the user database; any of the preceding accompanied by a warning diagnostic about which was chosen.
77173			
77174			
77175			
77176			
77177		<i>TZ</i>	Determine the timezone. The job shall be submitted for execution at the time specified by <i>timespec</i> or <i>-t time</i> relative to the timezone specified by the <i>TZ</i> variable. If <i>timespec</i> specifies a timezone, it overrides <i>TZ</i> . If <i>timespec</i> does not specify a timezone and <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
77178			
77179			
77180			
77181			

ASYNCHRONOUS EVENTS

77182 Default.

77183

STDOUT

77184 When standard input is a terminal, prompts of unspecified format for each line of the user input described in the STDIN section may be written to standard output.

77185

77186

STDERR

77187 The following shall be written to standard error when a job has been successfully submitted:

77188

77189

"job %s at %s\n", *at_job_id*, <*date*>

77190

where *date* shall be equivalent in format to the output of:

77191

`date +"%a %b %e %T %Y"`

77192

77193

The date and time written shall be adjusted so that they appear in the timezone of the user (as determined by the *TZ* variable).

77194

77195

Neither this, nor warning messages concerning the selection of the command interpreter, are considered a diagnostic that changes the exit status.

77196

Diagnostic messages, if any, shall be written to standard error.

OUTPUT FILES

77197 None.

77198

EXTENDED DESCRIPTION

77199 None.

77200

EXIT STATUS

77201 The following exit values shall be returned:

77202

77203

0 Successful completion.

77204

>0 An error occurred.

CONSEQUENCES OF ERRORS

77205 The job shall not be scheduled.

77206

77207

APPLICATION USAGE

77208

It may be useful to redirect standard output within the specified commands.

77209

EXAMPLES

77210

1. This sequence can be used at a terminal:

77211

```
batch
```

77212

```
sort < file >outfile
```

77213

```
EOT
```

77214

2. This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

77215

```
batch <<!
```

77216

```
diff file1 file2 2>&1 >outfile | mailx mygroup
```

77217

```
!
```

77218

77219

RATIONALE

77220

Early proposals described *batch* in a manner totally separated from *at*, even though the historical model treated it almost as a synonym for *at -qb*. A number of features were added to list and control batch work separately from those in *at*. Upon further reflection, it was decided that the benefit of this did not merit the change to the historical interface.

77221

77222

77223

77224

The *-m* option was included on the equivalent *at* command because it is historical practice to mail results to the submitter, even if all job-produced output is redirected. As explained in the RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling delays), despite some historical systems where *at now* would have been considered an error.

77225

77226

77227

77228

FUTURE DIRECTIONS

77229

None.

77230

SEE ALSO

77231

at

77232

XBD [Chapter 8](#) (on page 159)

77233

CHANGE HISTORY

77234

First released in Issue 2.

77235

Issue 6

77236

This utility is marked as part of the User Portability Utilities option.

77237

The NAME is changed to align with the IEEE P1003.2b draft standard.

77238

The normative text is reworded to avoid use of the term “must” for application requirements.

77239

Issue 7

77240

The *batch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

77241

77242

SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced by the *batch* utility.

77243

- 77244 **NAME**
- 77245 bc — arbitrary-precision arithmetic language
- 77246 **SYNOPSIS**
- 77247 bc [-l] [*file...*]
- 77248 **DESCRIPTION**
- 77249 The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files
- 77250 given, then read from the standard input. If the standard input and standard output to *bc* are
- 77251 attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing
- 77252 behavioral constraints described in the following sections.
- 77253 **OPTIONS**
- 77254 The *bc* utility shall conform to XBD [Section 12.2](#) (on page 201).
- 77255 The following option shall be supported:
- 77256 -l (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the
- 77257 default zero; see the EXTENDED DESCRIPTION section.
- 77258 **OPERANDS**
- 77259 The following operand shall be supported:
- 77260 *file* A pathname of a text file containing *bc* program statements. After all *files* have
- 77261 been read, *bc* shall read the standard input.
- 77262 **STDIN**
- 77263 See the INPUT FILES section.
- 77264 **INPUT FILES**
- 77265 Input files shall be text files containing a sequence of comments, statements, and function
- 77266 definitions that shall be executed as they are read.
- 77267 **ENVIRONMENT VARIABLES**
- 77268 The following environment variables shall affect the execution of *bc*:
- 77269 LANG Provide a default value for the internationalization variables that are unset or null. |
- 77270 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
- 77271 variables used to determine the values of locale categories.)
- 77272 LC_ALL If set to a non-empty string value, override the values of all the other
- 77273 internationalization variables.
- 77274 LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
- 77275 characters (for example, single-byte as opposed to multi-byte characters in
- 77276 arguments and input files).
- 77277 LC_MESSAGES
- 77278 Determine the locale that should be used to affect the format and contents of
- 77279 diagnostic messages written to standard error.
- 77280 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 77281 **ASYNCHRONOUS EVENTS**
- 77282 Default.

77283 **STDOUT**

77284 The output of the *bc* utility shall be controlled by the program read, and consist of zero or more
 77285 lines containing the value of all executed expressions without assignments. The radix and
 77286 precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the
 77287 EXTENDED DESCRIPTION section.

77288 **STDERR**

77289 The standard error shall be used only for diagnostic messages.

77290 **OUTPUT FILES**

77291 None.

77292 **EXTENDED DESCRIPTION**77293 **Grammar**

77294 The grammar in this section and the lexical conventions in the following section shall together
 77295 describe the syntax for *bc* programs. The general conventions for this style of grammar are
 77296 described in [Section 1.3](#) (on page 2235). A valid program can be represented as the non-terminal
 77297 symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax
 77298 description.

```

77299 %token      EOF NEWLINE STRING LETTER NUMBER
77300 %token      MUL_OP
77301 /*         '*', '/', '%' */
77302 %token      ASSIGN_OP
77303 /*         '=', '+=', '-=', '*=', '/=', '%=', '^=' */
77304 %token      REL_OP
77305 /*         '==', '<=', '>=', '!=', '<', '>' */
77306 %token      INCR_DECR
77307 /*         '++', '--' */
77308 %token      Define      Break      Quit      Length
77309 /*         'define', 'break', 'quit', 'length' */
77310 %token      Return      For      If      While      Sqrt
77311 /*         'return', 'for', 'if', 'while', 'sqrt' */
77312 %token      Scale      Ibase      Obase      Auto
77313 /*         'scale', 'ibase', 'obase', 'auto' */
77314 %start      program
77315 %%
77316 program      : EOF
77317              | input_item program
77318              ;
77319 input_item   : semicolon_list NEWLINE
77320              | function
77321              ;
77322 semicolon_list : /* empty */
77323              | statement
77324              | semicolon_list ';' statement
77325              | semicolon_list ';'
77326              ;
77327 statement_list : /* empty */

```

```

77328 | statement
77329 | statement_list NEWLINE
77330 | statement_list NEWLINE statement
77331 | statement_list ';'
77332 | statement_list ';' statement
77333 | ;
77334 statement : expression
77335 | STRING
77336 | Break
77337 | Quit
77338 | Return
77339 | Return '(' return_expression ')'
77340 | For '(' expression ';'
77341 |     relational_expression ';'
77342 |     expression ')' statement
77343 | If '(' relational_expression ')' statement
77344 | While '(' relational_expression ')' statement
77345 | '{' statement_list '}'
77346 | ;
77347 function : Define LETTER '(' opt_parameter_list ')'
77348 |     '{' NEWLINE opt_auto_define_list
77349 |     statement_list '}'
77350 | ;
77351 opt_parameter_list : /* empty */
77352 | parameter_list
77353 | ;
77354 parameter_list : LETTER
77355 | define_list ',' LETTER
77356 | ;
77357 opt_auto_define_list : /* empty */
77358 | Auto define_list NEWLINE
77359 | Auto define_list ';'
77360 | ;
77361 define_list : LETTER
77362 | LETTER '[' ']'
77363 | define_list ',' LETTER
77364 | define_list ',' LETTER '[' ']'
77365 | ;
77366 opt_argument_list : /* empty */
77367 | argument_list
77368 | ;
77369 argument_list : expression
77370 | LETTER '[' ']' ',' argument_list
77371 | ;
77372 relational_expression : expression
77373 | expression REL_OP expression
77374 | ;
77375 return_expression : /* empty */
77376 | expression

```

```

77377                                     ;
77378 expression                          : named_expression
77379                                     | NUMBER
77380                                     | '(' expression ')'
77381                                     | LETTER '(' opt_argument_list ')'
77382                                     | '-' expression
77383                                     | expression '+' expression
77384                                     | expression '-' expression
77385                                     | expression MUL_OP expression
77386                                     | expression '^' expression
77387                                     | INCR_DECR named_expression
77388                                     | named_expression INCR_DECR
77389                                     | named_expression ASSIGN_OP expression
77390                                     | Length '(' expression ')'
77391                                     | Sqrt '(' expression ')'
77392                                     | Scale '(' expression ')'
77393                                     ;
77394 named_expression                      : LETTER
77395                                     | LETTER '[' expression ']'
77396                                     | Scale
77397                                     | Ibase
77398                                     | Obase
77399                                     ;

```

77400 Lexical Conventions in bc

77401 The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as
77402 follows:

- 77403 1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a
77404 given point.
- 77405 2. A comment shall consist of any characters beginning with the two adjacent characters
77406 `"/*"` and terminated by the next occurrence of the two adjacent characters `"*/"`.
77407 Comments shall have no effect except to delimit lexical tokens.
- 77408 3. The `<newline>` shall be recognized as the token **NEWLINE**.
- 77409 4. The token **STRING** shall represent a string constant; it shall consist of any characters
77410 beginning with the double-quote character (`'"`) and terminated by another occurrence
77411 of the double-quote character. The value of the string is the sequence of all characters
77412 between, but not including, the two double-quote characters. All characters shall be taken
77413 literally from the input, and there is no way to specify a string containing a double-quote
77414 character. The length of the value of each string shall be limited to `{BC_STRING_MAX}`
77415 bytes.
- 77416 5. A `<blank>` shall have no effect except as an ordinary character if it appears within a
77417 **STRING** token, or to delimit a lexical token other than **STRING**.
- 77418 6. The combination of a backslash character immediately followed by a `<newline>` shall
77419 have no effect other than to delimit lexical tokens with the following exceptions:
 - 77420 • It shall be interpreted as the character sequence `"\<newline>"` in **STRING** tokens.
 - 77421 • It shall be ignored as part of a multi-line **NUMBER** token.

- 77422 7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the
77423 following grammar:
- ```
77424 NUMBER : integer
77425 | '.' integer
77426 | integer '.'
77427 | integer '.' integer
77428 ;

77429 integer : digit
77430 | integer digit
77431 ;

77432 digit : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
77433 | 8 | 9 | A | B | C | D | E | F
77434 ;
```
- 77435 8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by  
77436 the value of the internal register **ibase** (described below). Each of the **digit** characters  
77437 shall have the value from 0 to 15 in the order listed here, and the period character shall  
77438 represent the radix point. The behavior is undefined if digits greater than or equal to the  
77439 value of **ibase** appear in the token. However, note the exception for single-digit values  
77440 being assigned to **ibase** and **obase** themselves, in [Operations in bc](#) (on page 2414).
- 77441 9. The following keywords shall be recognized as tokens:
- ```
77442 auto      ibase      length      return      while
77443 break     if          obase       scale
77444 define    for          quit        sqrt
```
- 77445 10. Any of the following characters occurring anywhere except within a keyword shall be
77446 recognized as the token **LETTER**:
- ```
77447 a b c d e f g h i j k l m n o p q r s t u v w x y z
```
- 77448 11. The following single-character and two-character sequences shall be recognized as the  
77449 token **ASSIGN\_OP**:
- ```
77450 = += -= *= /= %= ^=
```
- 77451 12. If an '=' character, as the beginning of a token, is followed by a '-' character with no
77452 intervening delimiter, the behavior is undefined.
- 77453 13. The following single-characters shall be recognized as the token **MUL_OP**:
- ```
77454 * / %
```
- 77455 14. The following single-character and two-character sequences shall be recognized as the  
77456 token **REL\_OP**:
- ```
77457 == <= >= != < >
```
- 77458 15. The following two-character sequences shall be recognized as the token **INCR_DECR**:
- ```
77459 ++ --
```
- 77460 16. The following single characters shall be recognized as tokens whose names are the  
77461 character:
- ```
77462 <newline> ( ) , + - ; [ ] ^ { }
```
- 77463 17. The token **EOF** is returned when the end of input is reached.

77464

Operations in bc

77465

77466

77467

77468

77469

77470

77471

There are three kinds of identifiers: ordinary identifiers, array identifiers, and function identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed by square brackets (" []"). An array subscript is required except in an argument or auto list. Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall begin at zero so an array is indexed from 0 to {BC_DIM_MAX}-1. Subscripts shall be truncated to integers. The application shall ensure that function identifiers are followed by parentheses, possibly enclosing arguments. The three types of identifiers do not conflict.

77472

77473

77474

The following table summarizes the rules for precedence and associativity of all operators. Operators on the same line shall have the same precedence; rows are in order of decreasing precedence.

77475

Table 4-3 Operators in *bc*

77476

77477

77478

77479

77480

77481

77482

77483

Operator	Associativity
++, --	N/A
unary -	N/A
^	Right to left
*, /, %	Left to right
+, binary -	Left to right
=, +=, -=, *=, /=, %=, ^=	Right to left
==, <=, >=, !=, <, >	None

77484

77485

Each expression or named expression has a *scale*, which is the number of decimal digits that shall be maintained as the fractional portion of the expression.

77486

77487

77488

77489

Named expressions are places where values are stored. Named expressions shall be valid on the left side of an assignment. The value of a named expression shall be the value stored in the place named. Simple identifiers and array elements are named expressions; they have an initial value of zero and an initial scale of zero.

77490

77491

77492

77493

77494

77495

77496

The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an expression consisting of the name of one of these registers shall be zero; values assigned to any of these registers are truncated to integers. The **scale** register shall contain a global value used in computing the scale of expressions (as described below). The value of the register **scale** is limited to $0 \leq \text{scale} \leq \{\text{BC_SCALE_MAX}\}$ and shall have a default value of zero. The **ibase** and **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be limited to:

77497

$$2 \leq \text{ibase} \leq 16$$

77498

The value of **obase** shall be limited to:

77499

$$2 \leq \text{obase} \leq \{\text{BC_BASE_MAX}\}$$

77500

77501

77502

77503

77504

When either **ibase** or **obase** is assigned a single **digit** value from the list in [Lexical Conventions in bc](#) (on page 2412), the value shall be assumed in hexadecimal. (For example, **ibase=A** sets to base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall have initial values of 10.

77505

77506

77507

Internal computations shall be conducted as if in decimal, regardless of the input and output bases, to the specified number of decimal digits. When an exact result is not achieved (for example, **scale=0**; 3.2/1), the result shall be truncated.

77508

77509

For all values of **obase** specified by this volume of POSIX.1-200x, *bc* shall output numeric values by performing each of the following steps in order:

- 77510 1. If the value is less than zero, a hyphen ('-') character shall be output.
- 77511 2. One of the following is output, depending on the numerical value:
- 77512 • If the absolute value of the numerical value is greater than or equal to one, the
 - 77513 integer portion of the value shall be output as a series of digits appropriate to **obase**
 - 77514 (as described below), most significant digit first. The most significant non-zero digit
 - 77515 shall be output next, followed by each successively less significant digit.
 - 77516 • If the absolute value of the numerical value is less than one but greater than zero
 - 77517 and the scale of the numerical value is greater than zero, it is unspecified whether
 - 77518 the character 0 is output.
 - 77519 • If the numerical value is zero, the character 0 shall be output.
- 77520 3. If the scale of the value is greater than zero and the numeric value is not zero, a period
- 77521 character shall be output, followed by a series of digits appropriate to **obase** (as described
- 77522 below) representing the most significant portion of the fractional part of the value. If *s*
- 77523 represents the scale of the value being output, the number of digits output shall be *s* if
- 77524 **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s*
- 77525 if **obase** is less than 10. For **obase** values other than 10, this should be the number of
- 77526 digits needed to represent a precision of 10^s .

77527 For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

77528 0 1 2 3 4 5 6 7 8 9 A B C D E F

77529 which represent the values zero to 15, inclusive, respectively.

77530 For bases greater than 16, each digit shall be written as a separate multi-digit decimal number.

77531 Each digit except the most significant fractional digit shall be preceded by a single <space>. For

77532 bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-

77533 digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be

77534 written as:

77535 Δ01Δ15Δ24

77536 and in base 125, as:

77537 Δ008Δ024

77538 Very large numbers shall be split across lines with 70 characters per line in the POSIX locale;

77539 other locales may split at different character boundaries. Lines that are continued shall end with

77540 a backslash ('\').

77541 A function call shall consist of a function name followed by parentheses containing a comma-

77542 separated list of expressions, which are the function arguments. A whole array passed as an

77543 argument shall be specified by the array name followed by empty square brackets. All function

77544 arguments shall be passed by value. As a result, changes made to the formal parameters shall

77545 have no effect on the actual arguments. If the function terminates by executing a **return**

77546 statement, the value of the function shall be the value of the expression in the parentheses of the

77547 **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

77548 The result of **sqrt(expression)** shall be the square root of the expression. The result shall be

77549 truncated in the least significant decimal place. The scale of the result shall be the scale of the

77550 expression or the value of **scale**, whichever is larger.

77551 The result of **length(expression)** shall be the total number of significant decimal digits in the

77552 expression. The scale of the result shall be zero.

77553 The result of **scale(expression)** shall be the scale of the expression. The scale of the result shall be

77554 zero.

77555 A numeric constant shall be an expression. The scale shall be the number of digits that follow the
77556 radix point in the input representing the constant, or zero if no radix point appears.

77557 The sequence (*expression*) shall be an expression with the same value and scale as *expression*.
77558 The parentheses can be used to alter the normal precedence.

77559 The semantics of the unary and binary operators are as follows:

77560 *-expression*
77561 The result shall be the negative of the *expression*. The scale of the result shall be the scale of
77562 *expression*.

77563 The unary increment and decrement operators shall not modify the scale of the named
77564 expression upon which they operate. The scale of the result shall be the scale of that named
77565 expression.

77566 *++named-expression*
77567 The named expression shall be incremented by one. The result shall be the value of the
77568 named expression after incrementing.

77569 *--named-expression*
77570 The named expression shall be decremented by one. The result shall be the value of the
77571 named expression after decrementing.

77572 *named-expression++*
77573 The named expression shall be incremented by one. The result shall be the value of the
77574 named expression before incrementing.

77575 *named-expression--*
77576 The named expression shall be decremented by one. The result shall be the value of the
77577 named expression before decrementing.

77578 The exponentiation operator, circumflex ('^'), shall bind right to left.

77579 *expression^expression*
77580 The result shall be the first *expression* raised to the power of the second *expression*. If the
77581 second *expression* is not an integer, the behavior is undefined. If *a* is the scale of the left
77582 *expression* and *b* is the absolute value of the right *expression*, the scale of the result shall be:
77583 if $b \geq 0$ $\min(a * b, \max(\text{scale}, a))$ if $b < 0$ scale

77584 The multiplicative operators ('*', '/', '%') shall bind left to right.

77585 *expression*expression*
77586 The result shall be the product of the two expressions. If *a* and *b* are the scales of the two
77587 expressions, then the scale of the result shall be:
77588 $\min(a+b, \max(\text{scale}, a, b))$

77589 *expression/expression*
77590 The result shall be the quotient of the two expressions. The scale of the result shall be the
77591 value of **scale**.

77592 *expression%expression*
77593 For expressions *a* and *b*, *a%b* shall be evaluated equivalent to the steps:
77594 1. Compute *a/b* to current scale.
77595 2. Use the result to compute:
77596 $a - (a / b) * b$
77597 to scale:
77598 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

77599 The scale of the result shall be:

77600 $\max(\text{scale} + \text{scale}(b), \text{scale}(a))$

77601 When **scale** is zero, the '**%**' operator is the mathematical remainder operator.

77602 The additive operators ('+', '-') shall bind left to right.

77603 *expression+expression*

77604 The result shall be the sum of the two expressions. The scale of the result shall be the

77605 maximum of the scales of the expressions.

77606 *expression-expression*

77607 The result shall be the difference of the two expressions. The scale of the result shall be the

77608 maximum of the scales of the expressions.

77609 The assignment operators ('=', '+=', '-=', '*=', '/=', '%=', '^=') shall bind right to left.

77610 *named-expression=expression*

77611 This expression shall result in assigning the value of the expression on the right to the

77612 named expression on the left. The scale of both the named expression and the result shall be

77613 the scale of *expression*.

77614 The compound assignment forms:

77615 *named-expression <operator>= expression*

77616 shall be equivalent to:

77617 *named-expression=named-expression <operator> expression*

77618 except that the *named-expression* shall be evaluated only once.

77619 Unlike all other operators, the relational operators ('<', '>', '<=', '>=', '==', '!=') shall be

77620 only valid as the object of an **if**, **while**, or inside a **for** statement.

77621 *expression1<expression2*

77622 The relation shall be true if the value of *expression1* is strictly less than the value of

77623 *expression2*.

77624 *expression1>expression2*

77625 The relation shall be true if the value of *expression1* is strictly greater than the value of

77626 *expression2*.

77627 *expression1<=expression2*

77628 The relation shall be true if the value of *expression1* is less than or equal to the value of

77629 *expression2*.

77630 *expression1>=expression2*

77631 The relation shall be true if the value of *expression1* is greater than or equal to the value of

77632 *expression2*.

77633 *expression1==expression2*

77634 The relation shall be true if the values of *expression1* and *expression2* are equal.

77635 *expression1!=expression2*

77636 The relation shall be true if the values of *expression1* and *expression2* are unequal.

77637 There are only two storage classes in *bc*: global and automatic (local). Only identifiers that are

77638 local to a function need be declared with the **auto** command. The arguments to a function shall

77639 be local to the function. All other identifiers are assumed to be global and available to all

77640 functions. All identifiers, global and local, have initial values of zero. Identifiers declared as **auto**

77641 shall be allocated on entry to the function and released on returning from the function. They

77642 therefore do not retain values between function calls. Auto arrays shall be specified by the array

77643 name followed by empty square brackets. On entry to a function, the old values of the names
 77644 that appear as parameters and as automatic variables shall be pushed onto a stack. Until the
 77645 function returns, reference to these names shall refer only to the new values.

77646 References to any of these names from other functions that are called from this function also
 77647 refer to the new value until one of those functions uses the same name for a local variable.

77648 When a statement is an expression, unless the main operator is an assignment, execution of the
 77649 statement shall write the value of the expression followed by a <newline>.

77650 When a statement is a string, execution of the statement shall write the value of the string.

77651 Statements separated by semicolons or <newline>s shall be executed sequentially. In an
 77652 interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical
 77653 production:

77654 `input_item : semicolon_list NEWLINE`

77655 the sequential list of statements making up the **semicolon_list** shall be executed immediately
 77656 and any output produced by that execution shall be written without any delay due to buffering.

77657 In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

77658 The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested;
 77659 each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the
 77660 *relation* is false, execution shall resume after *statement*.

77661 A **for** statement(**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
77662 first-expression
77663 while (relation) {
77664     statement
77665     last-expression
77666 }
```

77667 The application shall ensure that all three expressions are present.

77668 The **break** statement shall cause termination of a **for** or **while** statement.

77669 The **auto** statement (**auto** *identifier* [*identifier*] ...) shall cause the values of the identifiers to be
 77670 pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers
 77671 shall be specified by following the array name by empty square brackets. The application shall
 77672 ensure that the **auto** statement is the first statement in a function definition.

77673 A **define** statement:

```
77674 define LETTER ( opt_parameter_list ) {
77675     opt_auto_define_list
77676     statement_list
77677 }
```

77678 defines a function named **LETTER**. If a function named **LETTER** was previously defined, the
 77679 **define** statement shall replace the previous definition. The expression:

77680 `LETTER (opt_argument_list)`

77681 shall invoke the function named **LETTER**. The behavior is undefined if the number of
 77682 arguments in the invocation does not match the number of parameters in the definition.
 77683 Functions shall be defined before they are invoked. A function shall be considered to be defined
 77684 within its own body, so recursive calls are valid. The values of numeric constants within a
 77685 function shall be interpreted in the base specified by the value of the **ibase** register when the
 77686 function is invoked.

77687 The **return** statements (**return** and **return**(*expression*)) shall cause termination of a function,

77688 popping of its auto variables, and specification of the result of the function. The first form shall
 77689 be equivalent to **return(0)**. The value and scale of the result returned by the function shall be the
 77690 value and scale of the expression returned.

77691 The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement
 77692 occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

77693 The following functions shall be defined when the **-I** option is specified:

77694 **s**(*expression*)
 77695 Sine of argument in radians.

77696 **c**(*expression*)
 77697 Cosine of argument in radians.

77698 **a**(*expression*)
 77699 Arctangent of argument.

77700 **l**(*expression*)
 77701 Natural logarithm of argument.

77702 **e**(*expression*)
 77703 Exponential function of argument.

77704 **j**(*expression*, *expression*)
 77705 Bessel function of integer order.

77706 The scale of the result returned by these functions shall be the value of the **scale** register at the
 77707 time the function is invoked. The value of the **scale** register after these functions have completed
 77708 their execution shall be the same value it had upon invocation. The behavior is undefined if any
 77709 of these functions is invoked with an argument outside the domain of the mathematical
 77710 function.

77711 EXIT STATUS

77712 The following exit values shall be returned:

77713 0 All input files were processed successfully.

77714 *unspecified* An error occurred.

77715 CONSEQUENCES OF ERRORS

77716 If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic
 77717 message to standard error and terminate without any further action.

77718 In an interactive invocation of *bc*, the utility should print an error message and recover following
 77719 any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined
 77720 behavior.

77721 APPLICATION USAGE

77722 Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

77723 For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has
 77724 occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by
 77725 interactive users (who can react to error messages) or by application programs that can
 77726 somehow validate the answers returned as not including error messages.

77727 The *bc* utility always uses the period (**' . '**) character to represent a radix point, regardless of any
 77728 decimal-point character specified as part of the current locale. In languages like C or *awk*, the
 77729 period character is used in program source, so it can be portable and unambiguous, while the
 77730 locale-specific character is used in input and output. Because there is no distinction between
 77731 source and input in *bc*, this arrangement would not be possible. Using the locale-specific
 77732 character in *bc*'s input would introduce ambiguities into the language; consider the following
 77733 example in a locale with a comma as the decimal-point character:

```

77734     define f(a,b) {
77735         ...
77736     }
77737     ...
77738     f(1,2,3)

```

77739 Because of such ambiguities, the period character is used in input. Having input follow different
77740 conventions from output would be confusing in either pipeline usage or interactive usage, so the
77741 period is also used in output.

77742 EXAMPLES

77743 In the shell, the following assigns an approximation of the first ten digits of ' π ' to the variable *x*:

```
77744 x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

77745 The following *bc* program prints the same approximation of ' π ', with a label, to standard
77746 output:

```

77747     scale = 10
77748     "pi equals "
77749     104348 / 33215

```

77750 The following defines a function to compute an approximate value of the exponential function
77751 (note that such a function is predefined if the `-l` option is specified):

```

77752     scale = 20
77753     define e(x){
77754         auto a, b, c, i, s
77755         a = 1
77756         b = 1
77757         s = 1
77758         for (i = 1; 1 == 1; i++){
77759             a = a*x
77760             b = b*i
77761             c = a/b
77762             if (c == 0) {
77763                 return(s)
77764             }
77765             s = s+c
77766         }
77767     }

```

77768 The following prints approximate values of the exponential function of the first ten integers:

```

77769     for (i = 1; i <= 10; ++i) {
77770         e(i)
77771     }

```

77772 RATIONALE

77773 The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to
77774 be part of this volume of POSIX.1-200x because *bc* was thought to have a more intuitive
77775 programmatic interface. Current implementations that implement *bc* using *dc* are expected to be
77776 compliant.

77777 The exit status for error conditions has been left unspecified for several reasons:

- 77778 • The *bc* utility is used in both interactive and non-interactive situations. Different exit codes
77779 may be appropriate for the two uses.

- 77780 • It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions,
77781 and syntax errors are all possibilities.
- 77782 • It is not clear what utility the exit status has.
- 77783 • In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with
77784 *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc*
77785 aborted.

77786 The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief
77787 that *bc file1 file2* is used most often when at least *file1* contains data/function
77788 declarations/initializations. Having *bc* continue with prerequisite files missing is probably not
77789 useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check
77790 all its files for accessibility before opening any of them.

77791 There was considerable debate on the appropriateness of the language accepted by *bc*. Several
77792 reviewers preferred to see either a pure subset of the C language or some changes to make the
77793 language more compatible with C. While the *bc* language has some obvious similarities to C, it
77794 has never claimed to be compatible with any version of C. An interpreter for a subset of C might
77795 be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility
77796 is known in historical practice, and it was not within the scope of this volume of POSIX.1-200x
77797 to define such a language and utility. If and when they are defined, it may be appropriate to
77798 include them in a future version of this standard. This left the following alternatives:

- 77799 1. Exclude any calculator language from this volume of POSIX.1-200x.

77800 The consensus of the standard developers was that a simple programmatic calculator
77801 language is very useful for both applications and interactive users. The only arguments
77802 for excluding any calculator were that it would become obsolete if and when a C-
77803 compatible one emerged, or that the absence would encourage the development of such a
77804 C-compatible one. These arguments did not sufficiently address the needs of current
77805 application developers.

- 77806 2. Standardize the historical *dc*, possibly with minor modifications.

77807 The consensus of the standard developers was that *dc* is a fundamentally less usable
77808 language and that that would be far too severe a penalty for avoiding the issue of being
77809 similar to but incompatible with C.

- 77810 3. Standardize the historical *bc*, possibly with minor modifications.

77811 This was the approach taken. Most of the proponents of changing the language would not
77812 have been satisfied until most or all of the incompatibilities with C were resolved. Since
77813 most of the changes considered most desirable would break historical applications and
77814 require significant modification to historical implementations, almost no modifications
77815 were made. The one significant modification that was made was the replacement of the
77816 historical *bc* assignment operators "*a*+", and so on, with the more modern "*a*=", and so
77817 on. The older versions are considered to be fundamentally flawed because of the lexical
77818 ambiguity in uses like *a*--1.

77819 In order to permit implementations to deal with backwards-compatibility as they see fit,
77820 the behavior of this one ambiguous construct was made undefined. (At least three
77821 implementations have been known to support this change already, so the degree of
77822 change involved should not be great.)

77823 The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of
77824 this operator for other values of **scale** is from historical implementations of *bc*, and has been
77825 maintained for the sake of historical applications despite its non-intuitive nature.

77826 Historical implementations permit setting **ibase** and **obase** to a broader range of values. This
77827 includes values less than 2, which were not seen as sufficiently useful to standardize. These

77828 implementations do not interpret input properly for values of **ibase** that are greater than 16. This
 77829 is because numeric constants are recognized syntactically, rather than lexically, as described in
 77830 this volume of POSIX.1-200x. They are built from lexical tokens of single hexadecimal digits and
 77831 periods. Since <blank>s between tokens are not visible at the syntactic level, it is not possible to
 77832 recognize the multi-digit “digits” used in the higher bases properly. The ability to recognize
 77833 input in these bases was not considered useful enough to require modifying these
 77834 implementations. Note that the recognition of numeric constants at the syntactic level is not a
 77835 problem with conformance to this volume of POSIX.1-200x, as it does not impact the behavior of
 77836 conforming applications (and correct *bc* programs). Historical implementations also accept input
 77837 with all of the digits ‘0’–‘9’ and ‘A’–‘F’ regardless of the value of **ibase**; since digits with
 77838 value greater than or equal to **ibase** are not really appropriate, the behavior when they appear is
 77839 undefined, except for the common case of:

```
77840 ibase=8;
77841     /* Process in octal base. */
77842     ...
77843 ibase=A
77844     /* Restore decimal base. */
```

77845 In some historical implementations, if the expression to be written is an uninitialized array
 77846 element, a leading <space> and/or up to four leading 0 characters may be output before the
 77847 character zero. This behavior is considered a bug; it is unlikely that any currently conforming
 77848 application relies on:

```
77849 echo 'b[3]' | bc
```

77850 returning 00000 rather than 0.

77851 Exact calculation of the number of fractional digits to output for a given value in a base other
 77852 than 10 can be computationally expensive. Historical implementations use a faster
 77853 approximation, and this is permitted. Note that the requirements apply only to values of **obase**
 77854 that this volume of POSIX.1-200x requires implementations to support (in particular, not to 1, 0,
 77855 or negative bases, if an implementation supports them as an extension).

77856 Historical implementations of *bc* did not allow array parameters to be passed as the last
 77857 parameter to a function. New implementations are encouraged to remove this restriction even
 77858 though it is not required by the grammar.

77859 FUTURE DIRECTIONS

77860 None.

77861 SEE ALSO

77862 [Section 1.3](#) (on page 2235), *awk*

77863 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

77864 CHANGE HISTORY

77865 First released in Issue 4.

77866 Issue 5

77867 The FUTURE DIRECTIONS section is added.

77868 Issue 6

77869 Updated to align with the IEEE P1003.2b draft standard, which included resolution of several
 77870 interpretations of the ISO POSIX-2: 1993 standard.

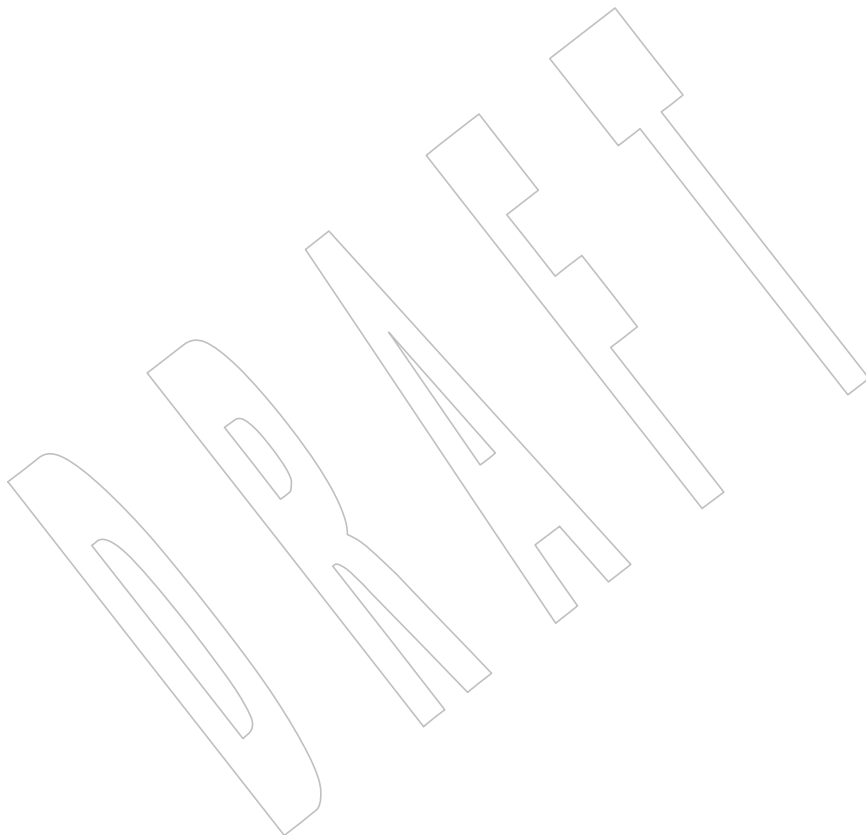
77871 The normative text is reworded to avoid use of the term “must” for application requirements.

77872

77873

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



77874 **NAME**
 77875 `bg` — run jobs in the background

77876 **SYNOPSIS**
 77877 UP `bg [job_id...]`

77878 **DESCRIPTION**
 77879 If job control is enabled (see the description of `set -m`), the `bg` utility shall resume suspended jobs
 77880 from the current environment (see [Section 2.12](#), on page 2277) by running them as background
 77881 jobs. If the job specified by `job_id` is already a running background job, the `bg` utility shall have
 77882 no effect and shall exit successfully.

77883 Using `bg` to place a job into the background shall cause its process ID to become “known in the
 77884 current shell execution environment”, as if it had been started as an asynchronous list; see
 77885 [Section 2.9.3.1](#) (on page 2266).

77886 **OPTIONS**
 77887 None.

77888 **OPERANDS**
 77889 The following operand shall be supported:
 77890 `job_id` Specify the job to be resumed as a background job. If no `job_id` operand is given,
 77891 the most recently suspended job shall be used. The format of `job_id` is described in
 77892 XBD [Section 3.202](#) (on page 61).

77893 **STDIN**
 77894 Not used.

77895 **INPUT FILES**
 77896 None.

77897 **ENVIRONMENT VARIABLES**
 77898 The following environment variables shall affect the execution of `bg`:
 77899 `LANG` Provide a default value for the internationalization variables that are unset or null.
 77900 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 77901 variables used to determine the values of locale categories.)

77902 `LC_ALL` If set to a non-empty string value, override the values of all the other
 77903 internationalization variables.

77904 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
 77905 characters (for example, single-byte as opposed to multi-byte characters in
 77906 arguments).

77907 `LC_MESSAGES`
 77908 Determine the locale that should be used to affect the format and contents of
 77909 diagnostic messages written to standard error.

77910 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

77911 **ASYNCHRONOUS EVENTS**
 77912 Default.

77913 **STDOUT**77914 The output of *bg* shall consist of a line in the format:

77915 "[%d] %s\n", <job-number>, <command>

77916 where the fields are as follows:

77917 <job-number> A number that can be used to identify the job to the *wait*, *fg*, and *kill* utilities. Using
77918 these utilities, the job can be identified by prefixing the job number with '% '.

77919 <command> The associated command that was given to the shell.

77920 **STDERR**

77921 The standard error shall be used only for diagnostic messages.

77922 **OUTPUT FILES**

77923 None.

77924 **EXTENDED DESCRIPTION**

77925 None.

77926 **EXIT STATUS**

77927 The following exit values shall be returned:

77928 0 Successful completion.

77929 >0 An error occurred.

77930 **CONSEQUENCES OF ERRORS**77931 If job control is disabled, the *bg* utility shall exit with an error and no job shall be placed in the
77932 background.77933 **APPLICATION USAGE**77934 A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see
77935 XBD Chapter 11 (on page 185). At that point, *bg* can put the job into the background. This is
77936 most effective when the job is expecting no terminal input and its output has been redirected to
77937 non-terminal files. A background job can be forced to stop when it has terminal output by
77938 issuing the command:77939 `stty tostop`

77940 A background job can be stopped with the command:

77941 `kill -s stop job ID`77942 The *bg* utility does not work as expected when it is operating in its own utility execution
77943 environment because that environment has no suspended jobs. In the following examples:77944 `... | xargs bg`
77945 `(bg)`77946 each *bg* operates in a different environment and does not share its parent shell's understanding
77947 of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.77948 **EXAMPLES**

77949 None.

77950 **RATIONALE**77951 The extensions to the shell specified in this volume of POSIX.1-200x have mostly been based on
77952 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
77953 based on the KornShell. The standard developers examined the characteristics of the C shell
77954 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
77955 the KornShell versions were selected for this volume of POSIX.1-200x to maintain a degree of
77956 uniformity with the rest of the KornShell features selected (such as the very popular command

77957

line editing features).

77958

The *bg* utility is expected to wrap its output if the output exceeds the number of display columns.

77959

FUTURE DIRECTIONS

77960

None.

77961

SEE ALSO

77962

[Section 2.9.3.1](#) (on page 2266), *fg*, *kill*, *jobs*, *wait*

77963

77964

[XBD Section 3.202](#) (on page 61), [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185)

+

77965

CHANGE HISTORY

77966

First released in Issue 4.

Issue 6

77967

This utility is marked as part of the User Portability Utilities option.

77968

77969

The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory in this version. This is a FIPS requirement.

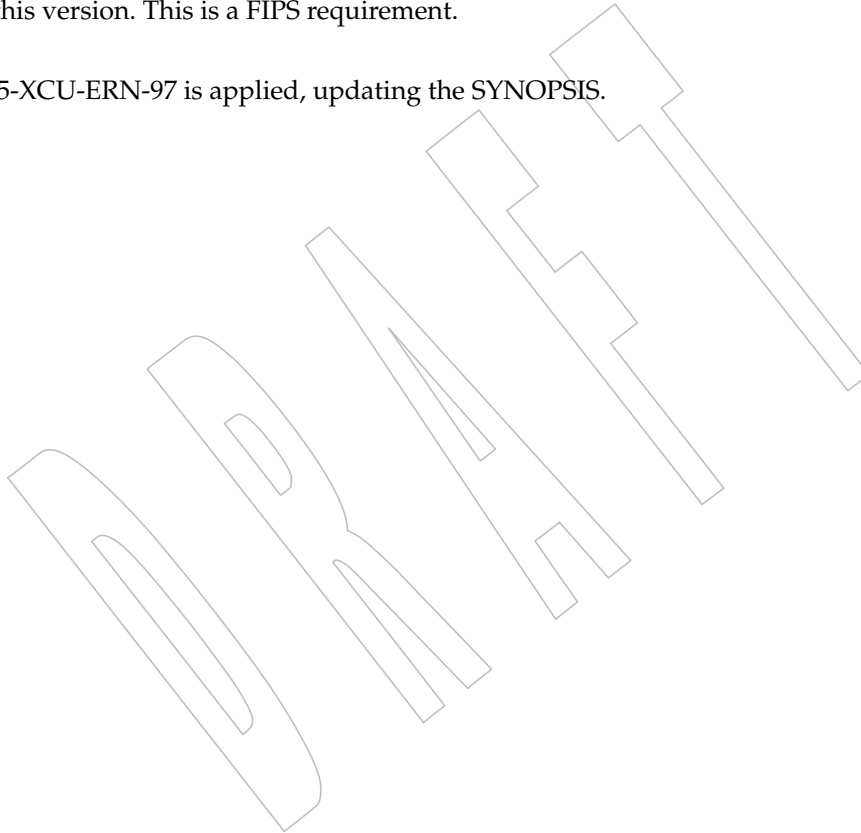
77970

Issue 7

77971

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

77972



77973 **NAME**
 77974 `c99` — compile standard C programs

77975 **SYNOPSIS**
 77976 CD `c99 [options...] pathname [pathname[-D directory`
 77977 `[-L directory] [-l library]]...`

77978 DESCRIPTION

77979 The `c99` utility is an interface to the standard C compilation system; it shall accept source code
 77980 conforming to the ISO C standard. The system conceptually consists of a compiler and link
 77981 editor. The input files referenced by *pathname* operands and `-I` option-arguments shall be
 77982 compiled and linked to produce an executable file. (It is unspecified whether the linking occurs
 77983 entirely within the operation of `c99`; some implementations may produce objects that are not
 77984 fully resolved until the file is executed.)

77985 If the `-c` option is specified, for all *pathname* operands of the form *file.c*, the files:

77986 `$(basename pathname .c).o`

77987 shall be created as the result of successful compilation. If the `-c` option is not specified, it is
 77988 unspecified whether such `.o` files are created or deleted for the *file.c* operands.

77989 If there are no options that prevent link editing (such as `-c` or `-E`), and all input files compile and
 77990 link without error, the resulting executable file shall be written according to the `-o outfile` option
 77991 (if present) or to the file `a.out`.

77992 The executable file shall be created as specified in [Section 1.1.1.4](#) (on page 2228), except that the
 77993 file permission bits shall be set to:

77994 `S_IRWXO | S_IRWXG | S_IRWXU`

77995 and the bits specified by the *umask* of the process shall be cleared.

77996 OPTIONS

77997 The `c99` utility shall conform to XBD [Section 12.2](#) (on page 201), except that:

- 77998 • Options can be interspersed with operands.
- 77999 • The order of specifying the `-I`, `-L`, and `-I` options, and the order of specifying `-I` options
 78000 with respect to *pathname* operands is significant.
- 78001 • Conforming applications shall specify each option separately; that is, grouping option
 78002 letters (for example, `-cO`) need not be recognized by all implementations.

78003 The following options shall be supported:

78004 `-c` Suppress the link-edit phase of the compilation, and do not remove any object files
 78005 that are produced.

78006 `-D name[=value]`

78007 Define *name* as if by a C-language `#define` directive. If no *=value* is given, a value of
 78008 1 shall be used. The `-D` option has lower precedence than the `-U` option. That is, if
 78009 *name* is used in both a `-U` and a `-D` option, *name* shall be undefined regardless of
 78010 the order of the options. Additional implementation-defined *names* may be
 78011 provided by the compiler. Implementations shall support at least 2 048 bytes of `-D`
 78012 definitions and 256 *names*.

- 78013 -E Copy C-language source files to standard output, expanding all preprocessor
78014 directives; no compilation shall be performed. If any operand is not a text file, the
78015 effects are unspecified.
- 78016 -g Produce symbolic information in the object or executable files; the nature of this +
78017 information is unspecified, and may be modified by implementation-defined +
78018 interactions with other options.
- 78019 -I *directory* Change the algorithm for searching for headers whose names are not absolute
78020 pathnames to look in the directory named by the *directory* pathname before looking
78021 in the usual places. Thus, headers whose names are enclosed in double-quotes (" ")
78022 shall be searched for first in the directory of the file with the #include line, then in
78023 directories named in -I options, and last in the usual places. For headers whose
78024 names are enclosed in angle brackets ("< >"), the header shall be searched for only
78025 in directories named in -I options and then in the usual places. Directories named
78026 in -I options shall be searched in the order specified. Implementations shall
78027 support at least ten instances of this option in a single c99 command invocation.
- 78028 -L *directory* Change the algorithm of searching for the libraries named in the -l objects to look
78029 in the directory named by the *directory* pathname before looking in the usual
78030 places. Directories named in -L options shall be searched in the order specified.
78031 Implementations shall support at least ten instances of this option in a single c99
78032 command invocation. If a directory specified by a -L option contains files with
78033 names starting with any of the strings "libc.", "libl.", "libpthread.",
78034 "libm.", "librt.", "libtrace.", "libxnet.", or "liby.", the results are
78035 unspecified.
- 78036 -l *library* Search the library named **liblibrary.a**. A library shall be searched when its name is +
78037 encountered, so the placement of a -l option is significant. Several standard +
78038 libraries can be specified in this manner, as described in the EXTENDED +
78039 DESCRIPTION section. Implementations may recognize implementation-defined +
78040 suffixes other than .a as denoting libraries.
- 78041 -O *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit
78042 '0', all special code optimizations shall be disabled. If it is the digit '1', the
78043 nature of the optimization is unspecified. If the -O option is omitted, the nature of
78044 the system's default optimization is unspecified. It is unspecified whether code
78045 generated in the presence of the -O 0 option is the same as that generated when
78046 -O is omitted. Other *optlevel* values may be supported.
- 78047 -o *outfile* Use the pathname *outfile*, instead of the default **a.out**, for the executable file +
78048 produced. If the -o option is present with -c or -E, the result is unspecified. +
- 78049 -s Produce object or executable files, or both, from which symbolic and other +
78050 information not required for proper execution using the *exec* family defined in the +
78051 System Interfaces volume of POSIX.1-200x has been removed (stripped). If both -g +
78052 and -s options are present, the action taken is unspecified.
- 78053 -U *name* Remove any initial definition of *name*.
- 78054 Multiple instances of the -D, -I, -L, -l, and -U options can be specified.

OPERANDS

78055 The application shall ensure that at least one *pathname* operand is specified. The following forms |
78056 for *pathname* operands shall be supported:
78057

- 78058 *file.c* A C-language source file to be compiled and optionally linked. The application
78059 shall ensure that the operand is of this form if the -c option is used.

78060 *file.a* A library of object files typically produced by the *ar* utility, and passed directly to
 78061 the link editor. Implementations may recognize implementation-defined suffixes
 78062 other than *.a* as denoting object file libraries.

78063 *file.o* An object file produced by *c99 -c* and passed directly to the link editor.
 78064 Implementations may recognize implementation-defined suffixes other than *.o* as
 78065 denoting object files.

78066 The processing of other files is implementation-defined. -

78067 STDIN

78068 Not used.

78069 INPUT FILES

78070 The input file shall be one of the following: a text file containing a C-language source program,
 78071 an object file in the format produced by *c99 -c*, or a library of object files, in the format produced
 78072 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
 78073 that produce files in these formats. Additional input file formats are implementation-defined.

78074 ENVIRONMENT VARIABLES

78075 The following environment variables shall affect the execution of *c99*:

78076 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 78077 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 78078 variables used to determine the values of locale categories.)

78079 *LC_ALL* If set to a non-empty string value, override the values of all the other
 78080 internationalization variables.

78081 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 78082 characters (for example, single-byte as opposed to multi-byte characters in
 78083 arguments and input files).

78084 *LC_MESSAGES*
 78085 Determine the locale that should be used to affect the format and contents of
 78086 diagnostic messages written to standard error.

78087 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

78088 *TMPDIR* Provide a pathname that should override the default directory for temporary files,
 78089 XSI if any. On XSI-conforming systems, provide a pathname that shall override the
 78090 default directory for temporary files, if any.

78091 ASYNCHRONOUS EVENTS

78092 Default.

78093 STDOUT

78094 If more than one *pathname* operand ending in *.c* (or possibly other unspecified suffixes) is given,
 78095 for each such file:

78096 "%s:\n", <pathname> |

78097 may be written. These messages, if written, shall precede the processing of each input file; they
 78098 shall not be written to the standard output if they are written to the standard error, as described
 78099 in the *STDERR* section.

78100 If the *-E* option is specified, the standard output shall be a text file that represents the results of
 78101 the preprocessing stage of the language; it may contain extra information appropriate for
 78102 subsequent compilation passes.

78103 **STDERR**

78104 The standard error shall be used only for diagnostic messages. If more than one *pathname*
 78105 operand ending in *.c* (or possibly other unspecified suffixes) is given, for each such file:

78106 "%s:\n", <pathname>

78107 may be written to allow identification of the diagnostic and warning messages with the
 78108 appropriate input file. These messages, if written, shall precede the processing of each input file;
 78109 they shall not be written to the standard error if they are written to the standard output, as
 78110 described in the STDOUT section.

78111 This utility may produce warning messages about certain conditions that do not warrant
 78112 returning an error (non-zero) exit value.

78113 **OUTPUT FILES**

78114 Object files or executable files or both are produced in unspecified formats. If an existing file that
 78115 does not resolve to a regular file matches the name of an object file being written or matches the
 78116 name of an executable file being created by *c99*, it is unspecified whether *c99* shall attempt to
 78117 write the object file or create the executable file, or shall issue a diagnostic and exit with a non-
 78118 zero exit status.

78119 **EXTENDED DESCRIPTION**78120 **Standard Libraries**

78121 The *c99* utility shall recognize the following **-l** options for standard libraries:

78122 **-l c** This option shall make available all interfaces referenced in the System Interfaces
 78123 volume of POSIX.1-200x, with the possible exception of those interfaces listed as
 78124 residing in < aio.h >, < arpa/inet.h >, < complex.h >, < fenv.h >, < math.h >,
 78125 < mqueue.h >, < netdb.h >, < net/if.h >, < netinet/in.h >, < pthread.h >, < sched.h >,
 78126 < semaphore.h >, < spawn.h >, < sys/socket.h >, *pthread_kill()*, and *pthread_sigmask()*
 78127 in < signal.h >, < trace.h >, interfaces marked as optional in < sys/mman.h >,
 78128 interfaces marked as ADV (Advisory Information) in < fcntl.h >, and interfaces
 78129 beginning with the prefix *clock_* or *time_* in < time.h >. This option shall not be
 78130 required to be present to cause a search of this library.

78131 **-l l** This option shall make available all interfaces required by the C-language output
 78132 of *lex* that are not made available through the **-l c** option.

78133 **-l pthread** This option shall make available all interfaces referenced in < pthread.h > and
 78134 *pthread_kill()* and *pthread_sigmask()* referenced in < signal.h >. An implementation
 78135 may search this library in the absence of this option.

78136 **-l m** This option shall make available all interfaces referenced in < math.h >,
 78137 < complex.h >, and < fenv.h >. An implementation may search this library in the
 78138 absence of this option.

78139 **-l rt** This option shall make available all interfaces referenced in < aio.h >, < mqueue.h >,
 78140 < sched.h >, < semaphore.h >, and < spawn.h >, interfaces marked as optional in
 78141 < sys/mman.h >, interfaces marked as ADV (Advisory Information) in < fcntl.h >,
 78142 and interfaces beginning with the prefix *clock_* and *time_* in < time.h >. An
 78143 implementation may search this library in the absence of this option.

78144 **-l trace** This option shall make available all interfaces referenced in < trace.h >. An
 78145 implementation may search this library in the absence of this option.

78146 **-l xnet** This option shall make available all interfaces referenced in < arpa/inet.h >,
 78147 < netdb.h >, < net/if.h >, < netinet/in.h >, and < sys/socket.h >. An implementation
 78148 may search this library in the absence of this option.

78149 **-l y** This option shall make available all interfaces required by the C-language output |
78150 of *yacc* that are not made available through the **-l c** option. |

78151 In the absence of options that inhibit invocation of the link editor, such as **-c** or **-E**, the *c99* utility |
78152 shall cause the equivalent of a **-l c** option to be passed to the link editor as the last *pathname* |
78153 operand or **-l** option, causing it to be searched after all other object files and libraries are loaded. |

78154 OB It is unspecified whether the libraries **libc.a**, **libl.a**, **libm.a**, **libpthread.a**, **librt.a**, **libtrace.a**, -
78155 **libxnet.a**, or **liby.a** exist as regular files. The implementation may accept as **-l** option-arguments |
78156 names of objects that do not exist as regular files.

78157 External Symbols

78158 The C compiler and link editor shall support the significance of external symbols up to a length |
78159 of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation- |
78160 defined maximum symbol length is unspecified.

78161 The compiler and link editor shall support a minimum of 511 external symbols per source or |
78162 object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be |
78163 written to the standard output if the implementation-defined limit is exceeded; other actions are |
78164 unspecified.

78165 Programming Environments

78166 All implementations shall support one of the following programming environments as a default. |
78167 Implementations may support more than one of the following programming environments. |
78168 Applications can use *sysconf()* or *getconf* to determine which programming environments are |
78169 supported.

78170 **Table 4-4** Programming Environments: Type Sizes

Programming Environment <i>getconf</i> Name	Bits in int	Bits in long	Bits in pointer	Bits in off_t
_POSIX_V7_ILP32_OFF32	32	32	32	32
_POSIX_V7_ILP32_OFFBIG	32	32	32	≥64
_POSIX_V7_LP64_OFF64	32	64	64	64
_POSIX_V7_LP64_OFFBIG	≥32	≥64	≥64	≥64

78177 All implementations shall support one or more environments where the widths of the following |
78178 types are no greater than the width of type **long**:

78179 **blksize_t** **ptrdiff_t** **tcflag_t**
78180 **cc_t** **size_t** **wchar_t**
78181 **mode_t** **speed_t** **wint_t**
78182 **nfds_t** **ssize_t**
78183 **pid_t** **suseconds_t**

78184 The executable files created when these environments are selected shall be in a proper format for |
78185 execution by the *exec* family of functions. Each environment may be one of the ones in [Table 4-4](#), |
78186 or it may be another environment. The names for the environments that meet this requirement |
78187 shall be output by a *getconf* command using the `POSIX_V7_WIDTH_RESTRICTED_ENVS` |
78188 argument, as a <newline>-separated list of names suitable for use with the *getconf -v* option. If |
78189 more than one environment meets the requirement, the names of all such environments shall be |
78190 output on separate lines. Any of these names can then be used in a subsequent *getconf* command |
78191 to obtain the flags specific to that environment with the following suffixes added as appropriate:

78192 _CFLAGS To get the C compiler flags.

78193 _LDFLAGS To get the linker/loader flags.

78194 _LIBS To get the libraries.

78195 This requirement may be removed in a future version.

78196 When this utility processes a file containing a function called *main()*, it shall be defined with a
78197 return type equivalent to **int**. Using return from the initial call to *main()* shall be equivalent
78198 (other than with respect to language scope issues) to calling *exit()* with the returned value.
78199 Reaching the end of the initial call to *main()* shall be equivalent to calling *exit(0)*. The
78200 implementation shall not declare a prototype for this function.

78201 Implementations provide configuration strings for C compiler flags, linker/loader flags, and
78202 libraries for each supported environment. When an application needs to use a specific
78203 programming environment rather than the implementation default programming environment
78204 while compiling, the application shall first verify that the implementation supports the desired
78205 environment. If the desired programming environment is supported, the application shall then
78206 invoke *c99* with the appropriate C compiler flags as the first options for the compile, the
78207 appropriate linker/loader flags after any other options except **-I** but before any operands or **-I**
78208 options, and the appropriate libraries at the end of the operands and **-I** options.

78209 Conforming applications shall not attempt to link together object files compiled for different
78210 programming models. Applications shall also be aware that binary data placed in shared
78211 memory or in files might not be recognized by applications built for other programming models.

78212 **Table 4-5** Programming Environments: *c99* and *cc* Arguments

Programming Environment <i>getconf</i> Name	Use	<i>c99</i> and <i>cc</i> Arguments <i>getconf</i> Name
_POSIX_V7_ILP32_OFF32	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_ILP32_OFF32_CFLAGS POSIX_V7_ILP32_OFF32_LDFLAGS POSIX_V7_ILP32_OFF32_LIBS
_POSIX_V7_ILP32_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_ILP32_OFFBIG_CFLAGS POSIX_V7_ILP32_OFFBIG_LDFLAGS POSIX_V7_ILP32_OFFBIG_LIBS
_POSIX_V7_LP64_OFF64	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_LP64_OFF64_CFLAGS POSIX_V7_LP64_OFF64_LDFLAGS POSIX_V7_LP64_OFF64_LIBS
_POSIX_V7_LPBIG_OFFBIG	C Compiler Flags Linker/Loader Flags Libraries	POSIX_V7_LPBIG_OFFBIG_CFLAGS POSIX_V7_LPBIG_OFFBIG_LDFLAGS POSIX_V7_LPBIG_OFFBIG_LIBS

78227 EXIT STATUS

78228 The following exit values shall be returned:

78229 0 Successful compilation or link edit.

78230 >0 An error occurred.

78231 CONSEQUENCES OF ERRORS

78232 When *c99* encounters a compilation error that causes an object file not to be created, it shall write
78233 a diagnostic to standard error and continue to compile other source code operands, but it shall
78234 not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a
78235 diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A
78236 conforming application shall rely on the exit status of *c99*, rather than on the existence or mode
78237 of the executable file.

APPLICATION USAGE

Since the *c99* utility usually creates files in the current directory during the compilation process, it is typically necessary to run the *c99* utility in a directory in which a file can be created.

On systems providing POSIX Conformance (see XBD Chapter 2, on page 15), *c99* is required only with the C-Language Development option; XSI-conformant systems always provide *c99*.

Some historical implementations have created *.o* files when *-c* is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on *.o* files being created, but it also must be prepared for any related *.o* files that already exist being deleted at the completion of the link edit.

There is the possible implication that if a user supplies versions of the standard functions (before they would be encountered by an implicit *-l c* or explicit *-l m*), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean name space, and so on), so the existence of files named in the same manner as the standard libraries within the *-L* directories is explicitly stated to produce unspecified behavior.

All of the functions specified in the System Interfaces volume of POSIX.1-200x may be made visible by implementations when the Standard C Library is searched. Conforming applications must explicitly request searching the other standard libraries when functions made visible by those libraries are used.

EXAMPLES

1. The following usage example compiles **foo.c** and creates the executable file **foo**:

```
c99 -o foo foo.c
```

The following usage example compiles **foo.c** and creates the object file **foo.o**:

```
c99 -c foo.c
```

The following usage example compiles **foo.c** and creates the executable file **a.out**:

```
c99 foo.c
```

The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable file **a.out**. It may also create and leave **foo.o**:

```
c99 foo.c bar.o
```

2. The following example shows how an application using threads interfaces can test for support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer** types and an **off_t** type using at least 64 bits:

```
if [ $(getconf _POSIX_V7_ILP32_OFFBIG) != "-1" ]
then
    c99 $(getconf POSIX_V7_ILP32_OFFBIG_CFLAGS) -D_XOPEN_SOURCE=700 \
        $(getconf POSIX_V7_ILP32_OFFBIG_LDFLAGS) foo.c -o foo \
        $(getconf POSIX_V7_ILP32_OFFBIG_LIBS) -l pthread
else
    echo ILP32_OFFBIG programming environment not supported
    exit 1
fi
```

3. The following examples clarify the use and interactions of *-L* and *-l* options.

Consider the case in which module **a.c** calls function *f()* in library **libQ.a**, and module **b.c** calls function *g()* in library **libp.a**. Assume that both libraries reside in */a/b/c*. The command line to compile and link in the desired way is:

78283 `c99 -L /a/b/c main.o a.c -l Q b.c -l p`

78284 In this case the `-L` option need only precede the first `-l` option, since both `libQ.a` and
78285 `libp.a` reside in the same directory.

78286 Multiple `-L` options can be used when library name collisions occur. Building on the
78287 previous example, suppose that the user wants to use a new `libp.a`, in `/a/a/a`, but still
78288 wants `f()` from `/a/b/c/libQ.a`:

78289 `c99 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p`

78290 In this example, the linker searches the `-L` options in the order specified, and finds
78291 `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving references for `b.c`. The order of the `-l`
78292 options is still important, however.

78293 4. The following example shows how an application can use a programming environment
78294 where the widths of the following types:

78295 **blksize_t, cc_t, mode_t, nfd_t, pid_t, ptrdiff_t, size_t, speed_t, ssize_t, suseconds_t,**
78296 **tcflag_t, wchar_t, wint_t**

78297 are no greater than the width of type **long**:

```
78298 # First choose one of the listed environments ...
78299 # ... if there are no additional constraints, the first one will do:
78300 CENV=$(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS | head -n 1)
78301 # ... or, if an environment that supports large files is preferred,
78302 # look for names that contain "OFF64" or "OFFBIG". (This chooses
78303 # the last one in the list if none match.)
78304 for CENV in $(getconf POSIX_V7_WIDTH_RESTRICTED_ENVS)
78305 do
78306     case $CENV in
78307         *OFF64*|*OFFBIG*) break ;;
78308     esac
78309 done
78310 # The chosen environment name can now be used like this:
78311 c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200xxxL \
78312 $(getconf ${CENV}_LDFLAGS) foo.c -o foo \
78313 $(getconf ${CENV}_LIBS)
```

78314 RATIONALE

78315 The `c99` utility is based on the `c89` utility originally introduced in the ISO POSIX-2:1993
78316 standard.

78317 Some of the changes from `c89` include the ability to intersperse options and operands (which
78318 many `c89` implementations allowed despite it not being specified), the description of `-l` as an
78319 option instead of an operand, and the modification to the contents of the Standard Libraries
78320 section to account for new headers and options; for example, `<spawn.h>` added to the
78321 description of `-l rt`, and `-l trace` added for the Tracing option.

78322 POSIX.1-200x specifies that the `c99` utility must be able to use regular files for `*.o` files and for
78323 `a.out` files. Implementations are free to overwrite existing files of other types when attempting to
78324 create object files and executable files, but are not required to do so. If something other than a
78325 regular file is specified and using it fails for any reason, `c99` is required to issue a diagnostic
78326 message and exit with a non-zero exit status. But for some file types, the problem may not be
78327 noticed for a long time. For example, if a FIFO named `a.out` exists in the current directory, `c99`
78328 may attempt to open `a.out` and will hang in the `open()` call until another process opens the FIFO

78329 for reading. Then *c99* may write most of the **a.out** to the FIFO and fail when it tries to seek back
 78330 close to the start of the file to insert a timestamp (FIFOs are not seekable files). The *c99* utility is
 78331 also allowed to issue a diagnostic immediately if it encounters an **a.out** or ***.o** file that is not a
 78332 regular file. For portable use, applications should ensure that any **a.out**, **-o** option-argument, or
 78333 ***.o** files corresponding to any ***.c** files do not conflict with names already in use that are not
 78334 regular files or symbolic links that point to regular files.

78335 FUTURE DIRECTIONS

78336 None.

78337 SEE ALSO

78338 [Section 1.1.1.4](#) (on page 2228), *ar*, *getconf*, *make*, *nm*, *strip*, *umask*

78339 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [Chapter 13](#) (on page 205) |

78340 XSH *exec*, *sysconf()*

78341 CHANGE HISTORY

78342 First released in Issue 6. Included for alignment with the ISO/IEC 9899:1999 standard.

78343 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/12 is applied, correcting the EXTENDED
 78344 DESCRIPTION of **-l c** and **-l m**. Previously, the text did not take into account the presence of
 78345 the *c99* math headers.

78346 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/13 is applied, changing the reference to
 78347 the **libxnet** library to **libxnet.a**.

78348 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/5 is applied, updating the OPTIONS
 78349 section, so that the names of files contained in the directory specified by the **-L** option are not
 78350 assumed to end in the **.a** suffix. The set of library prefixes is also updated.

78351 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/6 is applied, removing the lead
 78352 underscore from the POSIX_V6_WIDTH_RESTRICTED_ENVS variable in the EXTENDED
 78353 DESCRIPTION and the EXAMPLES sections.

78354 Issue 7

78355 Austin Group Interpretation 1003.1-2001 #020 (SD5-XCU-ERN-10) is applied, adding to the
 78356 OUTPUT FILES section and also adding associated RATIONALE.

78357 Austin Group Interpretation 1003.1-2001 #095 is applied, clarifying the **-l library** operand.

78358 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
 78359 apply (options can be interspersed with operands).

78360 SD5-XCU-ERN-11 is applied, adding the **<net/if.h>** header to the descriptions of **-l c** and
 78361 **-l xnet**.

78362 SD5-XCU-ERN-65 is applied, updating the EXAMPLES section.

78363 SD5-XCU-ERN-67 and SD5-XCU-ERN-97 are applied, updating the SYNOPSIS. |

78364 SD5-XCU-ERN-133 is applied, updating the EXTENDED DESCRIPTION. |

78365 The *getconf* variables for the supported programming environments are updated to be V7.

78366 The **-l trace** operand is marked obsolescent. +

78367 The *c99* reference page is rewritten to describe **-l** as an option rather than an operand.

78368 **NAME**

78369 cal — print a calendar

78370 **SYNOPSIS**78371 XSI cal `[[month] year]`78372 **DESCRIPTION**

78373 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from
 78374 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,
 78375 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on
 78376 September 14, 1752.

78377 **OPTIONS**

78378 None.

78379 **OPERANDS**

78380 The following operands shall be supported:

78381 *month* Specify the month to be displayed, represented as a decimal integer from 1
 78382 (January) to 12 (December). The default shall be the current month.

78383 *year* Specify the year for which the calendar is displayed, represented as a decimal
 78384 integer from 1 to 9999. The default shall be the current year.

78385 **STDIN**

78386 Not used.

78387 **INPUT FILES**

78388 None.

78389 **ENVIRONMENT VARIABLES**78390 The following environment variables shall affect the execution of *cal*:

78391 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 78392 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 78393 variables used to determine the values of locale categories.)

78394 *LC_ALL* If set to a non-empty string value, override the values of all the other
 78395 internationalization variables.

78396 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 78397 characters (for example, single-byte as opposed to multi-byte characters in
 78398 arguments).

78399 *LC_MESSAGES*

78400 Determine the locale that should be used to affect the format and contents of
 78401 diagnostic messages written to standard error, and informative messages written
 78402 to standard output.

78403 *LC_TIME* Determine the format and contents of the calendar.

78404 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

78405 *TZ* Determine the timezone used to calculate the value of the current month.

78406 **ASYNCHRONOUS EVENTS**

78407 Default.

78408 **STDOUT**

78409 The standard output shall be used to display the calendar, in an unspecified format.

78410 **STDERR**

78411 The standard error shall be used only for diagnostic messages.

78412 **OUTPUT FILES**

78413 None.

78414 **EXTENDED DESCRIPTION**

78415 None.

78416 **EXIT STATUS**

78417 The following exit values shall be returned:

78418 0 Successful completion.

78419 >0 An error occurred.

78420 **CONSEQUENCES OF ERRORS**

78421 Default.

78422 **APPLICATION USAGE**

78423 Note that:

78424 `cal 83`

78425 refers to A.D. 83, not 1983.

78426 **EXAMPLES**

78427 None.

78428 **RATIONALE**

78429 None.

78430 **FUTURE DIRECTIONS**78431 A future version of this standard may support locale-specific recognition of the date of adoption
78432 of the Gregorian calendar.78433 **SEE ALSO**78434 XBD [Chapter 8](#) (on page 159)78435 **CHANGE HISTORY**

78436 First released in Issue 2.

78437 **Issue 6**78438 The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of
78439 the Gregorian calendar.78440 **Issue 7**

78441 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

78442 **NAME**

78443 cat — concatenate and print files

78444 **SYNOPSIS**78445 cat [-u] [*file...*]78446 **DESCRIPTION**78447 The *cat* utility shall read files in sequence and shall write their contents to the standard output in
78448 the same sequence.78449 **OPTIONS**78450 The *cat* utility shall conform to XBD [Section 12.2](#) (on page 201).

78451 The following option shall be supported:

78452 **-u** Write bytes from the input file to the standard output without delay as each is
78453 read.78454 **OPERANDS**

78455 The following operand shall be supported:

78456 *file* A pathname of an input file. If no *file* operands are specified, the standard input
78457 shall be used. If a *file* is '-', the *cat* utility shall read from the standard input at
78458 that point in the sequence. The *cat* utility shall not close and reopen standard input
78459 when it is referenced in this way, but shall accept multiple occurrences of '-' as a
78460 *file* operand.78461 **STDIN**78462 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
78463 See the INPUT FILES section.78464 **INPUT FILES**

78465 The input files can be any file type.

78466 **ENVIRONMENT VARIABLES**78467 The following environment variables shall affect the execution of *cat*:78468 **LANG** Provide a default value for the internationalization variables that are unset or null.
78469 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
78470 variables used to determine the values of locale categories.)78471 **LC_ALL** If set to a non-empty string value, override the values of all the other
78472 internationalization variables.78473 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
78474 characters (for example, single-byte as opposed to multi-byte characters in
78475 arguments).78476 **LC_MESSAGES**78477 Determine the locale that should be used to affect the format and contents of
78478 diagnostic messages written to standard error.78479 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.78480 **ASYNCHRONOUS EVENTS**

78481 Default.

78482 **STDOUT**

78483 The standard output shall contain the sequence of bytes read from the input files. Nothing else
78484 shall be written to the standard output.

78485 **STDERR**

78486 The standard error shall be used only for diagnostic messages.

78487 **OUTPUT FILES**

78488 None.

78489 **EXTENDED DESCRIPTION**

78490 None.

78491 **EXIT STATUS**

78492 The following exit values shall be returned:

78493 0 All input files were output successfully.

78494 >0 An error occurred.

78495 **CONSEQUENCES OF ERRORS**

78496 Default.

78497 **APPLICATION USAGE**

78498 The `-u` option has value in prototyping non-blocking reads from FIFOs. The intent is to support
78499 the following sequence:

```
78500 mkfifo foo
78501 cat -u foo > /dev/ttyl3 &
78502 cat -u > foo
```

78503 It is unspecified whether standard output is or is not buffered in the default case. This is
78504 sometimes of interest when standard output is associated with a terminal, since buffering may
78505 delay the output. The presence of the `-u` option guarantees that unbuffered I/O is available. It is
78506 implementation-defined whether the `cat` utility buffers output if the `-u` option is not specified.
78507 Traditionally, the `-u` option is implemented using the equivalent of the `setvbuf()` function
78508 defined in the System Interfaces volume of POSIX.1-200x.

78509 **EXAMPLES**

78510 The following command:

```
78511 cat myfile
```

78512 writes the contents of the file **myfile** to standard output.

78513 The following command:

```
78514 cat doc1 doc2 > doc.all
```

78515 concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

78516 Because of the shell language mechanism used to perform output redirection, a command such
78517 as this:

```
78518 cat doc doc.end > doc
```

78519 causes the original data in **doc** to be lost.

78520 The command:

```
78521 cat start - middle - end > file
```

78522 when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a
78523 single invocation of `cat`. Note, however, that if standard input is a regular file, this would be
78524 equivalent to the command:

78525 `cat start - middle /dev/null end > file`

78526 because the entire contents of the file would be consumed by *cat* the first time '-' was used as a
78527 *file* operand and an end-of-file condition would be detected immediately when '-' was
78528 referenced the second time.

78529 RATIONALE

78530 Historical versions of the *cat* utility include the options `-e`, `-t`, and `-v`, which permit the ends of
78531 lines, `<tab>`s, and invisible characters, respectively, to be rendered visible in the output. The
78532 standard developers omitted these options because they provide too fine a degree of control
78533 over what is made visible, and similar output can be obtained using a command such as:

78534 `sed -n -e 's/$/$/' -e l pathname`

78535 The `-s` option was omitted because it corresponds to different functions in BSD and System
78536 V-based systems. The BSD `-s` option to squeeze blank lines can be accomplished by the shell
78537 script shown in the following example:

```
78538 sed -n '  
78539 # Write non-empty lines.  
78540 ./ {  
78541     p  
78542     d  
78543 }  
78544 # Write a single empty line, then look for more empty lines.  
78545 /^$/ p  
78546 # Get next line, discard the held <newline> (empty line),  
78547 # and look for more empty lines.  
78548 :Empty  
78549 /^$/ {  
78550     N  
78551     s/././  
78552     b Empty  
78553 }  
78554 # Write the non-empty line before going back to search  
78555 # for the first in a set of empty lines.  
78556 p  
78557 '
```

78558 The System V `-s` option to silence error messages can be accomplished by redirecting the
78559 standard error. Note that the BSD documentation for *cat* uses the term "blank line" to mean the
78560 same as the POSIX "empty line": a line consisting only of a `<newline>`.

78561 The BSD `-n` option was omitted because similar functionality can be obtained from the `-n`
78562 option of the *pr* utility.

78563 FUTURE DIRECTIONS

78564 None.

78565 SEE ALSO

78566 [more](#)

78567 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

78568 [XSH `setvbuf\(\)`](#)

78569

CHANGE HISTORY

78570

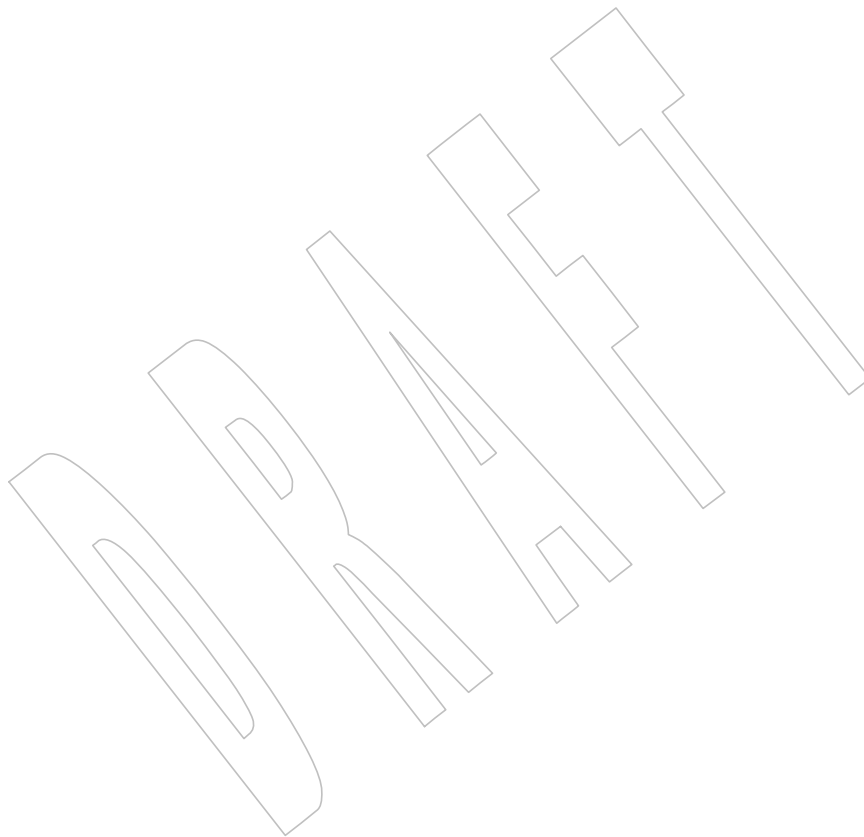
First released in Issue 2.

78571

Issue 7

78572

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



78573 **NAME**
78574 cd — change the working directory

78575 **SYNOPSIS**
78576 cd [-L|-P] [*directory*]

78577 cd -

78578 DESCRIPTION

78579 The *cd* utility shall change the working directory of the current shell execution environment (see
78580 [Section 2.12](#), on page 2277) by executing the following steps in sequence. (In the following steps,
78581 the symbol **curpath** represents an intermediate value used to simplify the description of the
78582 algorithm used by *cd*. There is no requirement that **curpath** be made visible to the application.)

- 78583 1. If no *directory* operand is given and the *HOME* environment variable is empty or
78584 undefined, the default behavior is implementation-defined and no further steps shall be
78585 taken.
- 78586 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty
78587 value, the *cd* utility shall behave as if the directory named in the *HOME* environment
78588 variable was specified as the *directory* operand.
- 78589 3. If the *directory* operand begins with a slash character, set **curpath** to the operand and
78590 proceed to step 7.
- 78591 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.
- 78592 5. Starting with the first pathname in the colon-separated pathnames of *CDPATH* (see the
78593 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the
78594 concatenation of that pathname, a slash character, and the *directory* operand names a
78595 directory. If the pathname is null, test if the concatenation of dot, a slash character, and
78596 the operand names a directory. In either case, if the resulting string names an existing
78597 directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with
78598 the next pathname in *CDPATH* until all pathnames have been tested.
- 78599 6. If the **-P** option is in effect, set **curpath** to the *directory* operand. Otherwise, set **curpath** to
78600 the string formed by the concatenation of the value of *PWD*, a slash character, and the
78601 operand.
- 78602 7. If the **-P** option is in effect, proceed to step 10. If **curpath** does not begin with a slash
78603 character, set **curpath** to the string formed by the concatenation of the value of *PWD*, a
78604 slash character, and **curpath**.
- 78605 8. The **curpath** value shall then be converted to canonical form as follows, considering each
78606 component from beginning to end, in sequence:
 - 78607 a. Dot components and any slashes that separate them from the next component shall
78608 be deleted.
 - 78609 b. For each dot-dot component, if there is a preceding component and it is neither
78610 root nor dot-dot, then:
 - 78611 i. If the preceding component does not refer (in the context of pathname
78612 resolution with symbolic links followed) to a directory, then the *cd* utility
78613 shall display an appropriate error message and no further steps shall be
78614 taken.

- 78615 ii. The preceding component, all slashes separating the preceding component
78616 from dot-dot, dot-dot, and all slashes separating dot-dot from the following
78617 component (if any) shall be deleted.
- 78618 c. An implementation may further simplify **curpath** by removing any trailing slash
78619 characters that are not also leading slashes, replacing multiple non-leading
78620 consecutive slashes with a single slash, and replacing three or more leading slashes
78621 with a single slash. If, as a result of this canonicalization, the **curpath** variable is
78622 null, no further steps shall be taken.
- 78623 9. If **curpath** is longer than {PATH_MAX} bytes (including the terminating null) and the
78624 *directory* operand was not longer than {PATH_MAX} bytes (including the terminating
78625 null), then **curpath** shall be converted from an absolute pathname to an equivalent
78626 relative pathname if possible. This conversion shall always be considered possible if the
78627 value of *PWD*, with a trailing slash added if it does not already have one, is an initial
78628 substring of **curpath**. Whether or not it is considered possible under other circumstances
78629 is unspecified. Implementations may also apply this conversion if **curpath** is not longer
78630 than {PATH_MAX} bytes or the *directory* operand was longer than {PATH_MAX} bytes.
- 78631 10. The *cd* utility shall then perform actions equivalent to the *chdir()* function called with
78632 **curpath** as the *path* argument. If these actions fail for any reason, the *cd* utility shall
78633 display an appropriate error message and the remainder of this step shall not be
78634 executed. If the **-P** option is not in effect, the *PWD* environment variable shall be set to
78635 the value that **curpath** had on entry to step 9 (i.e., before conversion to a relative
78636 pathname). If the **-P** option is in effect, the *PWD* environment variable shall be set to an
78637 absolute pathname for the current working directory and shall not contain filename
78638 components that, in the context of pathname resolution, refer to a file of type symbolic
78639 link. If there is insufficient permission on the new directory, or on any parent of that
78640 directory, to determine the current working directory, the value of the *PWD* environment
78641 variable is unspecified.

78642 If, during the execution of the above steps, the *PWD* environment variable is changed, the
78643 *OLDPWD* environment variable shall also be changed to the value of the old working directory
78644 (that is the current working directory immediately prior to the call to *cd*).

78645 OPTIONS

78646 The *cd* utility shall conform to XBD [Section 12.2](#) (on page 201).

78647 The following options shall be supported by the implementation:

- 78648 **-L** Handle the operand dot-dot logically; symbolic link components shall not be
78649 resolved before dot-dot components are processed (see steps 8. and 9. in the
78650 DESCRIPTION).
- 78651 **-P** Handle the operand dot-dot physically; symbolic link components shall be
78652 resolved before dot-dot components are processed (see step 7. in the
78653 DESCRIPTION).

78654 If both **-L** and **-P** options are specified, the last of these options shall be used and all others
78655 ignored. If neither **-L** nor **-P** is specified, the operand shall be handled dot-dot logically; see the
78656 DESCRIPTION.

78657 OPERANDS

78658 The following operands shall be supported:

- 78659 *directory* An absolute or relative pathname of the directory that shall become the new
78660 working directory. The interpretation of a relative pathname by *cd* depends on the
78661 **-L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an
78662 empty string, the results are unspecified.

78663 – When a hyphen is used as the operand, this shall be equivalent to the command:

78664 cd "\$OLDPWD" && pwd

78665 which changes to the previous working directory and then writes its name.

78666 **STDIN**

78667 Not used.

78668 **INPUT FILES**

78669 None.

78670 **ENVIRONMENT VARIABLES**

78671 The following environment variables shall affect the execution of *cd*:

78672 *CDPATH* A colon-separated list of pathnames that refer to directories. The *cd* utility shall use
78673 this list in its attempt to change the directory, as described in the DESCRIPTION.
78674 An empty string in place of a directory pathname represents the current directory.
78675 If *CDPATH* is not set, it shall be treated as if it were an empty string.

78676 *HOME* The name of the directory, used when no *directory* operand is specified.

78677 *LANG* Provide a default value for the internationalization variables that are unset or null. |
78678 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
78679 variables used to determine the values of locale categories.)

78680 *LC_ALL* If set to a non-empty string value, override the values of all the other
78681 internationalization variables.

78682 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
78683 characters (for example, single-byte as opposed to multi-byte characters in
78684 arguments).

78685 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
78686 diagnostic messages written to standard error.
78687

78688 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

78689 *OLDPWD* A pathname of the previous working directory, used by *cd -*.

78690 *PWD* This variable shall be set as specified in the DESCRIPTION. If an application sets
78691 or unsets the value of *PWD*, the behavior of *cd* is unspecified.

78692 **ASYNCHRONOUS EVENTS**

78693 Default.

78694 **STDOUT**

78695 If a non-empty directory name from *CDPATH* is used, or if *cd -* is used, an absolute pathname of
78696 the new working directory shall be written to the standard output as follows:

78697 "%s\n", <new directory>

78698 Otherwise, there shall be no output.

78699 **STDERR**

78700 The standard error shall be used only for diagnostic messages.

78701 **OUTPUT FILES**

78702 None.

78703 **EXTENDED DESCRIPTION**

78704 None.

78705 **EXIT STATUS**

78706 The following exit values shall be returned:

78707 0 The directory was successfully changed.

78708 >0 An error occurred.

78709 **CONSEQUENCES OF ERRORS**

78710 The working directory shall remain unchanged.

78711 **APPLICATION USAGE**78712 Since *cd* affects the current shell execution environment, it is always provided as a shell regular
78713 built-in. If it is called in a subshell or separate utility execution environment, such as one of the
78714 following:78715 (cd /tmp)
78716 nohup cd
78717 find . -exec cd {} \;

78718 it does not affect the working directory of the caller's environment.

78719 The user must have execute (search) permission in *directory* in order to change to it.78720 **EXAMPLES**

78721 None.

78722 **RATIONALE**78723 The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of
78724 the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.78725 A common extension when *HOME* is undefined is to get the login directory from the user
78726 database for the invoking user. This does not occur on System V implementations.78727 Some historical shells, such as the KornShell, took special actions when the directory name
78728 contained a dot-dot component, selecting the logical parent of the directory, rather than the
78729 actual parent directory; that is, it moved up one level toward the '/' in the pathname,
78730 remembering what the user typed, rather than performing the equivalent of:78731 `chdir("../");`78732 In such a shell, the following commands would not necessarily produce equivalent output for all
78733 directories:78734 `cd .. && ls ls ..`78735 This behavior is now the default. It is not consistent with the definition of dot-dot in most
78736 historical practice; that is, while this behavior has been optionally available in the KornShell,
78737 other shells have historically not supported this functionality. The logical pathname is stored in
78738 the *PWD* environment variable when the *cd* utility completes and this value is used to construct
78739 the next directory name if *cd* is invoked with the *-L* option.78740 **FUTURE DIRECTIONS**

78741 None.

78742 **SEE ALSO**78743 [Section 2.12](#) (on page 2277), *pwd*78744 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)78745 [XSH *chdir*\(\)](#)

78746

CHANGE HISTORY

78747

First released in Issue 2.

78748

Issue 6

78749

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

78750

- The *cd* **-** operand, *PWD*, and *OLDPWD* are added.

78751

78752

The **-L** and **-P** options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

78753

78754

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/14 is applied, changing the SYNOPSIS to make it clear that the **-L** and **-P** options are mutually-exclusive.

78755

78756

Issue 7

78757

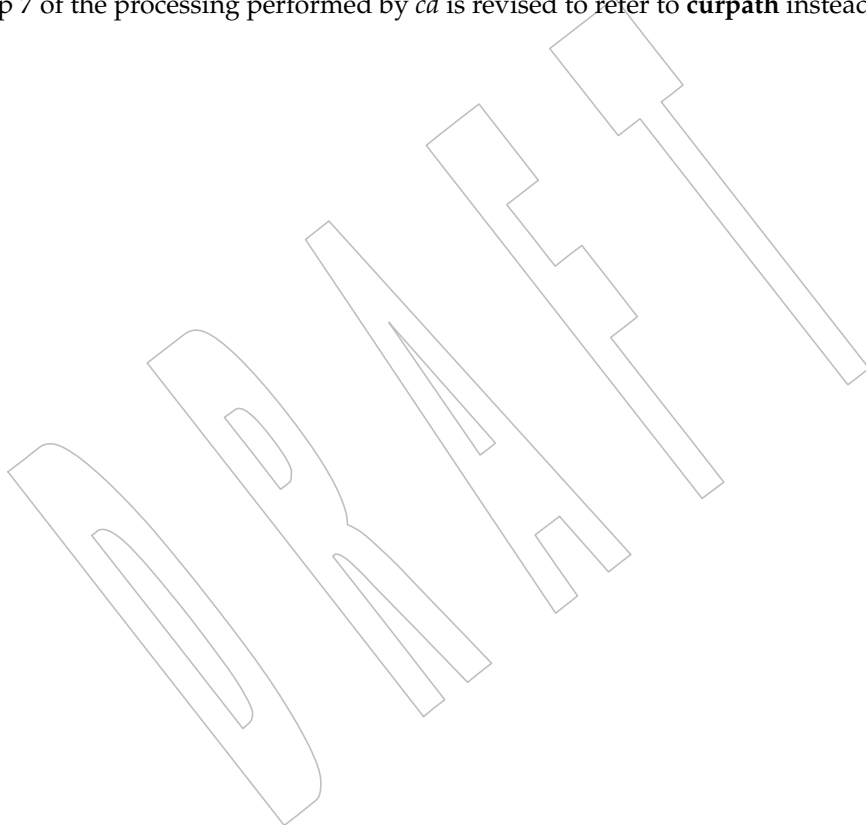
Austin Group Interpretation 1003.1-2001 #037 is applied.

78758

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

78759

Step 7 of the processing performed by *cd* is revised to refer to **curpath** instead of “the operand”. +



78760 **NAME**
 78761 cflow — generate a C-language flowgraph (DEVELOPMENT)

78762 **SYNOPSIS**
 78763 XSI cflow [-r] [-d num] [-D name[=def]]... [-i incl] [-I dir]...
 78764 [-U dir]... file...

78765 **DESCRIPTION**
 78766 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex*, or *yacc*
 78767 source files, and attempt to build a graph, written to standard output, charting the external
 78768 references.

78769 **OPTIONS**
 78770 The *cflow* utility shall conform to XBD Section 12.2 (on page 201), except that the order of the **-D**,
 78771 **-I**, and **-U** options (which are identical to their interpretation by *c99*) is significant.

78772 The following options shall be supported:

78773 **-d num** Indicate the depth at which the flowgraph is cut off. The application shall ensure
 78774 that the argument *num* is a decimal integer. By default this is a very large number
 78775 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive
 78776 integer shall be ignored.

78777 **-i incl** Increase the number of included symbols. The *incl* option-argument is one of the
 78778 following characters:

78779 *x* Include external and static data symbols. The default shall be to include only
 78780 functions in the flowgraph.

78781 *_* (Underscore) Include names that begin with an underscore. The default shall
 78782 be to exclude these functions (and data if **-i x** is used).

78783 **-r** Reverse the caller: callee relationship, producing an inverted listing showing the
 78784 callers of each function. The listing shall also be sorted in lexicographical order by
 78785 callee.

78786 **OPERANDS**
 78787 The following operand is supported:

78788 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by
 78789 **.I** shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the
 78790 output of *c99* -E. Such files shall be processed as appropriate, determined by their
 78791 suffix.

78792 Files suffixed by **.s** (conventionally assembler source) may have more limited
 78793 information extracted from them.

78794 **STDIN**
 78795 Not used.

78796 **INPUT FILES**
 78797 The input files shall be object files or assembler, C-language, *lex*, or *yacc* source files.

78798 **ENVIRONMENT VARIABLES**
 78799 The following environment variables shall affect the execution of *cflow*:

78800 **LANG** Provide a default value for the internationalization variables that are unset or null.
 78801 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 78802 variables used to determine the values of locale categories.)

- 78803 *LC_ALL* If set to a non-empty string value, override the values of all the other
78804 internationalization variables.
- 78805 *LC_COLLATE*
78806 Determine the locale for the ordering of the output when the `-r` option is used.
- 78807 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
78808 characters (for example, single-byte as opposed to multi-byte characters in
78809 arguments and input files).
- 78810 *LC_MESSAGES*
78811 Determine the locale that should be used to affect the format and contents of
78812 diagnostic messages written to standard error.
- 78813 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

78814 Default.
78815

STDOUT

78816 The flowgraph written to standard output shall be formatted as follows:
78817

78818 "`%d %s:%s\n`", *<reference number>*, *<global>*, *<definition>*

78819 Each line of output begins with a reference (that is, line) number, followed by indentation of at
78820 least one column position per level. This is followed by the name of the global, a colon, and its
78821 definition. Normally globals are only functions not defined as an external or beginning with an
78822 underscore; see the **OPTIONS** section for the `-i` inclusion option. For information extracted from
78823 C-language source, the definition consists of an abstract type declaration (for example, `char *`)
78824 and, delimited by angle brackets, the name of the source file and the line number where the
78825 definition was found. Definitions extracted from object files indicate the filename and location
78826 counter under which the symbol appeared (for example, *text*).

78827 Once a definition of a name has been written, subsequent references to that name contain only
78828 the reference number of the line where the definition can be found. For undefined references,
78829 only "`<>`" shall be written.

STDERR

78830 The standard error shall be used only for diagnostic messages.
78831

OUTPUT FILES

78832 None.
78833

EXTENDED DESCRIPTION

78834 None.
78835

EXIT STATUS

78836 The following exit values shall be returned:
78837

78838 0 Successful completion.

78839 >0 An error occurred.

CONSEQUENCES OF ERRORS

78840 Default.
78841

78842

APPLICATION USAGE

78843

Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

78844

78845

EXAMPLES

78846

Given the following in **file.c**:

78847

```
int i;
```

78848

```
int f();
```

78849

```
int g();
```

78850

```
int h();
```

78851

```
int
```

78852

```
main()
```

78853

```
{
```

78854

```
    f();
```

78855

```
    g();
```

78856

```
    f();
```

78857

```
}
```

78858

```
int
```

78859

```
f()
```

78860

```
{
```

78861

```
    i = h();
```

78862

```
}
```

78863

The command:

78864

```
cflow -i x file.c
```

78865

produces the output:

78866

```
1 main: int(), <file.c 6>
```

78867

```
2   f: int(), <file.c 13>
```

78868

```
3     h: <>
```

78869

```
4     i: int, <file.c 1>
```

78870

```
5   g: <>
```

78871

RATIONALE

78872

None.

78873

FUTURE DIRECTIONS

78874

None.

78875

SEE ALSO

78876

c99, *lex*, *yacc*

78877

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

78878

CHANGE HISTORY

78879

First released in Issue 2.

78880

Issue 6

78881

The normative text is reworded to avoid use of the term “must” for application requirements.

78882

Issue 7

78883

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

78884 **NAME**
 78885 chgrp — change the file group ownership

78886 **SYNOPSIS**
 78887 chgrp [-h] group file...
 78888 chgrp -R [-H|-L|-P] group file...

78889 **DESCRIPTION**
 78890 The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID
 78891 specified by the *group* operand.

78892 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the
 78893 directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to
 78894 the *chown()* function defined in the System Interfaces volume of POSIX.1-200x, called with the
 78895 following arguments:

- 78896 • The *file* operand shall be used as the *path* argument.
- 78897 • The user ID of the file shall be used as the *owner* argument.
- 78898 • The specified group ID shall be used as the *group* argument.

78899 Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
 78900 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
 78901 group-ID bits of other file types may be cleared.

78902 **OPTIONS**
 78903 The *chgrp* utility shall conform to XBD [Section 12.2](#) (on page 201).

78904 The following options shall be supported by the implementation:

- 78905 **-h** If the system supports group IDs for symbolic links, for each *file* operand that
 78906 names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the
 78907 symbolic link instead of the file referenced by the symbolic link. If the system does
 78908 not support group IDs for symbolic links, for each *file* operand that names a file of
 78909 type symbolic link, *chgrp* shall do nothing more with the current file and shall go
 78910 on to any remaining files.
- 78911 **-H** If the **-R** option is specified and a symbolic link referencing a file of type directory
 78912 is specified on the command line, *chgrp* shall change the group of the directory
 78913 referenced by the symbolic link and all files in the file hierarchy below it.
- 78914 **-L** If the **-R** option is specified and a symbolic link referencing a file of type directory
 78915 is specified on the command line or encountered during the traversal of a file
 78916 hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic
 78917 link and all files in the file hierarchy below it.
- 78918 **-P** If the **-R** option is specified and a symbolic link is specified on the command line
 78919 or encountered during the traversal of a file hierarchy, *chgrp* shall change the group
 78920 ID of the symbolic link if the system supports this operation. The *chgrp* utility shall
 78921 not follow the symbolic link to any other part of the file hierarchy.
- 78922 **-R** Recursively change file group IDs. For each *file* operand that names a directory,
 78923 *chgrp* shall change the group of the directory and all files in the file hierarchy
 78924 below it. Unless a **-H**, **-L**, or **-P** option is specified, it is unspecified which of these
 78925 options will be used as the default.

78926 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
 78927 considered an error. The last option specified shall determine the behavior of the utility.

78928 **OPERANDS**

78929 The following operands shall be supported:

78930 *group* A group name from the group database or a numeric group ID. Either specifies a
 78931 group ID to be given to each file named by one of the *file* operands. If a numeric
 78932 *group* operand exists in the group database as a group name, the group ID number
 78933 associated with that group name is used as the group ID.

78934 *file* A pathname of a file whose group ID is to be modified.

78935 **STDIN**

78936 Not used.

78937 **INPUT FILES**

78938 None.

78939 **ENVIRONMENT VARIABLES**78940 The following environment variables shall affect the execution of *chgrp*:

78941 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 78942 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 78943 variables used to determine the values of locale categories.)

78944 *LC_ALL* If set to a non-empty string value, override the values of all the other
 78945 internationalization variables.

78946 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 78947 characters (for example, single-byte as opposed to multi-byte characters in
 78948 arguments).

78949 *LC_MESSAGES*

78950 Determine the locale that should be used to affect the format and contents of
 78951 diagnostic messages written to standard error.

78952 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

78953 **ASYNCHRONOUS EVENTS**

78954 Default.

78955 **STDOUT**

78956 Not used.

78957 **STDERR**

78958 The standard error shall be used only for diagnostic messages.

78959 **OUTPUT FILES**

78960 None.

78961 **EXTENDED DESCRIPTION**

78962 None.

78963 **EXIT STATUS**

78964 The following exit values shall be returned:

78965 0 The utility executed successfully and all requested changes were made.

78966 >0 An error occurred.

78967 **CONSEQUENCES OF ERRORS**

78968 Default.

78969
78970
78971

78972
78973
78974

78975
78976

78977
78978
78979
78980
78981

78982
78983
78984

78985
78986

78987
78988

78989
78990

78991
78992

78993
78994
78995

78996
78997

78998
78999

79000
79001
79002**APPLICATION USAGE**

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the *group* specified is not the effective group ID or one of the supplementary group IDs of the calling process.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. The standard developers chose to mask these by specifying only 0 and >0 as exit values.

The functionality of *chgrp* is described substantially through references to *chown()*. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *chown*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

XSH *chown*

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to "Default."

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/15 is applied, changing the SYNOPSIS to make it clear that **-h** and **-R** are optional.

Issue 7

SD5-XCU-ERN-8 is applied, removing the **-R** from the first line of the SYNOPSIS.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79003	NAME	
79004		chmod — change the file modes
79005	SYNOPSIS	
79006		chmod [-R] mode file...
79007	DESCRIPTION	
79008		The <i>chmod</i> utility shall change any or all of the file mode bits of the file named by each <i>file</i> operand in the way specified by the <i>mode</i> operand.
79009		
79010		It is implementation-defined whether and how the <i>chmod</i> utility affects any alternate or additional file access control mechanism (see XBD Section 4.4, on page 96) being used for the specified file.
79011		
79012		
79013		Only a process whose effective user ID matches the user ID of the file, or a process with the appropriate privileges, shall be permitted to change the file mode bits of a file.
79014		
79015	OPTIONS	
79016		The <i>chmod</i> utility shall conform to XBD Section 12.2 (on page 201).
79017		The following option shall be supported:
79018	-R	Recursively change file mode bits. For each <i>file</i> operand that names a directory, <i>chmod</i> shall change the file mode bits of the directory and all files in the file hierarchy below it.
79019		
79020		
79021	OPERANDS	
79022		The following operands shall be supported:
79023	<i>mode</i>	Represents the change to be made to the file mode bits of each file named by one of the <i>file</i> operands; see the EXTENDED DESCRIPTION section.
79024		
79025	<i>file</i>	A pathname of a file whose file mode bits shall be modified.
79026	STDIN	
79027		Not used.
79028	INPUT FILES	
79029		None.
79030	ENVIRONMENT VARIABLES	
79031		The following environment variables shall affect the execution of <i>chmod</i> :
79032	<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)
79033		
79034		
79035	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
79036		
79037	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
79038		
79039		
79040	<i>LC_MESSAGES</i>	
79041		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
79042		
79043	XSI	<i>NLSPATH</i> Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .

79044 ASYNCHRONOUS EVENTS

79045 Default.

79046 STDOUT

79047 Not used.

79048 STDERR

79049 The standard error shall be used only for diagnostic messages.

79050 OUTPUT FILES

79051 None.

79052 EXTENDED DESCRIPTION

79053 The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The
79054 *symbolic_mode* form is described by the grammar later in this section.79055 Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*.
79056 The operations shall be performed on each *file* in the order in which the **clauses** are specified.79057 The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits,
79058 respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.79059 The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute/search* portions of file mode
79060 bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who**
79061 contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.79062 The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a
79063 directory or if the current (unmodified) file mode bits have at least one of the execute bits
79064 (S_IXUSR, S_IXGRP, or S_IXOTH) set. It shall be ignored if the file is not a directory and none of
79065 the execute bits are set in the current file mode bits.79066 The **permcop** symbols **u**, **g**, and **o** shall represent the current permissions associated with the
79067 user, group, and other parts of the file mode bits, respectively. For the remainder of this section,
79068 **perm** refers to the non-terminals **perm** and **permcop** in the grammar.79069 If multiple **actionlists** are grouped with a single **wholist** in the grammar, each **actionlist** shall be
79070 applied in the order specified with that **wholist**. The *op* symbols shall represent the operation
79071 performed, as follows:79072 + If **perm** is not specified, the '+' operation shall not change the file mode bits.79073 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
79074 other permissions, except for those with corresponding bits in the file mode creation mask
79075 of the invoking process, shall be set.79076 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.79077 - If **perm** is not specified, the '-' operation shall not change the file mode bits.79078 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
79079 other permissions, except for those with corresponding bits in the file mode creation mask
79080 of the invoking process, shall be cleared.79081 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be
79082 cleared.79083 = Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the
79084 file mode bits specified in this volume of POSIX.1-200x.79085 If **perm** is not specified, the '=' operation shall make no further modifications to the file
79086 mode bits.79087 If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and
79088 other permissions, except for those with corresponding bits in the file mode creation mask

79089 of the invoking process, shall be set.

79090 Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

79091 When using the symbolic mode form on a regular file, it is implementation-defined whether or
79092 not:

- 79093 • Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all
79094 execute bits are currently clear and none are being set are ignored.
- 79095 • Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-
79096 on-execution bits.
- 79097 • Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all
79098 execute bits are currently clear are ignored. However, if the command `ls -l file` writes an *s*
79099 in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is
79100 set, the commands `chmod u-s file` or `chmod g-s file`, respectively, shall not be ignored.

79101 When using the symbolic mode form on other file types, it is implementation-defined whether
79102 or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
79103 honored.

79104 If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols
79105 being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be
79106 modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm**
79107 symbol **s**.

79108 XSI The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can
79109 be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who**
79110 symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these
79111 combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other
79112 than directory is unspecified.

79113 For an octal integer *mode* operand, the file mode bits shall be set absolutely.

79114 For each bit set in the octal number, the corresponding file permission bit shown in the following
79115 table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in
79116 the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-
79117 execution, bits shown in the following table shall be set; if these bits are not set in the octal
79118 number, they are cleared. For other file types, it is implementation-defined whether or not
79119 requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are
79120 honored.

79121	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit	Octal	Mode Bit
79122	4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
79123	2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
79124	XSI 1000	S_ISVTX	0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

79125 When bits are set in the octal number other than those listed in the table above, the behavior is
79126 unspecified.

79127

Grammar for chmod79128
79129
79130
79131
79132

The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode* operand. The general conventions for this style of grammar are described in [Section 1.3](#) (on page 2235). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

79133
79134

The lexical processing is based entirely on single characters. Implementations need not allow <blank>s within the single argument being processed.

79135
79136

```
%start    symbolic_mode
%%
```

79137
79138
79139

```
symbolic_mode    : clause
                  | symbolic_mode ',' clause
                  ;
```

79140
79141
79142

```
clause           : actionlist
                  | wholist actionlist
                  ;
```

79143
79144
79145

```
wholist         : who
                  | wholist who
                  ;
```

79146
79147

```
who             : 'u' | 'g' | 'o' | 'a'
                  ;
```

79148
79149
79150

```
actionlist      : action
                  | actionlist action
                  ;
```

79151
79152
79153
79154

```
action          : op
                  | op permlist
                  | op permcopy
                  ;
```

79155
79156

```
permcopy       : 'u' | 'g' | 'o'
                  ;
```

79157
79158

```
op             : '+' | '-' | '='
                  ;
```

79159
79160
79161

```
permlist       : perm
                  | perm permlist
                  ;
```

79162
79163

```
XSI perm       : 'r' | 'w' | 'x' | 'X' | 's' | 't'
                  ;
```

79164

EXIT STATUS

79165

The following exit values shall be returned:

79166

0 The utility executed successfully and all requested changes were made.

79167

>0 An error occurred.

79168

CONSEQUENCES OF ERRORS

79169

Default.

APPLICATION USAGE

Some implementations of the *chmod* utility change the mode of a directory before the files in the directory when performing a recursive (**-R** option) change; others change the directory mode after the files in the directory. If an application tries to remove read or search permission for a file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users should not try to make a hierarchy inaccessible to themselves.

Some implementations of *chmod* never used the *umask* of the process when changing modes; systems conformant with this volume of POSIX.1-200x do so when **who** is not specified. Note the difference between:

```
chmod a-w file
```

which removes all write permissions, and:

```
chmod -- -w file
```

which removes write permissions that would be allowed if **file** was created with the same *umask*.

Conforming applications should never assume that they know how the set-user-ID and set-group-ID bits on directories are interpreted.

EXAMPLES

Mode	Results
<i>a+=</i>	Equivalent to <i>a+,a=</i> ; clears all file mode bits.
<i>go+-w</i>	Equivalent to <i>go+,go-w</i> ; clears group and other write bits.
<i>g=o-w</i>	Equivalent to <i>g=o,g-w</i> ; sets group bit to match other bits and then clears group write bit.
<i>g-r+w</i>	Equivalent to <i>g-r,g+w</i> ; clears group read bit and sets group write bit.
<i>uo=g</i>	Sets owner bits to match group bits and sets other bits to match group bits.

RATIONALE

The functionality of *chmod* is described substantially through references to concepts defined in the System Interfaces volume of POSIX.1-200x. In this way, there is less duplication of effort required for describing the interactions of permissions. However, the behavior of this utility is not described in terms of the *chmod()* function from the System Interfaces volume of POSIX.1-200x because that specification requires certain side effects upon alternate file access control mechanisms that might not be appropriate, depending on the implementation.

Implementations that support mandatory file and record locking as specified by the 1984 /usr/group standard historically used the combination of set-group-ID bit set and group execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory locking mode is not changed without explicit indication that that was what the user intended. Therefore, the details on how the implementation treats these conditions must be defined in the documentation. This volume of POSIX.1-200x does not require mandatory locking (nor does the System Interfaces volume of POSIX.1-200x), but does allow it as an extension. However, this volume of POSIX.1-200x does require that the *ls* and *chmod* utilities work consistently in this area. If *ls -l file* indicates that the set-group-ID bit is set, *chmod g-s file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it

79218 can overflow the range of valid exit status values. This problem is avoided here by specifying
79219 only 0 and >0 as exit values.

79220 The System Interfaces volume of POSIX.1-200x indicates that implementation-defined
79221 restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of
79222 POSIX.1-200x allows the *chmod* utility to choose to modify these bits before calling *chmod()* (or
79223 some function providing equivalent capabilities) for non-regular files. Among other things, this
79224 allows implementations that use the set-user-ID and set-group-ID bits on directories to enable
79225 extended features to handle these extensions in an intelligent manner.

79226 The **X perm** symbol was adopted from BSD-based systems because it provides commonly
79227 desired functionality when doing recursive (**-R** option) modifications. Similar functionality is
79228 not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X**
79229 with *op+*; it has been extended in this volume of POSIX.1-200x because it is also useful with *op=*.
79230 (It has also been added for *op-* even though it duplicates **x**, in this case, because it is intuitive
79231 and easier to explain.)

79232 The grammar was extended with the *permcop* non-terminal to allow historical-practice forms of
79233 symbolic modes like **o=u -g** (that is, set the “other” permissions to the permissions of “owner”
79234 minus the permissions of “group”).

79235 FUTURE DIRECTIONS

79236 None.

79237 SEE ALSO

79238 *ls*, *umask*

79239 XBD [Section 4.4](#) (on page 96), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

79240 XSH *chmod*

79241 CHANGE HISTORY

79242 First released in Issue 2.

79243 Issue 6

79244 The following new requirements on POSIX implementations derive from alignment with the
79245 Single UNIX Specification:

- 79246 • Octal modes have been kept and made mandatory despite being marked obsolescent in the
79247 ISO POSIX-2:1993 standard.

79248 IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS
79249 section to “Default.”.

79250 The Open Group Base Resolution bwg2001-010 is applied, adding the description of the
79251 S_ISVTX bit and the **t perm** symbol as part of the XSI option.

79252 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/16 is applied, changing the XSI shaded
79253 text in the EXTENDED DESCRIPTION from:

79254 “The **perm** symbol **t** shall specify the S_ISVTX bit and shall apply to directories only. The
79255 effect when using it with any other file type is unspecified. It can be used with the **who**
79256 symbols **o**, **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**
79257 or **g** in conjunction with the **perm** symbol **t**; it shall be ignored for **u** and **g**.”

79258 to:

79259
79260
79261
79262
79263

“The **perm** symbol **t** shall specify the S_ISVTX bit. When used with a file of type directory, it can be used with the **who** symbol **a**, or with no **who** symbol. It shall not be an error to specify a **who** symbol of **u**, **g**, or **o** in conjunction with the **perm** symbol **t**, but the meaning of these combinations is unspecified. The effect when using the **perm** symbol **t** with any file type other than directory is unspecified.”

79264

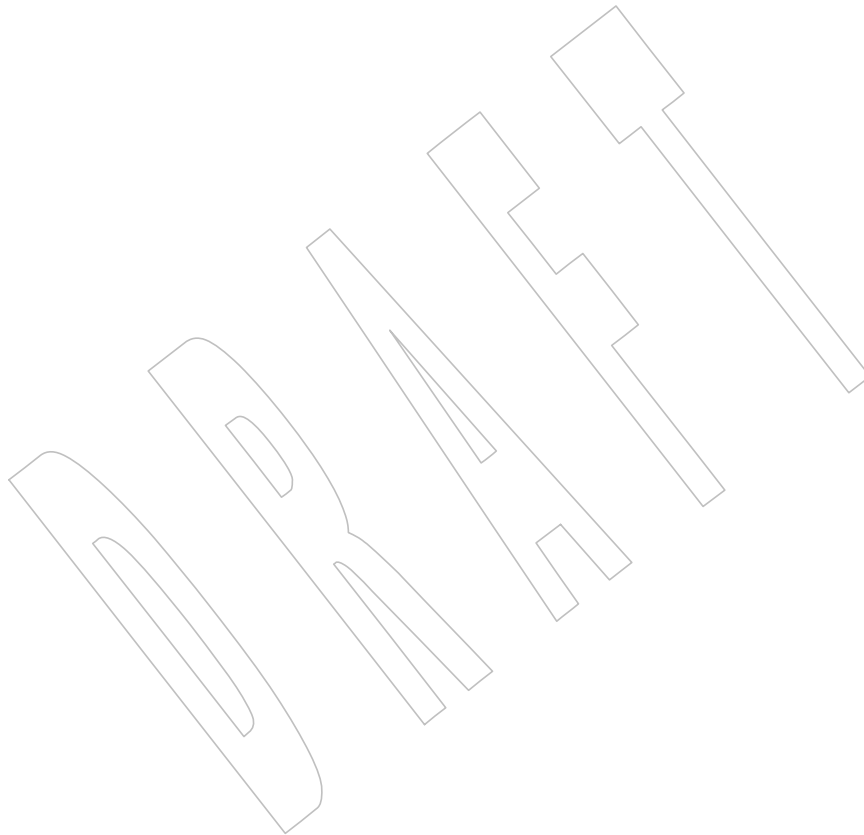
This change is to permit historical behavior.

79265

Issue 7

79266

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



79267 **NAME**

79268 chown — change the file ownership

79269 **SYNOPSIS**

79270 chown [-h] owner[:group] file...

79271 chown -R [-H|-L|-P] owner[:group] file...

79272 **DESCRIPTION**79273 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID
79274 specified by the *owner* operand.79275 For each *file* operand, or, if the **-R** option is used, each file encountered while walking the
79276 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to
79277 the *chown()* function defined in the System Interfaces volume of POSIX.1-200x, called with the
79278 following arguments:

- 79279
1. The *file* operand shall be used as the *path* argument.
 - 79280 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*
79281 argument.
 - 79282 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used
79283 as the *group* argument; otherwise, the group ownership shall not be changed.

79284 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-group-
79285 ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-
79286 group-ID bits of other file types may be cleared.79287 **OPTIONS**79288 The *chown* utility shall conform to XBD [Section 12.2](#) (on page 201).

79289 The following options shall be supported by the implementation:

- 79290
- h**
- If the system supports user IDs for symbolic links, for each
- file*
- operand that names
-
- 79291 a file of type symbolic link,
- chown*
- shall attempt to set the user ID of the symbolic
-
- 79292 link. If the system supports group IDs for symbolic links, and a group ID was
-
- 79293 specified, for each
- file*
- operand that names a file of type symbolic link,
- chown*
- shall
-
- 79294 attempt to set the group ID of the symbolic link. If the system does not support
-
- 79295 user or group IDs for symbolic links, for each
- file*
- operand that names a file of type
-
- 79296 symbolic link,
- chown*
- shall do nothing more with the current file and shall go on to
-
- 79297 any remaining files.
-
- 79298
- H**
- If the
- R**
- option is specified and a symbolic link referencing a file of type directory
-
- 79299 is specified on the command line,
- chown*
- shall change the user ID (and group ID, if
-
- 79300 specified) of the directory referenced by the symbolic link and all files in the file
-
- 79301 hierarchy below it.
-
- 79302
- L**
- If the
- R**
- option is specified and a symbolic link referencing a file of type directory
-
- 79303 is specified on the command line or encountered during the traversal of a file
-
- 79304 hierarchy,
- chown*
- shall change the user ID (and group ID, if specified) of the
-
- 79305 directory referenced by the symbolic link and all files in the file hierarchy below it.
-
- 79306
- P**
- If the
- R**
- option is specified and a symbolic link is specified on the command line
-
- 79307 or encountered during the traversal of a file hierarchy,
- chown*
- shall change the
-
- 79308 owner ID (and group ID, if specified) of the symbolic link if the system supports
-
- 79309 this operation. The
- chown*
- utility shall not follow the symbolic link to any other part
-
- 79310 of the file hierarchy.

79311 **-R** Recursively change file user and group IDs. For each *file* operand that names a
79312 directory, *chown* shall change the user ID (and group ID, if specified) of the
79313 directory and all files in the file hierarchy below it. Unless a **-H**, **-L**, or **-P** option is
79314 specified, it is unspecified which of these options will be used as the default.

79315 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
79316 considered an error. The last option specified shall determine the behavior of the utility.

79317 OPERANDS

79318 The following operands shall be supported:

79319 *owner[:group]* A user ID and optional group ID to be assigned to *file*. The *owner* portion of this
79320 operand shall be a user name from the user database or a numeric user ID. Either
79321 specifies a user ID which shall be given to each file named by one of the *file*
79322 operands. If a numeric *owner* operand exists in the user database as a user name,
79323 the user ID number associated with that user name shall be used as the user ID.
79324 Similarly, if the *group* portion of this operand is present, it shall be a group name
79325 from the group database or a numeric group ID. Either specifies a group ID which
79326 shall be given to each file. If a numeric group operand exists in the group database
79327 as a group name, the group ID number associated with that group name shall be
79328 used as the group ID.

79329 *file* A pathname of a file whose user ID is to be modified.

79330 STDIN

79331 Not used.

79332 INPUT FILES

79333 None.

79334 ENVIRONMENT VARIABLES

79335 The following environment variables shall affect the execution of *chown*:

79336 **LANG** Provide a default value for the internationalization variables that are unset or null. |
79337 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
79338 variables used to determine the values of locale categories.)

79339 **LC_ALL** If set to a non-empty string value, override the values of all the other
79340 internationalization variables.

79341 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
79342 characters (for example, single-byte as opposed to multi-byte characters in
79343 arguments).

79344 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
79345 diagnostic messages written to standard error.
79346

79347 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

79348 ASYNCHRONOUS EVENTS

79349 Default.

79350 STDOUT

79351 Not used.

79352 STDERR

79353 The standard error shall be used only for diagnostic messages.

79354 **OUTPUT FILES**

79355 None.

79356 **EXTENDED DESCRIPTION**

79357 None.

79358 **EXIT STATUS**

79359 The following exit values shall be returned:

79360 0 The utility executed successfully and all requested changes were made.

79361 >0 An error occurred.

79362 **CONSEQUENCES OF ERRORS**

79363 Default.

79364 **APPLICATION USAGE**79365 Only the owner of a file or the user with appropriate privileges may change the owner or group
79366 of a file.79367 Some implementations restrict the use of *chown* to a user with appropriate privileges.79368 **EXAMPLES**

79369 None.

79370 **RATIONALE**79371 The System V and BSD versions use different exit status codes. Some implementations used the
79372 exit status as a count of the number of errors that occurred; this practice is unworkable since it
79373 can overflow the range of valid exit status values. These are masked by specifying only 0 and >0
79374 as exit values.79375 The functionality of *chown* is described substantially through references to functions in the
79376 System Interfaces volume of POSIX.1-200x. In this way, there is no duplication of effort required
79377 for describing the interactions of permissions, multiple groups, and so on.79378 The 4.3 BSD method of specifying both owner and group was included in this volume of
79379 POSIX.1-200x because:

- 79380 • There are cases where the desired end condition could not be achieved using the *chgrp* and
79381 *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the
79382 desired group and the desired owner is not a member of the current group, the *chown()*
79383 function could fail unless both owner and group are changed at the same time.)
- 79384 • Even if they could be changed independently, in cases where both are being changed, there
79385 is a 100% performance penalty caused by being forced to invoke both utilities.

79386 The BSD syntax *user[.group]* was changed to *user[:group]* in this volume of POSIX.1-200x because
79387 the period is a valid character in login names (as specified by the Base Definitions volume of
79388 POSIX.1-200x, login names consist of characters in the portable filename character set). The
79389 colon character was chosen as the replacement for the period character because it would never
79390 be allowed as a character in a user name or group name on historical implementations.

79391 The **-R** option is considered by some observers as an undesirable departure from the historical
79392 UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there
79393 seemed to be no good reason to require other tools to have to duplicate that functionality.
79394 However, the **-R** option was deemed an important user convenience, is far more efficient than
79395 forking a separate process for each element of the directory hierarchy, and is in widespread
79396 historical use.

79397
79398
79399
79400
79401
79402
79403
79404
79405
79406
79407
79408
79409
79410
79411
79412
79413
79414
79415
79416
79417

FUTURE DIRECTIONS

None.

SEE ALSO

chgrp, chmod

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

XSH *chown*

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **-h**, **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to “Default.”.

The “otherwise, ...” text in item 3. of the DESCRIPTION is changed to “otherwise, the group ownership shall not be changed”.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/17 is applied, changing the SYNOPSIS to make it clear that **-h** and **-R** are optional.

Issue 7

SD5-XCU-ERN-9 is applied, removing the **-R** from the first line of the SYNOPSIS.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79418 **NAME**

79419 cksum — write file checksums and sizes

79420 **SYNOPSIS**79421 cksum [*file*...]79422 **DESCRIPTION**

79423 The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC)
 79424 for each input file, and also write to standard output the number of octets in each file. The CRC
 79425 used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996
 79426 standard (Ethernet).

79427 The encoding for the CRC checksum is defined by the generating polynomial:

$$79428 G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

79429 Mathematically, the CRC value corresponding to a given file shall be defined by the following
 79430 procedure:

- 79431 1. The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial
 79432 *M(x)* of degree *n*−1. These *n* bits are the bits from the file, with the most significant bit
 79433 being the most significant bit of the first octet of the file and the last bit being the least
 79434 significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral
 79435 number of octets, followed by one or more octets representing the length of the file as a
 79436 binary value, least significant octet first. The smallest number of octets capable of
 79437 representing this integer shall be used.
- 79438 2. *M(x)* is multiplied by x^{32} (that is, shifted left 32 bits) and divided by *G(x)* using mod 2
 79439 division, producing a remainder *R(x)* of degree ≤ 31.
- 79440 3. The coefficients of *R(x)* are considered to be a 32-bit sequence.
- 79441 4. The bit sequence is complemented and the result is the CRC.

79442 **OPTIONS**

79443 None.

79444 **OPERANDS**

79445 The following operand shall be supported:

79446 *file* A pathname of a file to be checked. If no *file* operands are specified, the standard
 79447 input shall be used.

79448 **STDIN**

79449 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
 79450 section.

79451 **INPUT FILES**

79452 The input files can be any file type.

79453 **ENVIRONMENT VARIABLES**79454 The following environment variables shall affect the execution of *cksum*:

79455 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 79456 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 79457 variables used to determine the values of locale categories.)

79458 *LC_ALL* If set to a non-empty string value, override the values of all the other
 79459 internationalization variables.

79460 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 79461 characters (for example, single-byte as opposed to multi-byte characters in
 79462 arguments).

79463 *LC_MESSAGES*
 79464 Determine the locale that should be used to affect the format and contents of
 79465 diagnostic messages written to standard error.

79466 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

79467 **ASYNCHRONOUS EVENTS**

79468 Default.

79469 **STDOUT**

79470 For each file processed successfully, the *cksum* utility shall write in the following format:

79471 "%u %d %s\n", <checksum>, <# of octets>, <pathname>

79472 If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

79473 **STDERR**

79474 The standard error shall be used only for diagnostic messages.

79475 **OUTPUT FILES**

79476 None.

79477 **EXTENDED DESCRIPTION**

79478 None.

79479 **EXIT STATUS**

79480 The following exit values shall be returned:

79481 0 All files were processed successfully.

79482 >0 An error occurred.

79483 **CONSEQUENCES OF ERRORS**

79484 Default.

79485 **APPLICATION USAGE**

79486 The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of
 79487 the same, such as to ensure that files transmitted over noisy media arrive intact. However, this
 79488 comparison cannot be considered cryptographically secure. The chances of a damaged file
 79489 producing the same CRC as the original are small; deliberate deception is difficult, but probably
 79490 not impossible.

79491 Although input files to *cksum* can be any type, the results need not be what would be expected
 79492 on character special device files or on file types not described by the System Interfaces volume of
 79493 POSIX.1-200x. Since this volume of POSIX.1-200x does not specify the block size used when
 79494 doing input, checksums of character special files need not process all of the data in those files.

79495 The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted
 79496 between two systems and undergoes any data transformation (such as changing little-endian
 79497 byte ordering to big-endian), identical CRC values cannot be expected. Implementations
 79498 performing such transformations may extend *cksum* to handle such situations.

79499 **EXAMPLES**

79500 None.

79501
79502
79503
79504
79505

79506
79507
79508
79509
79510
79511
79512
79513
79514
79515
79516
79517
79518
79519
79520
79521
79522
79523
79524
79525
79526
79527
79528
79529
79530
79531
79532
79533
79534
79535
79536
79537
79538
79539
79540
79541
79542
79543
79544
79545
79546
79547
79548
79549
79550
79551
79552
79553
79554**RATIONALE**

The following C-language program can be used as a model to describe the algorithm. It assumes that a **char** is one octet. It also assumes that the entire file is available for one pass through the function. This was done for simplicity in demonstrating the algorithm, rather than as an implementation model.

```
static unsigned long crctab[] = {
0x00000000,
0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbd6d,
0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xdf37027, 0xddb056fe,
0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
0xe5ffe643, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcd6bb16,
0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93d6db, 0x6f52c06c,
0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
0xc27dede8, 0xcf3ecb31, 0xcbfd686, 0xd5b88683, 0xd1799b34,
0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcd6fd59, 0x608edb80,
0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9fff77d71,
0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
0x4e8ee645, 0x4a4ffbf2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fcd1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xeeee2ed18,
```

```

79555     0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
79556     0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
79557     0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
79558     0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
79559     };

79560     unsigned long memcrc(const unsigned char *b, size_t n)
79561     {
79562     /* Input arguments:
79563     * const char*   b == byte sequence to checksum
79564     * size_t       n == length of sequence
79565     */

79566         register unsigned   i, c, s = 0;

79567         for (i = n; i > 0; --i) {
79568             c = (unsigned)(*b++);
79569             s = (s << 8) ^ crctab[(s >> 24) ^ c];
79570         }

79571         /* Extend with the length of the string. */
79572         while (n != 0) {
79573             c = n & 0377;
79574             n >>= 8;
79575             s = (s << 8) ^ crctab[(s >> 24) ^ c];
79576         }

79577         return ~s;
79578     }

```

79579 The historical practice of writing the number of “blocks” has been changed to writing the
79580 number of octets, since the latter is not only more useful, but also since historical
79581 implementations have not been consistent in defining what a “block” meant.

79582 The algorithm used was selected to increase the operational robustness of *cksum*. Neither the
79583 System V nor BSD *sum* algorithm was selected. Since each of these was different and each was
79584 the default behavior on those systems, no realistic compromise was available if either were
79585 selected—some set of historical applications would break. Therefore, the name was changed to
79586 *cksum*. Although the historical *sum* commands will probably continue to be provided for many
79587 years, programs designed for portability across systems should use the new name.

79588 The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for
79589 the frame check sequence field. The algorithm used does not match the technical definition of a
79590 *checksum*; the term is used for historical reasons. The length of the file is included in the CRC
79591 calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also
79592 because it guards against inadvertent collisions between files that begin with different series of
79593 zero octets. The chance that two different files produce identical CRCs is much greater when
79594 their lengths are not considered. Keeping the length and the checksum of the file itself separate
79595 would yield a slightly more robust algorithm, but historical usage has always been that a single
79596 number (the checksum as printed) represents the signature of the file. It was decided that
79597 historical usage was the more important consideration.

79598 Early proposals contained modifications to the Ethernet algorithm that involved extracting table
79599 values whenever an intermediate result became zero. This was demonstrated to be less robust
79600 than the current method and mathematically difficult to describe or justify.

79601 The calculation used is identical to that given in pseudo-code in the referenced Sarwate article.
79602 The pseudo-code rendition is:

```

79603     X <- 0; Y <- 0;

```

```

79604     for i <- m -1 step -1 until 0 do
79605         begin
79606             T <- X(1) ^ A[i];
79607             X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
79608             comment: f[T] and f'[T] denote the T-th words in the
79609                 table f and f' ;
79610             X <- X ^ f[T]; Y <- Y ^ f'[T];
79611         end

```

79612 The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]**
79613 represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables
79614 **f** and **f'** are a single table containing 32-bit values.

79615 The referenced Sarwate article also discusses generating the table.

79616 **FUTURE DIRECTIONS**

79617 None.

79618 **SEE ALSO**

79619 XBD [Chapter 8](#) (on page 159)

79620 **CHANGE HISTORY**

79621 First released in Issue 4.

79622 **Issue 7**

79623 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79624 **NAME**

79625 cmp — compare two files

79626 **SYNOPSIS**

79627 cmp [-l|-s] file1 file2

79628 **DESCRIPTION**

79629 The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the
 79630 same. Under default options, if they differ, it shall write to standard output the byte and line
 79631 number at which the first difference occurred. Bytes and lines shall be numbered beginning with
 79632 1.

79633 **OPTIONS**79634 The *cmp* utility shall conform to XBD Section 12.2 (on page 201).

79635 The following options shall be supported:

79636 **-l** (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for
 79637 each difference.

79638 **-s** Write nothing for differing files; return exit status only.

79639 **OPERANDS**

79640 The following operands shall be supported:

79641 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall
 79642 be used.

79643 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input
 79644 shall be used.

79645 If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or
 79646 character special file, the results are undefined.

79647 **STDIN**

79648 The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the
 79649 INPUT FILES section.

79650 **INPUT FILES**

79651 The input files can be any file type.

79652 **ENVIRONMENT VARIABLES**79653 The following environment variables shall affect the execution of *cmp*:

79654 **LANG** Provide a default value for the internationalization variables that are unset or null.
 79655 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 79656 variables used to determine the values of locale categories.)

79657 **LC_ALL** If set to a non-empty string value, override the values of all the other
 79658 internationalization variables.

79659 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 79660 characters (for example, single-byte as opposed to multi-byte characters in
 79661 arguments).

79662 **LC_MESSAGES**

79663 Determine the locale that should be used to affect the format and contents of
 79664 diagnostic messages written to standard error and informative messages written to
 79665 standard output.

79666 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

79667 ASYNCHRONOUS EVENTS

79668 Default.

79669 STDOUT

79670 In the POSIX locale, results of the comparison shall be written to standard output. When no
79671 options are used, the format shall be:

79672 "%s %s differ: char %d, line %d\n", *file1*, *file2*,
79673 <*byte number*>, <*line number*>

79674 When the *-l* option is used, the format shall be:

79675 "%d %o %o\n", <*byte number*>, <*differing byte*>,
79676 <*differing byte*>

79677 for each byte that differs. The first <*differing byte*> number is from *file1* while the second is from
79678 *file2*. In both cases, <*byte number*> shall be relative to the beginning of the file, beginning with 1.

79679 No output shall be written to standard output when the *-s* option is used.

79680 STDERR

79681 The standard error shall be used only for diagnostic messages. If the *-l* option is used and *file1*
79682 and *file2* differ in length, or if the *-s* option is not used and *file1* and *file2* are identical for the
79683 entire length of the shorter file, in the POSIX locale the following diagnostic message shall be
79684 written:

79685 "cmp: EOF on %s%s\n", <*name of shorter file*>, <*additional info*>

79686 The <*additional info*> field shall either be null or a string that starts with a <blank> and contains
79687 no <newline>s. Some implementations report on the number of lines in this case.

79688 OUTPUT FILES

79689 None.

79690 EXTENDED DESCRIPTION

79691 None.

79692 EXIT STATUS

79693 The following exit values shall be returned:

79694 0 The files are identical.

79695 1 The files are different; this includes the case where one file is identical to the first part of the
79696 other.

79697 >1 An error occurred.

79698 CONSEQUENCES OF ERRORS

79699 Default.

79700 APPLICATION USAGE

79701 Although input files to *cmp* can be any type, the results might not be what would be expected on
79702 character special device files or on file types not described by the System Interfaces volume of
79703 POSIX.1-200x. Since this volume of POSIX.1-200x does not specify the block size used when
79704 doing input, comparisons of character special files need not compare all of the data in those files.

79705 For files which are not text files, line numbers simply reflect the presence of a <newline>,
79706 without any implication that the file is organized into lines.

79707
79708
79709
79710
79711
79712
79713
79714
79715
79716
79717
79718
79719
79720
79721
79722
79723
79724
79725
79726
79727
79728
79729
79730
79731
79732
79733
79734
79735
79736

EXAMPLES

None.

RATIONALE

The global language in [Section 1.4](#) (on page 2235) indicates that using two mutually-exclusive options together produces unspecified results. Some System V implementations consider the option usage:

```
cmp -l -s ...
```

to be an error. They also treat:

```
cmp -s -l ...
```

as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

The word **char** in the standard output format comes from historical usage, even though it is actually a byte number. When *cmp* is supported in other locales, implementations are encouraged to use the word *byte* or its equivalent in another language. Users should not interpret this difference to indicate that the functionality of the utility changed between locales.

Some implementations report on the number of lines in the identical-but-shorter file case. This is allowed by the inclusion of the *<additional info>* fields in the output format. The restriction on having a leading *<blank>* and no *<newline>*s is to make parsing for the filename easier. It is recognized that some filenames containing white-space characters make parsing difficult anyway, but the restriction does aid programs used on systems where the names are predominantly well behaved.

FUTURE DIRECTIONS

None.

SEE ALSO

comm, *diff*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

First released in Issue 2.

Issue 7

SD5-XCU-ERN-96 is applied, updating the STDERR section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

79737 **NAME**
 79738 comm — select or reject lines common to two files

79739 **SYNOPSIS**
 79740 comm [-123] *file1 file2*

79741 **DESCRIPTION**
 79742 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating
 79743 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and
 79744 lines in both files.

79745 If the lines in both files are not ordered according to the collating sequence of the current locale,
 79746 the results are unspecified.

79747 **OPTIONS**
 79748 The *comm* utility shall conform to XBD [Section 12.2](#) (on page 201).

79749 The following options shall be supported:

- 79750 -1 Suppress the output column of lines unique to *file1*.
- 79751 -2 Suppress the output column of lines unique to *file2*.
- 79752 -3 Suppress the output column of lines duplicated in *file1* and *file2*.

79753 **OPERANDS**
 79754 The following operands shall be supported:

79755 *file1* A pathname of the first file to be compared. If *file1* is '-', the standard input shall
 79756 be used.

79757 *file2* A pathname of the second file to be compared. If *file2* is '-', the standard input
 79758 shall be used.

79759 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or
 79760 character special file, the results are undefined.

79761 **STDIN**
 79762 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.
 79763 See the INPUT FILES section.

79764 **INPUT FILES**
 79765 The input files shall be text files.

79766 **ENVIRONMENT VARIABLES**
 79767 The following environment variables shall affect the execution of *comm*:

79768 *LANG* Provide a default value for the internationalization variables that are unset or null.
 79769 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 79770 variables used to determine the values of locale categories.)

79771 *LC_ALL* If set to a non-empty string value, override the values of all the other
 79772 internationalization variables.

79773 *LC_COLLATE*
 79774 Determine the locale for the collating sequence *comm* expects to have been used
 79775 when the input files were sorted.

79776 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 79777 characters (for example, single-byte as opposed to multi-byte characters in
 79778 arguments and input files).

79779 *LC_MESSAGES*
 79780 Determine the locale that should be used to affect the format and contents of
 79781 diagnostic messages written to standard error.

79782 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

79783 ASYNCHRONOUS EVENTS

79784 Default.

79785 STDOUT

79786 The *comm* utility shall produce output depending on the options selected. If the *-1*, *-2*, and *-3*
 79787 options are all selected, *comm* shall write nothing to standard output.

79788 If the *-1* option is not selected, lines contained only in *file1* shall be written using the format:

79789 "%s\n", <line in file1>

79790 If the *-2* option is not selected, lines contained only in *file2* are written using the format:

79791 "%s%s\n", <lead>, <line in file2>

79792 where the string <lead> is as follows:

79793 <tab> The *-1* option is not selected.

79794 null string The *-1* option is selected.

79795 If the *-3* option is not selected, lines contained in both files shall be written using the format:

79796 "%s%s\n", <lead>, <line in both>

79797 where the string <lead> is as follows:

79798 <tab><tab> Neither the *-1* nor the *-2* option is selected.

79799 <tab> Exactly one of the *-1* and *-2* options is selected.

79800 null string Both the *-1* and *-2* options are selected.

79801 If the input files were ordered according to the collating sequence of the current locale, the lines
 79802 written shall be in the collating sequence of the original lines.

79803 STDERR

79804 The standard error shall be used only for diagnostic messages.

79805 OUTPUT FILES

79806 None.

79807 EXTENDED DESCRIPTION

79808 None.

79809 EXIT STATUS

79810 The following exit values shall be returned:

79811 0 All input files were successfully output as specified.

79812 >0 An error occurred.

79813 CONSEQUENCES OF ERRORS

79814 Default.

79815 **APPLICATION USAGE**79816 If the input files are not properly presorted, the output of *comm* might not be useful.79817 **EXAMPLES**79818 If a file named **xcu** contains a sorted list of the utilities in this volume of POSIX.1-200x, a file
79819 named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue
79820 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface
79821 Definition Third Edition:79822 `comm -23 xcu xpg3 | comm -23 - svid89`79823 would print a list of utilities in this volume of POSIX.1-200x not specified by either of the other
79824 documents:79825 `comm -12 xcu xpg3 | comm -12 - svid89`

79826 would print a list of utilities specified by all three documents, and:

79827 `comm -12 xpg3 svid89 | comm -23 - xcu`79828 would print a list of utilities specified by both XPG3 and the SVID, but not specified in this
79829 volume of POSIX.1-200x.79830 **RATIONALE**

79831 None.

79832 **FUTURE DIRECTIONS**

79833 None.

79834 **SEE ALSO**79835 *cmp, diff, sort, uniq*79836 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +79837 **CHANGE HISTORY**

79838 First released in Issue 2.

79839 **Issue 6**

79840 The normative text is reworded to avoid use of the term “must” for application requirements.

79841 **NAME**
 79842 `command` — execute a simple command

79843 **SYNOPSIS**
 79844 `command [-p] command_name [argument...]`
 79845 `command [-v|-V] command_name`

79846 **DESCRIPTION**
 79847 The *command* utility shall cause the shell to treat the arguments as a simple command,
 79848 suppressing the shell function lookup that is described in [Section 2.9.1.1](#) (on page 2264), item 1b.

79849 If the *command_name* is the same as the name of one of the special built-in utilities, the special
 79850 properties in the enumerated list at the beginning of [Section 2.14](#) (on page 2280) shall not occur.
 79851 In every other respect, if *command_name* is not the name of a function, the effect of *command*
 79852 (with no options) shall be the same as omitting *command*.

79853 When the `-v` or `-V` option is used, the *command* utility shall provide information concerning
 79854 how a command name is interpreted by the shell.

79855 **OPTIONS**
 79856 The *command* utility shall conform to XBD [Section 12.2](#) (on page 201).

79857 The following options shall be supported:

79858 `-p` Perform the command search using a default value for *PATH* that is guaranteed to
 79859 find all of the standard utilities.

79860 `-v` Write a string to standard output that indicates the pathname or command that
 79861 will be used by the shell, in the current shell execution environment (see [Section](#)
 79862 [2.12](#), on page 2277), to invoke *command_name*, but do not invoke *command_name*.

79863 • Utilities, regular built-in utilities, *command_names* including a slash character,
 79864 and any implementation-defined functions that are found using the *PATH*
 79865 variable (as described in [Section 2.9.1.1](#), on page 2264), shall be written as
 79866 absolute pathnames.

79867 • Shell functions, special built-in utilities, regular built-in utilities not
 79868 associated with a *PATH* search, and shell reserved words shall be written as
 79869 just their names.

79870 • An alias shall be written as a command line that represents its alias
 79871 definition.

79872 • Otherwise, no output shall be written and the exit status shall reflect that the
 79873 name was not found.

79874 `-V` Write a string to standard output that indicates how the name given in the
 79875 *command_name* operand will be interpreted by the shell, in the current shell
 79876 execution environment (see [Section 2.12](#), on page 2277), but do not invoke
 79877 *command_name*. Although the format of this string is unspecified, it shall indicate
 79878 in which of the following categories *command_name* falls and shall include the
 79879 information stated:

79880 • Utilities, regular built-in utilities, and any implementation-defined functions
 79881 that are found using the *PATH* variable (as described in [Section 2.9.1.1](#), on
 79882 page 2264), shall be identified as such and include the absolute pathname in
 79883 the string.

- 79884 • Other shell functions shall be identified as functions.
- 79885 • Aliases shall be identified as aliases and their definitions included in the
- 79886 string.
- 79887 • Special built-in utilities shall be identified as special built-in utilities.
- 79888 • Regular built-in utilities not associated with a *PATH* search shall be identified
- 79889 as regular built-in utilities. (The term “regular” need not be used.)
- 79890 • Shell reserved words shall be identified as reserved words.

OPERANDS

The following operands shall be supported:

- 79893 *argument* One of the strings treated as an argument to *command_name*.
- 79894 *command_name*
- 79895 The name of a utility or a special built-in utility.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *command*:

- 79902 *LANG* Provide a default value for the internationalization variables that are unset or null. |
79903 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
79904 variables used to determine the values of locale categories.)
- 79905 *LC_ALL* If set to a non-empty string value, override the values of all the other
79906 internationalization variables.
- 79907 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
79908 characters (for example, single-byte as opposed to multi-byte characters in
79909 arguments).
- 79910 *LC_MESSAGES*
- 79911 Determine the locale that should be used to affect the format and contents of
79912 diagnostic messages written to standard error and informative messages written to
79913 standard output.
- 79914 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 79915 *PATH* Determine the search path used during the command search described in [Section](#)
79916 [2.9.1.1](#) (on page 2264), except as described under the **-p** option.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When the **-v** option is specified, standard output shall be formatted as:

79921 "%s\n", *<pathname or command>*

When the **-V** option is specified, standard output shall be formatted as:

79923 "%s\n", *<unspecified>*

79924 **STDERR**

79925 The standard error shall be used only for diagnostic messages.

79926 **OUTPUT FILES**

79927 None.

79928 **EXTENDED DESCRIPTION**

79929 None.

79930 **EXIT STATUS**79931 When the `-v` or `-V` options are specified, the following exit values shall be returned:

79932 0 Successful completion.

79933 >0 The *command_name* could not be found or an error occurred.

79934 Otherwise, the following exit values shall be returned:

79935 126 The utility specified by *command_name* was found but could not be invoked.79936 127 An error occurred in the *command* utility or the utility specified by *command_name* could not
79937 be found.79938 Otherwise, the exit status of *command* shall be that of the simple command specified by the
79939 arguments to *command*.79940 **CONSEQUENCES OF ERRORS**

79941 Default.

79942 **APPLICATION USAGE**79943 The order for command search allows functions to override regular built-ins and path searches.
79944 This utility is necessary to allow functions that have the same name as a utility to call the utility
79945 (instead of a recursive call to the function).79946 The system default path is available using *getconf*; however, since *getconf* may need to have the
79947 *PATH* set up before it can be called itself, the following can be used:79948 `command -p getconf PATH`79949 There are some advantages to suppressing the special characteristics of special built-ins on
79950 occasion. For example:79951 `command exec > unwritable-file`79952 does not cause a non-interactive script to abort, so that the output status can be checked by the
79953 script.79954 The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an
79955 error occurs so that applications can distinguish “failure to find a utility” from “invoked utility
79956 exited with an error indication”. The value 127 was chosen because it is not commonly used for
79957 other meanings; most utilities use small values for “normal error conditions” and the values
79958 above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen
79959 in a similar manner to indicate that the utility could be found, but not invoked. Some scripts
79960 produce meaningful error messages differentiating the 126 and 127 cases. The distinction
79961 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
79962 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
79963 any other reason.79964 Since the `-v` and `-V` options of *command* produce output in relation to the current shell execution
79965 environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell
79966 or separate utility execution environment, such as one of the following:79967 `(PATH=foo command -v)`79968 `nohup command -v`

79969
79970
79971

it does not necessarily produce correct results. For example, when called with *nohup* or an *exec* function, in a separate utility execution environment, most implementations are not able to identify aliases, functions, or special built-ins.

79972
79973
79974
79975
79976
79977
79978
79979
79980

Two types of regular built-ins could be encountered on a system and these are described separately by *command*. The description of command search in [Section 2.9.1.1](#) (on page 2264) allows for a standard utility to be implemented as a regular built-in as long as it is found in the appropriate place in a *PATH* search. So, for example, *command -v true* might yield */bin/true* or some similar pathname. Other implementation-defined utilities that are not defined by this volume of POSIX.1-200x might exist only as built-ins and have no pathname associated with them. These produce output identified as (regular) built-ins. Applications encountering these are not able to count on *execing* them, using them with *nohup*, overriding them with a different *PATH*, and so on.

79981

EXAMPLES

79982

1. Make a version of *cd* that always prints out the new working directory exactly once:

79983
79984
79985
79986

```
cd() {
    command cd "$@" >/dev/null
    pwd
}
```

79987

2. Start off a “secure shell script” in which the script avoids being spoofed by its parent:

79988
79989
79990
79991
79992
79993
79994
79995
79996
79997
79998
79999
80000

```
IFS='
# The preceding value should be <space><tab><newline>.
# Set IFS to its default value.

\unalias -a
# Unset all possible aliases.
# Note that unalias is escaped to prevent an alias
# being used for unalias.

unset -f command
# Ensure command is not a user function.

PATH="$(command -p getconf PATH):$PATH"
# Put on a reliable PATH prefix.

# ...
```

80001
80002
80003
80004
80005
80006
80007

At this point, given correct permissions on the directories called by *PATH*, the script has the ability to ensure that any utility it calls is the intended one. It is being very cautious because it assumes that implementation extensions may be present that would allow user functions to exist when it is invoked; this capability is not specified by this volume of POSIX.1-200x, but it is not prohibited as an extension. For example, the *ENV* variable precedes the invocation of the script with a user start-up script. Such a script could define functions to spoof the application.

80008

RATIONALE

80009

Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

80010
80011

There is nothing in the description of *command* that implies the command line is parsed any differently from that of any other simple command. For example:

80012

```
command a | b ; c
```

80013
80014

is not parsed in any special way that causes ' | ' or ' ; ' to be treated other than a pipe operator or semicolon or that prevents function lookup on **b** or **c**.

80015 The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since
 80016 *command* also goes to the file system to search for utilities, the name *builtin* would not be
 80017 intuitive.

80018 The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special
 80019 built-in for the following reasons:

- 80020 • The removal of exportable functions made the special precedence of a special built-in
 80021 unnecessary.
- 80022 • A special built-in has special properties (see [Section 2.14](#), on page 2280) that were
 80023 inappropriate for invoking other utilities. For example, two commands such as:

80024 `date > unwritable-file`

80025 `command date > unwritable-file`

80026 would have entirely different results; in a non-interactive script, the former would
 80027 continue to execute the next command, the latter would abort. Introducing this semantic
 80028 difference along with suppressing functions was seen to be non-intuitive.

80029 The `-p` option is present because it is useful to be able to ensure a safe path search that finds all
 80030 the standard utilities. This search might not be identical to the one that occurs through one of the
 80031 *exec* functions (as defined in the System Interfaces volume of POSIX.1-200x) when *PATH* is unset.
 80032 At the very least, this feature is required to allow the script to access the correct version of *getconf*
 80033 so that the value of the default path can be accurately retrieved.

80034 The *command* `-v` and `-V` options were added to satisfy requirements from users that are
 80035 currently accomplished by three different historical utilities: *type* in the System V shell, *whence* in
 80036 the KornShell, and *which* in the C shell. Since there is no historical agreement on how and what
 80037 to accomplish here, the POSIX *command* utility was enhanced and the historical utilities were left
 80038 unmodified. The C shell *which* merely conducts a path search. The KornShell *whence* is more
 80039 elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases,
 80040 exported aliases, and undefined functions.

80041 The output format of `-V` was left mostly unspecified because human users are its only audience.
 80042 Applications should not be written to care about this information; they can use the output of `-v`
 80043 to differentiate between various types of commands, but the additional information that may be
 80044 emitted by the more verbose `-V` is not needed and should not be arbitrarily constrained in its
 80045 verbosity or localization for application parsing reasons.

80046 FUTURE DIRECTIONS

80047 None.

80048 SEE ALSO

80049 [Section 2.9.1.1](#) (on page 2264), [Section 2.12](#) (on page 2277), [Section 2.14](#) (on page 2280), *sh*, *type*

80050 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

80051 XSH *exec*

80052 CHANGE HISTORY

80053 First released in Issue 4.

80054 Issue 7

80055 The *command* utility is moved from the User Portability Utilities option to the Base. User
 80056 Portability Utilities is now an option for interactive utilities.

80057 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80058 The APPLICATION USAGE and EXAMPLES are revised to replace the non-standard +
 80059 *getconf_CS_PATH* with *getconf PATH*.

80060 **NAME**80061 `compress` — compress data80062 **SYNOPSIS**80063 XSI `compress [-fv] [-b bits] [file...]`80064 `compress [-cfv] [-b bits] [file]`80065 **DESCRIPTION**80066 The *compress* utility shall attempt to reduce the size of the named files by using adaptive Lempel-
80067 Ziv coding algorithm.80068 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,
80069 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation.80070 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on
80071 December 10th, 1985, and assigned to Sperry Corporation.80072 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be
80073 changed and an error value greater than two shall be returned. Except when the output is to the
80074 standard output, each file shall be replaced by one with the extension *.Z*. If the invoking process
80075 has appropriate privileges, the ownership, modes, access time, and modification time of the
80076 original file are preserved. If appending the *.Z* to the filename would make the name exceed
80077 {NAME_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be
80078 compressed to the standard output.80079 **OPTIONS**80080 The *compress* utility shall conform to XBD [Section 12.2](#) (on page 201).

80081 The following options shall be supported:

80082 **-b *bits*** Specify the maximum number of bits to use in a code. For a conforming
80083 application, the *bits* argument shall be:80084 $9 \leq bits \leq 14$ 80085 The implementation may allow *bits* values of greater than 14. The default is 14, 15,
80086 or 16.80087 **-c** Cause *compress* to write to the standard output; the input file is not changed, and
80088 no *.Z* files are created.80089 **-f** Force compression of *file*, even if it does not actually reduce the size of the file, or if
80090 the corresponding *file.Z* file already exists. If the **-f** option is not given, and the
80091 process is not running in the background, the user is prompted as to whether an
80092 existing *file.Z* file should be overwritten.80093 **-v** Write the percentage reduction of each file to standard error.80094 **OPERANDS**

80095 The following operand shall be supported:

80096 *file* A pathname of a file to be compressed.80097 **STDIN**80098 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.

80099 **INPUT FILES**80100 If *file* operands are specified, the input files contain the data to be compressed.80101 **ENVIRONMENT VARIABLES**80102 The following environment variables shall affect the execution of *compress*:80103 *LANG* Provide a default value for the internationalization variables that are unset or null. |
80104 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
80105 variables used to determine the values of locale categories.)80106 *LC_ALL* If set to a non-empty string value, override the values of all the other
80107 internationalization variables.80108 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
80109 characters (for example, single-byte as opposed to multi-byte characters in
80110 arguments).80111 *LC_MESSAGES*80112 Determine the locale that should be used to affect the format and contents of
80113 diagnostic messages written to standard error.80114 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.80115 **ASYNCHRONOUS EVENTS**

80116 Default.

80117 **STDOUT**80118 If no *file* operands are specified, or if a *file* operand is '-', or if the *-c* option is specified, the
80119 standard output contains the compressed output.80120 **STDERR**80121 The standard error shall be used only for diagnostic and prompt messages and the output from
80122 *-v*.80123 **OUTPUT FILES**80124 The output files shall contain the compressed output. The format of compressed files is
80125 unspecified and interchange of such files between implementations (including access via
80126 unspecified file sharing mechanisms) is not required by POSIX.1-200x.80127 **EXTENDED DESCRIPTION**

80128 None.

80129 **EXIT STATUS**

80130 The following exit values shall be returned:

80131 0 Successful completion.

80132 1 An error occurred.

80133 2 One or more files were not compressed because they would have increased in size (and the
80134 *-f* option was not specified).

80135 >2 An error occurred.

80136 **CONSEQUENCES OF ERRORS**

80137 The input file shall remain unmodified.

80138 **APPLICATION USAGE**

80139 The amount of compression obtained depends on the size of the input, the number of *bits* per
 80140 code, and the distribution of common substrings. Typically, text such as source code or English is
 80141 reduced by 50-60%. Compression is generally much better than that achieved by Huffman
 80142 coding or adaptive Huffman coding (*compact*), and takes less time to compute.

80143 Although *compress* strictly follows the default actions upon receipt of a signal or when an error
 80144 occurs, some unexpected results may occur. In some implementations it is likely that a partially
 80145 compressed file is left in place, alongside its uncompressed input file. Since the general
 80146 operation of *compress* is to delete the uncompressed file only after the *.Z* file has been
 80147 successfully filled, an application should always carefully check the exit status of *compress* before
 80148 arbitrarily deleting files that have like-named neighbors with *.Z* suffixes.

80149 The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the
 80150 restrictions imposed by the lack of an explicit published file format). Some implementations
 80151 based on 16-bit architectures cannot support 15 or 16-bit uncompression.

80152 **EXAMPLES**

80153 None.

80154 **RATIONALE**

80155 None.

80156 **FUTURE DIRECTIONS**

80157 None.

80158 **SEE ALSO**

80159 *uncompress*, *zcat*

80160 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

80161 **CHANGE HISTORY**

80162 First released in Issue 4.

80163 **Issue 6**

80164 The normative text is reworded to avoid use of the term “must” for application requirements.

80165 An error case is added for systems not supporting adaptive Lempel-Ziv coding.

80166 **Issue 7**

80167 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80168
80169
80170
80171
80172
80173
80174
80175
80176
80177
80178
80179
80180
80181
80182
80183
80184
80185
80186
80187
80188
80189
80190
80191
80192
80193
80194
80195
80196
80197
80198
80199
80200
80201
80202
80203
80204
80205
80206
80207
80208
80209
80210

NAME

cp — copy files

SYNOPSIS

```
cp [-Pfp] source_file target_file
cp [-Pfp] source_file... target
cp -R [-H|-L|-P] [-fip] source_file... target
```

DESCRIPTION

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The *cp* utility shall copy the contents of *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by *target_file*.

The second synopsis form is denoted by two or more operands where the **-R** option is not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* does not name a directory. The *cp* utility shall copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents of the file referenced by *source_file*) to the destination path named by the concatenation of *target*, a slash character, and the last component of *source_file*.

The third and fourth synopsis forms are denoted by two or more operands where the **-R** option is specified. The *cp* utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows:

- If *target* exists and names an existing directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the pathname of the file relative to the directory containing *source_file*.
- If *target* does not exist and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target*; the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target*, a slash character, and the pathname of the file relative to *source_file*.

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and does not name a directory.

In the following description, the term *dest_file* refers to the file named by the destination path. The term *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

- If the **-R** option was not specified, *cp* shall take actions based on the type and contents of the file referenced by the symbolic link, and not by the symbolic link itself, unless the **-P** option was specified.
- If the **-R** option was specified:
 - If none of the options **-H**, **-L**, nor **-P** were specified, it is unspecified which of **-H**, **-L**, or **-P** will be used as a default.
 - If the **-H** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.
 - If the **-L** option was specified, *cp* shall take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

80211 — If the `-P` option was specified, *cp* shall copy any symbolic link specified as a
 80212 *source_file* operand and any symbolic links encountered during traversal of a file
 80213 hierarchy, and shall not follow any symbolic links.

80214 For each *source_file*, the following steps shall be taken:

- 80215 1. If *source_file* references the same file as *dest_file*, *cp* may write a diagnostic message to
 80216 standard error; it shall do nothing more with *source_file* and shall go on to any remaining
 80217 files.
- 80218 2. If *source_file* is of type directory, the following steps shall be taken:
 - 80219 a. If the `-R` option was not specified, *cp* shall write a diagnostic message to standard
 80220 error, do nothing more with *source_file*, and go on to any remaining files.
 - 80221 b. If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, *cp*
 80222 shall do nothing more with *source_file* and go on to any remaining files.
 - 80223 c. If *dest_file* exists and it is a file type not specified by the System Interfaces volume
 80224 of POSIX.1-200x, the behavior is implementation-defined.
 - 80225 d. If *dest_file* exists and it is not of type directory, *cp* shall write a diagnostic message
 80226 to standard error, do nothing more with *source_file* or any files below *source_file* in
 80227 the file hierarchy, and go on to any remaining files.
 - 80228 e. If the directory *dest_file* does not exist, it shall be created with file permission bits
 80229 set to the same value as those of *source_file*, modified by the file creation mask of
 80230 the user if the `-p` option was not specified, and then bitwise-inclusively OR'ed
 80231 with `S_IRWXU`. If *dest_file* cannot be created, *cp* shall write a diagnostic message to
 80232 standard error, do nothing more with *source_file*, and go on to any remaining files.
 80233 It is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source_file*.
 - 80234 f. The files in the directory *source_file* shall be copied to the directory *dest_file*, taking
 80235 the four steps (1 to 4) listed here with the files as *source_files*.
 - 80236 g. If *dest_file* was created, its file permission bits shall be changed (if necessary) to be
 80237 the same as those of *source_file*, modified by the file creation mask of the user if the
 80238 `-p` option was not specified.
 - 80239 h. The *cp* utility shall do nothing more with *source_file* and go on to any remaining
 80240 files.
- 80241 3. If *source_file* is of type regular file, the following steps shall be taken:
 - 80242 a. If *dest_file* exists, the following steps shall be taken:
 - 80243 i. If the `-i` option is in effect, the *cp* utility shall write a prompt to the standard
 80244 error and read a line from the standard input. If the response is not
 80245 affirmative, *cp* shall do nothing more with *source_file* and go on to any
 80246 remaining files.
 - 80247 ii. A file descriptor for *dest_file* shall be obtained by performing actions
 80248 equivalent to the `open()` function defined in the System Interfaces volume of
 80249 POSIX.1-200x called using *dest_file* as the *path* argument, and the bitwise-
 80250 inclusive OR of `O_WRONLY` and `O_TRUNC` as the *oflag* argument.
 - 80251 iii. If the attempt to obtain a file descriptor fails and the `-f` option is in effect, *cp*
 80252 shall attempt to remove the file by performing actions equivalent to the
 80253 `unlink()` function defined in the System Interfaces volume of POSIX.1-200x
 80254 called using *dest_file* as the *path* argument. If this attempt succeeds, *cp* shall
 80255 continue with step 3b.

- 80256 b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions
 80257 equivalent to the *open()* function defined in the System Interfaces volume of
 80258 POSIX.1-200x called using *dest_file* as the *path* argument, and the bitwise-inclusive
 80259 OR of *O_WRONLY* and *O_CREAT* as the *oflag* argument. The file permission bits
 80260 of *source_file* shall be the *mode* argument.
- 80261 c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to
 80262 standard error, do nothing more with *source_file*, and go on to any remaining files.
- 80263 d. The contents of *source_file* shall be written to the file descriptor. Any write errors
 80264 shall cause *cp* to write a diagnostic message to standard error and continue to step
 80265 3e.
- 80266 e. The file descriptor shall be closed.
- 80267 f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in
 80268 step 3d, it is unspecified if *cp* continues with any remaining files. If no write error
 80269 occurred in step 3d, *cp* shall go on to any remaining files.
- 80270 4. Otherwise, the following steps shall be taken:
- 80271 a. If the **-R** option was specified, the following steps shall be taken:
- 80272 i. The *dest_file* shall be created with the same file type as *source_file*.
- 80273 ii. If *source_file* is a file of type FIFO, the file permission bits shall be the same
 80274 as those of *source_file*, modified by the file creation mask of the user if the **-p**
 80275 option was not specified. Otherwise, the permissions, owner ID, and group
 80276 ID of *dest_file* are implementation-defined.
- 80277 If this creation fails for any reason, *cp* shall write a diagnostic message to
 80278 standard error, do nothing more with *source_file*, and go on to any
 80279 remaining files.
- 80280 iii. If *source_file* is a file of type symbolic link, and the options require the
 80281 symbolic link itself to be acted upon, the pathname contained in *dest_file*
 80282 shall be the same as the pathname contained in *source_file*.
- 80283 If this fails for any reason, *cp* shall write a diagnostic message to standard
 80284 error, do nothing more with *source_file*, and go on to any remaining files.

80285 If the implementation provides additional or alternate access control mechanisms (see XBD
 80286 Section 4.4, on page 96), their effect on copies of files is implementation-defined.

80287 OPTIONS

80288 The *cp* utility shall conform to XBD Section 12.2 (on page 201).

80289 The following options shall be supported:

- 80290 **-f** If a file descriptor for a destination file cannot be obtained, as described in step
 80291 3.a.ii., attempt to unlink the destination file and proceed.
- 80292 **-H** Take actions based on the type and contents of the file referenced by any symbolic
 80293 link specified as a *source_file* operand.
- 80294 **-i** Write a prompt to standard error before copying to any existing non-directory
 80295 destination file. If the response from the standard input is affirmative, the copy
 80296 shall be attempted; otherwise, it shall not.
- 80297 **-L** Take actions based on the type and contents of the file referenced by any symbolic
 80298 link specified as a *source_file* operand or any symbolic links encountered during
 80299 traversal of a file hierarchy.

- 80300 **-P** Take actions on any symbolic link specified as a *source_file* operand or any
80301 symbolic link encountered during traversal of a file hierarchy.
- 80302 **-p** Duplicate the following characteristics of each source file in the corresponding
80303 destination file:
- 80304 1. The time of last data modification and time of last access. If this duplication
80305 fails for any reason, *cp* shall write a diagnostic message to standard error.
 - 80306 2. The user ID and group ID. If this duplication fails for any reason, it is
80307 unspecified whether *cp* writes a diagnostic message to standard error.
 - 80308 3. The file permission bits and the S_ISUID and S_ISGID bits. Other,
80309 implementation-defined, bits may be duplicated as well. If this duplication
80310 fails for any reason, *cp* shall write a diagnostic message to standard error.
- 80311 If the user ID or the group ID cannot be duplicated, the file permission bits
80312 S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but
80313 are not duplicated in the destination file, it is unspecified whether *cp* writes a
80314 diagnostic message to standard error.
- 80315 The order in which the preceding characteristics are duplicated is unspecified. The
80316 *dest_file* shall not be deleted if these characteristics cannot be preserved.
- 80317 **-R** Copy file hierarchies.
- 80318 Specifying more than one of the mutually-exclusive options **-H**, **-L**, and **-P** shall not be
80319 considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

80320 The following operands shall be supported:

- 80321 *source_file* A pathname of a file to be copied.
- 80322 *target_file* A pathname of an existing or nonexistent file, used for the output when a single
80323 file is copied.
- 80324 *target* A pathname of a directory to contain the copied files.

STDIN

80325 The standard input shall be used to read an input line in response to each prompt specified in
80326 the STDERR section. Otherwise, the standard input shall not be used.

INPUT FILES

80327 The input files specified as operands may be of any file type.

ENVIRONMENT VARIABLES

80328 The following environment variables shall affect the execution of *cp*:

- 80329 **LANG** Provide a default value for the internationalization variables that are unset or null. |
80330 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
80331 variables used to determine the values of locale categories.)
- 80332 **LC_ALL** If set to a non-empty string value, override the values of all the other
80333 internationalization variables.
- 80334 **LC_COLLATE** Determine the locale for the behavior of ranges, equivalence classes, and multi-
80335 character collating elements used in the extended regular expression defined for
80336 the **yesexpr** locale keyword in the **LC_MESSAGES** category.
- 80337 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
80338 characters (for example, single-byte as opposed to multi-byte characters in
80339 arguments and input files) and the behavior of character classes used in the
80340 |
80341 |
80342 |
80343 |
80344 |

- 80345 extended regular expression defined for the **yesexpr** locale keyword in the
80346 *LC_MESSAGES* category.
- 80347 *LC_MESSAGES*
80348 Determine the locale for the processing of affirmative responses that should be
80349 used to affect the format and contents of diagnostic messages written to standard
80350 error.
- 80351 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 80352 **ASYNCHRONOUS EVENTS**
80353 Default.
- 80354 **STDOUT**
80355 Not used.
- 80356 **STDERR**
80357 A prompt shall be written to standard error under the conditions specified in the *DESCRIPTION*
80358 section. The prompt shall contain the destination pathname, but its format is otherwise
80359 unspecified. Otherwise, the standard error shall be used only for diagnostic messages.
- 80360 **OUTPUT FILES**
80361 The output files may be of any type.
- 80362 **EXTENDED DESCRIPTION**
80363 None.
- 80364 **EXIT STATUS**
80365 The following exit values shall be returned:
80366 0 All files were copied successfully.
80367 >0 An error occurred.
- 80368 **CONSEQUENCES OF ERRORS**
80369 If *cp* is prematurely terminated by a signal or error, files or file hierarchies may be only partially
80370 copied and files and directories may have incorrect permissions or access and modification
80371 times.
- 80372 **APPLICATION USAGE**
80373 The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to
80374 prevent users from creating programs that are set-user-ID or set-group-ID to them when copying
80375 files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example,
80376 if a file is set-user-ID and the copy has a different group ID than the source, a new group of users
80377 has execute permission to a set-user-ID program than did previously. In particular, this is a
80378 problem for superusers copying users' trees.
- 80379 **EXAMPLES**
80380 None.
- 80381 **RATIONALE**
80382 The **-i** option exists on BSD systems, giving applications and users a way to avoid accidentally
80383 removing files when copying. Although the 4.3 BSD version does not prompt if the standard
80384 input is not a terminal, the standard developers decided that use of **-i** is a request for
80385 interaction, so when the destination path exists, the utility takes instructions from whatever
80386 responds on standard input.

80387 The exact format of the interactive prompts is unspecified. Only the general nature of the
80388 contents of prompts are specified because implementations may desire more descriptive
80389 prompts than those used on historical implementations. Therefore, an application using the **-i**
80390 option relies on the system to provide the most suitable dialog directly with the user, based on
80391 the behavior specified.

80392 The `-p` option is historical practice on BSD systems, duplicating the time of last data
 80393 modification and time of last access. This volume of POSIX.1-200x extends it to preserve the user
 80394 and group IDs, as well as the file permissions. This requirement has obvious problems in that
 80395 the directories are almost certainly modified after being copied. This volume of POSIX.1-200x
 80396 requires that the modification times be preserved. The statement that the order in which the
 80397 characteristics are duplicated is unspecified is to permit implementations to provide the
 80398 maximum amount of security for the user. Implementations should take into account the
 80399 obvious security issues involved in setting the owner, group, and mode in the wrong order or
 80400 creating files with an owner, group, or mode different from the final value.

80401 It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be
 80402 set due to the widespread practice of users using `-p` to duplicate some portion of the file
 80403 characteristics, indifferent to the duplication of others. Historic implementations only write
 80404 diagnostic messages on errors other than [E`PERM`].

80405 Earlier versions of this standard included support for the `-r` option to copy file hierarchies. The
 80406 `-r` option is historical practice on BSD and BSD-derived systems. This option is no longer
 80407 specified by POSIX.1-200x but may be present in some implementations. The `-R` option was
 80408 added as a close synonym to the `-r` option, selected for consistency with all other options in this
 80409 volume of POSIX.1-200x that do recursive directory descent.

80410 The difference between `-R` and the removed `-r` option is in the treatment by `cp` of file types other
 80411 than regular and directory. It was implementation-defined how the `-` option treated special files
 80412 to allow both historical implementations and those that chose to support `-r` with the same
 80413 abilities as `-R` defined by this volume of POSIX.1-200x. The original `-r` flag, for historic reasons,
 80414 did not handle special files any differently from regular files, but always read the file and copied
 80415 its contents. This had obvious problems in the presence of special file types; for example,
 80416 character devices, FIFOs, and sockets.

80417 When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy
 80418 files that are on the same level in the hierarchy or above the file where the failure occurred. It is
 80419 unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which
 80420 cannot succeed in any case).

80421 Permissions, owners, and groups of created special file types have been deliberately left as
 80422 implementation-defined. This is to allow systems to satisfy special requirements (for example,
 80423 allowing users to create character special devices, but requiring them to be owned by a certain
 80424 group). In general, it is strongly suggested that the permissions, owner, and group be the same
 80425 as if the user had run the historical `mknod`, `ln`, or other utility to create the file. It is also probable
 80426 that additional privileges are required to create block, character, or other implementation-
 80427 defined special file types.

80428 Additionally, the `-p` option explicitly requires that all set-user-ID and set-group-ID permissions
 80429 be discarded if any of the owner or group IDs cannot be set. This is to keep users from
 80430 unintentionally giving away special privilege when copying programs.

80431 When creating regular files, historical versions of `cp` use the mode of the source file as modified
 80432 by the file mode creation mask. Other choices would have been to use the mode of the source file
 80433 unmodified by the creation mask or to use the same mode as would be given to a new file
 80434 created by the user (plus the execution bits of the source file) and then modify it by the file mode
 80435 creation mask. In the absence of any strong reason to change historic practice, it was in large part
 80436 retained.

80437 When creating directories, historical versions of `cp` use the mode of the source directory, plus
 80438 read, write, and search bits for the owner, as modified by the file mode creation mask. This is
 80439 done so that `cp` can copy trees where the user has read permission, but the owner does not. A
 80440 side effect is that if the file creation mask denies the owner permissions, `cp` fails. Also, once the
 80441 copy is done, historical versions of `cp` set the permissions on the created directory to be the same

80442 as the source directory, unmodified by the file creation mask.

80443 This behavior has been modified so that *cp* is always able to create the contents of the directory,
80444 regardless of the file creation mask. After the copy is done, the permissions are set to be the same
80445 as the source directory, as modified by the file creation mask. This latter change from historical
80446 behavior is to prevent users from accidentally creating directories with permissions beyond
80447 those they would normally set and for consistency with the behavior of *cp* in creating files.

80448 It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations
80449 are strongly encouraged to do so. Historical implementations have detected the attempt in most
80450 cases.

80451 There are two methods of copying subtrees in this volume of POSIX.1-200x. The other method is
80452 described as part of the *pax* utility (see *pax*). Both methods are historical practice. The *cp* utility
80453 provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each
80454 provides additional functionality to the other; in particular, *pax* maintains the hard-link structure
80455 of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results
80456 be similar (using appropriate option combinations in both utilities). The results are not required
80457 to be identical; there seemed insufficient gain to applications to balance the difficulty of
80458 implementations having to guarantee that the results would be exactly identical.

80459 The wording allowing *cp* to copy a directory to implementation-defined file types not specified
80460 by the System Interfaces volume of POSIX.1-200x is provided so that implementations
80461 supporting symbolic links are not required to prohibit copying directories to symbolic links.
80462 Other extensions to the System Interfaces volume of POSIX.1-200x file types may need to use
80463 this loophole as well.

80464 FUTURE DIRECTIONS

80465 None.

80466 SEE ALSO

80467 *mv*, *find*, *ln*, *pax*

80468 XBD Section 4.4 (on page 96), Chapter 8 (on page 159), Section 12.2 (on page 201)

80469 XSH *open()*, *unlink*

80470 CHANGE HISTORY

80471 First released in Issue 2.

80472 Issue 6

80473 The **-r** option is marked obsolescent.

80474 The new options **-H**, **-L**, and **-P** are added to align with the IEEE P1003.2b draft standard. These
80475 options affect the processing of symbolic links.

80476 IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the **-P** option.

80477 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/18 is applied, correcting an error in the
80478 SEE ALSO section.

80479 Issue 7

80480 SD5-XCU-ERN-31 and SD5-XCU-ERN-42 are applied, updating the DESCRIPTION.

80481 The obsolescent **-r** option is removed.

80482 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80483 SD5-XCU-ERN-102 is applied, clarifying the **-i** option within the OPTIONS section.

80484 The **-P** option is added to the SYNOPSIS and to the DESCRIPTION with respect to the **-R** +
80485 option.

80486 **NAME**
 80487 `crontab` — schedule periodic background work

80488 **SYNOPSIS**
 80489 `crontab` [*file*]

80490 UP `crontab` [**-e** | **-l** | **-r**]

80491 DESCRIPTION

80492 UP The *crontab* utility shall create, replace, or edit a user's crontab entry; a crontab entry is a list of
 80493 commands and the times at which they shall be executed. The new crontab entry can be input by
 80494 specifying *file* or input from standard input if no *file* operand is specified, or by using an editor,
 80495 if **-e** is specified.

80496 Upon execution of a command from a crontab entry, the implementation shall supply a default
 80497 environment, defining at least the following environment variables:

80498 *HOME* A pathname of the user's home directory.

80499 *LOGNAME* The user's login name.

80500 *PATH* A string representing a search path guaranteed to find all of the standard utilities.

80501 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by
 80502 this volume of POSIX.1-200x, the value shall be a pathname for *sh*.

80503 The values of these variables when *crontab* is invoked as specified by this volume of
 80504 POSIX.1-200x shall not affect the default values provided when the scheduled command is run.

80505 If standard output and standard error are not redirected by commands executed from the
 80506 crontab entry, any generated output or errors shall be mailed, via an implementation-defined
 80507 method, to the user.

80508 XSI Users shall be permitted to use *crontab* if their names appear in the file **cron.allow** which is
 80509 located in an implementation-defined directory. If that file does not exist, the file **cron.deny**,
 80510 which is located in an implementation-defined directory, shall be checked to determine whether
 80511 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate
 80512 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage
 80513 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line.

80514 OPTIONS

80515 The *crontab* utility shall conform to XBD [Section 12.2](#) (on page 201).

80516 The following options shall be supported:

80517 UP **-e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if
 80518 the crontab entry does not exist. When editing is complete, the entry shall be
 80519 installed as the user's crontab entry.

80520 **-l** (The letter ell.) List the invoking user's crontab entry.

80521 **-r** Remove the invoking user's crontab entry.

80522 OPERANDS

80523 The following operand shall be supported:

80524 *file* The pathname of a file that contains specifications, in the format defined in the
 80525 INPUT FILES section, for crontab entries.

80526 **STDIN**

80527 See the INPUT FILES section.

80528 **INPUT FILES**80529 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
80530 consisting of lines of six fields each. The fields shall be separated by <blank>s. The first five
80531 fields shall be integer patterns that specify the following:

- 80532 1. Minute [0,59]
- 80533 2. Hour [0,23]
- 80534 3. Day of the month [1,31]
- 80535 4. Month of the year [1,12]
- 80536 5. Day of the week ([0,6] with 0=Sunday)

80537 Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of
80538 elements separated by commas. An element shall be either a number or two numbers separated
80539 by a hyphen (meaning an inclusive range). The specification of days can be made by two fields
80540 (day of the month and day of the week). If month, day of month, and day of week are all
80541 asterisks, every day shall be matched. If either the month or day of month is specified as an
80542 element or list, but the day of week is an asterisk, the month and day of month fields shall
80543 specify the days that match. If both month and day of month are specified as an asterisk, but day
80544 of week is an element or list, then only the specified days of the week match. Finally, if either the
80545 month or day of month is specified as an element or list, and the day of week is also specified as
80546 an element or list, then any day matching either the month and day of month, or the day of
80547 week, shall be matched.

80548 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
80549 times. A percent sign character in this field shall be translated to a <newline>. Any character
80550 preceded by a backslash (including the '%') shall cause that character to be treated literally.
80551 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the
80552 command interpreter. The other lines shall be made available to the command as standard input.

80553 Blank lines and those whose first non-<blank> is '#' shall be ignored.

80554 XSI The text files **cron.allow** and **cron.deny**, which are located in an implementation-defined
80555 directory, shall contain zero or more user names, one per line, of users who are, respectively,
80556 authorized or denied access to the service underlying the *crontab* utility.

80557 **ENVIRONMENT VARIABLES**80558 The following environment variables shall affect the execution of *crontab*:

80559 **EDITOR** Determine the editor to be invoked when the **-e** option is specified. The default
80560 editor shall be *vi*.

80561 **LANG** Provide a default value for the internationalization variables that are unset or null. |
80562 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
80563 variables used to determine the values of locale categories.)

80564 **LC_ALL** If set to a non-empty string value, override the values of all the other
80565 internationalization variables.

80566 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
80567 characters (for example, single-byte as opposed to multi-byte characters in
80568 arguments and input files).

80569 **LC_MESSAGES**

80570 Determine the locale that should be used to affect the format and contents of
80571 diagnostic messages written to standard error.

80572 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

80573 **ASYNCHRONOUS EVENTS**

80574 Default.

80575 **STDOUT**

80576 If the `-l` option is specified, the crontab entry shall be written to the standard output.

80577 **STDERR**

80578 The standard error shall be used only for diagnostic messages.

80579 **OUTPUT FILES**

80580 None.

80581 **EXTENDED DESCRIPTION**

80582 None.

80583 **EXIT STATUS**

80584 The following exit values shall be returned:

80585 0 Successful completion.

80586 >0 An error occurred.

80587 **CONSEQUENCES OF ERRORS**

80588 UP The user's crontab entry is not submitted, removed, **edited**, or listed.

80589 **APPLICATION USAGE**

80590 The format of the crontab entry shown here is guaranteed only for the POSIX locale. Other
80591 cultures may be supported with substantially different interfaces, although implementations are
80592 encouraged to provide comparable levels of functionality.

80593 The default settings of the *HOME*, *LOGNAME*, *PATH*, and *SHELL* variables that are given to the
80594 scheduled job are not affected by the settings of those variables when *crontab* is run; as stated,
80595 they are defaults. The text about "invoked as specified by this volume of POSIX.1-200x" means
80596 that the implementation may provide extensions that allow these variables to be affected at
80597 runtime, but that the user has to take explicit action in order to access the extension, such as give
80598 a new option flag or modify the format of the crontab entry.

80599 A typical user error is to type only *crontab*; this causes the system to wait for the new crontab
80600 entry on standard input. If end-of-file is typed (generally <control>-D), the crontab entry is
80601 replaced by an empty file. In this case, the user should type the interrupt character, which
80602 prevents the crontab entry from being replaced.

80603 **EXAMPLES**

80604 1. Clean up **core** files every weekday morning at 3:15 am:

80605 `15 3 * * 1-5 find "$HOME" -name core -exec rm -f {} + 2>/dev/null` |

80606 2. Mail a birthday greeting:

80607 `0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.`

80608 3. As an example of specifying the two types of days:

80609 `0 0 1,15 * 1`

80610 would run a command on the first and fifteenth of each month, as well as on every
80611 Monday. To specify days by only one field, the other field should be set to `'*'`; for
80612 example:

80613 `0 0 * * 1`

80614 would run a command only on Mondays.

80615 **RATIONALE**

80616 All references to a *cron* daemon and to *cron files* have been omitted. Although historical
80617 implementations have used this arrangement, there is no reason to limit future implementations.

80618 This description of *crontab* is designed to support only users with normal privileges. The format
80619 of the input is based on the System V *crontab*; however, there is no requirement here that the
80620 actual system database used by the *cron* daemon (or a similar mechanism) use this format
80621 internally. For example, systems derived from BSD are likely to have an additional field
80622 appended that indicates the user identity to be used when the job is submitted.

80623 The `-e` option was adopted from the SVID as a user convenience, although it does not exist in all
80624 historical implementations.

80625 **FUTURE DIRECTIONS**

80626 None.

80627 **SEE ALSO**

80628 *at*

80629 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

80630 **CHANGE HISTORY**

80631 First released in Issue 2.

80632 **Issue 6**

80633 This utility is marked as part of the User Portability Utilities option.

80634 The normative text is reworded to avoid use of the term “must” for application requirements.

80635 **Issue 7**

80636 The *crontab* utility (except for the `-e` option) is moved from the User Portability Utilities option
80637 to the Base. User Portability Utilities is now an option for interactive utilities.

80638 SD5-XCU-ERN-95 is applied, removing the references to fixed locations for the files referenced
80639 by the *crontab* utility.

80640 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80641 The first example is changed to remove the unreliable use of `find | xargs`. +

80642 **NAME**

80643 csplit — split files based on context

80644 **SYNOPSIS**80645 csplit [-ks] [-f *prefix*] [-n *number*] *file arg...*80646 **DESCRIPTION**80647 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into
80648 other files as directed by the *arg* operands, and write the sizes of the files.80649 **OPTIONS**80650 The *csplit* utility shall conform to XBD Section 12.2 (on page 201).

80651 The following options shall be supported:

80652 **-f *prefix*** Name the created files *prefix00*, *prefix01*, ..., *prefixn*. The default is *xx00* ... *xxn*. If
80653 the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an
80654 error shall result, *csplit* shall exit with a diagnostic message, and no files shall be
80655 created.80656 **-k** Leave previously created files intact. By default, *csplit* shall remove created files if
80657 an error occurs.80658 **-n *number*** Use *number* decimal digits to form filenames for the file pieces. The default shall be
80659 2.80660 **-s** Suppress the output of file size messages.80661 **OPERANDS**

80662 The following operands shall be supported:

80663 ***file*** The pathname of a text file to be split. If *file* is '-', the standard input shall be
80664 used.80665 Each *arg* operand can be one of the following:80666 ***/rexp/[offset]***
80667 A file shall be created using the content of the lines from the current line up to, but
80668 not including, the line that results from the evaluation of the regular expression
80669 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for
80670 basic regular expressions described in XBD Section 9.3 (on page 169). The
80671 application shall use the sequence "\/" to specify a slash character within the *rexp*.
80672 The optional offset shall be a positive or negative integer value representing a
80673 number of lines. A positive integer value can be preceded by '+'. If the selection
80674 of lines from an *offset* expression of this type would create a file with zero lines, or
80675 one with greater than the number of lines left in the input file, the results are
80676 unspecified. After the section is created, the current line shall be set to the line that
80677 results from the evaluation of the regular expression with any offset applied. If the
80678 current line is the first line in the file and a regular expression operation has not yet
80679 been performed, the pattern match of *rexp* shall be applied from the current line to
80680 the end of the file. Otherwise, the pattern match of *rexp* shall be applied from the
80681 line following the current line to the end of the file.80682 ***%rexp%[offset]***80683 Equivalent to */rexp/[offset]*, except that no file shall be created for the selected
80684 section of the input file. The application shall use the sequence "%%" to specify a
80685 percent-sign character within the *rexp*.

80686 *line_no* Create a file from the current line up to (but not including) the line number *line_no*.
 80687 Lines in the file shall be numbered starting at one. The current line becomes
 80688 *line_no*.

80689 {*num*} Repeat operand. This operand can follow any of the operands described
 80690 previously. If it follows a *rexp* type operand, that operand shall be applied *num*
 80691 more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines,
 80692 *num* times, from that point.

80693 An error shall be reported if an operand does not reference a line between the current position
 80694 and the end of the file.

STDIN

80695 See the INPUT FILES section.

INPUT FILES

80697 The input file shall be a text file.

ENVIRONMENT VARIABLES

80700 The following environment variables shall affect the execution of *csplit*:

80701 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 80702 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 80703 variables used to determine the values of locale categories.)

80704 *LC_ALL* If set to a non-empty string value, override the values of all the other
 80705 internationalization variables.

80706 *LC_COLLATE*

80707 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 80708 character collating elements within regular expressions.

80709 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 80710 characters (for example, single-byte as opposed to multi-byte characters in
 80711 arguments and input files) and the behavior of character classes within regular
 80712 expressions.

80713 *LC_MESSAGES*

80714 Determine the locale that should be used to affect the format and contents of
 80715 diagnostic messages written to standard error.

80716 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

80717 If the *-k* option is specified, created files shall be retained. Otherwise, the default action occurs.

STDOUT

80720 Unless the *-s* option is used, the standard output shall consist of one line per file created, with a
 80721 format as follows:

80722 "%d\n", <file size in bytes>

STDERR

80724 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

80726 The output files shall contain portions of the original input file; otherwise, unchanged.

EXTENDED DESCRIPTION

80728 None.

80729 **EXIT STATUS**

80730 The following exit values shall be returned:

80731 0 Successful completion.

80732 >0 An error occurred.

80733 **CONSEQUENCES OF ERRORS**80734 By default, created files shall be removed if an error occurs. When the `-k` option is specified,
80735 created files shall not be removed if an error occurs.80736 **APPLICATION USAGE**

80737 None.

80738 **EXAMPLES**80739 1. This example creates four files, **cobol00** ... **cobol03**:80740 `csplit -f cobol file '/procedure division/' /par5./ /par16./`

80741 After editing the split files, they can be recombined as follows:

80742 `cat cobol0[0-3] > file`

80743 Note that this example overwrites the original file.

80744 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up
80745 to 9999 lines; this is because lines in the file are numbered from 1 rather than zero, for
80746 historical reasons:80747 `csplit -k file 100 {99}`80748 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with
80749 a `'}'` at the beginning of the line, this example creates a file containing each separate C
80750 routine (up to 21) in **prog.c**:80751 `csplit -k prog.c '%main(%' '/^}'+1' {20}`80752 **RATIONALE**80753 The `-n` option was added to extend the range of filenames that could be handled.80754 Consideration was given to adding a `-a` flag to use the alphabetic filename generation used by
80755 the historical *split* utility, but the functionality added by the `-n` option was deemed to make
80756 alphabetic naming unnecessary.80757 **FUTURE DIRECTIONS**

80758 None.

80759 **SEE ALSO**80760 *sed*, *split*80761 XBD [Chapter 8](#) (on page 159), [Section 9.3](#) (on page 169), [Section 12.2](#) (on page 201) +80762 **CHANGE HISTORY**

80763 First released in Issue 2.

80764 **Issue 5**

80765 The FUTURE DIRECTIONS section is added.

80766 **Issue 6**

80767 This utility is marked as part of the User Portability Utilities option.

80768 The APPLICATION USAGE section is added.

80769 The description of regular expression operands is changed to align with the IEEE P1003.2b draft
80770 standard.

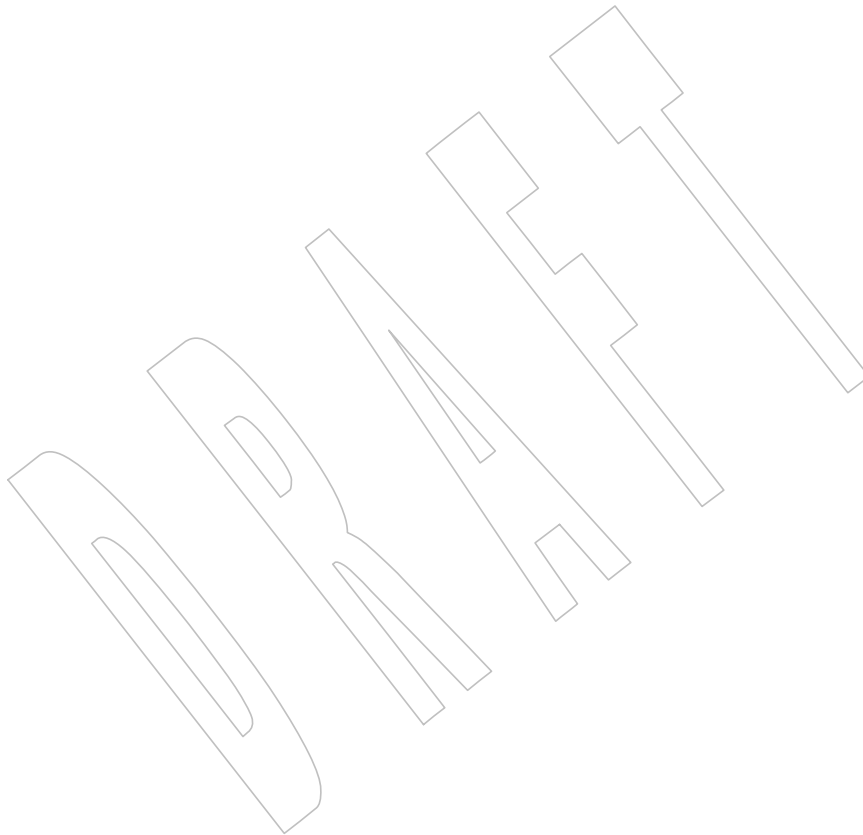
80771 The normative text is reworded to avoid use of the term “must” for application requirements.

80772 **Issue 7**

80773 The *csplit* utility is moved from the User Portability Utilities option to the Base. User Portability
80774 Utilities is now an option for interactive utilities.

80775 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80776 The SYNOPSIS and OPERANDS sections are revised to use a single *arg* to split a file into two +
80777 pieces.



80778 **NAME**
 80779 ctags — create a tags file (**DEVELOPMENT, FORTRAN**)

80780 **SYNOPSIS**
 80781 SD ctags [-a] [-f *tagsfile*] *pathname...*
 80782 ctags -x *pathname...*

80783 **DESCRIPTION**
 80784 The *ctags* utility shall be provided on systems that support the the Software Development
 80785 Utilities option, and either or both of the C-Language Development Utilities option and
 80786 FORTRAN Development Utilities option. On other systems, it is optional.

80787 The *ctags* utility shall write a *tagsfile* or an index of objects from C-language or FORTRAN source
 80788 files specified by the *pathname* operands. The *tagsfile* shall list the locators of language-specific
 80789 objects within the source files. A locator consists of a name, pathname, and either a search
 80790 pattern or a line number that can be used in searching for the object definition. The objects that
 80791 shall be recognized are specified in the EXTENDED DESCRIPTION section.

80792 **OPTIONS**
 80793 The *ctags* utility shall conform to XBD [Section 12.2](#) (on page 201).

80794 The following options shall be supported:

80795 **-a** Append to *tagsfile*.
 80796 **-f *tagsfile*** Write the object locator lists into *tagsfile* instead of the default file named **tags** in the
 80797 current directory.
 80798 **-x** Produce a list of object names, the line number, and filename in which each is
 80799 defined, as well as the text of that line, and write this to the standard output. A
 80800 *tagsfile* shall not be created when **-x** is specified.

80801 **OPERANDS**
 80802 The following *pathname* operands are supported:

80803 ***file.c*** Files with basenames ending with the **.c** suffix shall be treated as C-language
 80804 source code. Such files that are not valid input to *c99* produce unspecified results.
 80805 ***file.h*** Files with basenames ending with the **.h** suffix shall be treated as C-language
 80806 source code. Such files that are not valid input to *c99* produce unspecified results.
 80807 ***file.f*** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-
 80808 language source code. Such files that are not valid input to *fort77* produce
 80809 unspecified results.

80810 The handling of other files is implementation-defined.

80811 **STDIN**
 80812 See the INPUT FILES section.

80813 **INPUT FILES**
 80814 The input files shall be text files containing source code in the language indicated by the
 80815 operand filename suffixes.

80816 **ENVIRONMENT VARIABLES**80817 The following environment variables shall affect the execution of *ctags*:80818 *LANG* Provide a default value for the internationalization variables that are unset or null. |
80819 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
80820 variables used to determine the values of locale categories.)80821 *LC_ALL* If set to a non-empty string value, override the values of all the other
80822 internationalization variables.80823 *LC_COLLATE*
80824 Determine the order in which output is sorted for the *-x* option. The POSIX locale
80825 determines the order in which the *tagsfile* is written.80826 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
80827 characters (for example, single-byte as opposed to multi-byte characters in
80828 arguments and input files). When processing C-language source code, if the locale
80829 is not compatible with the C locale described by the ISO C standard, the results are
80830 unspecified.80831 *LC_MESSAGES*
80832 Determine the locale that should be used to affect the format and contents of
80833 diagnostic messages written to standard error.80834 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.80835 **ASYNCHRONOUS EVENTS**

80836 Default.

80837 **STDOUT**80838 The list of object name information produced by the *-x* option shall be written to standard
80839 output in the following format:80840 "%s %d %s %s", *<object-name>*, *<line-number>*, *<filename>*, *<text>*80841 where *<text>* is the text of line *<line-number>* of file *<filename>*.80842 **STDERR**

80843 The standard error shall be used only for diagnostic messages.

80844 **OUTPUT FILES**80845 When the *-x* option is not specified, the format of the output file shall be:80846 "%s\t%s\t/%s\n", *<identifier>*, *<filename>*, *<pattern>*80847 where *<pattern>* is a search pattern that could be used by an editor to find the defining instance
80848 of *<identifier>* in *<filename>* (where *defining instance* is indicated by the declarations listed in the
80849 EXTENDED DESCRIPTION).80850 An optional circumflex (*'^'*) can be added as a prefix to *<pattern>*, and an optional dollar sign
80851 can be appended to *<pattern>* to indicate that the pattern is anchored to the beginning (end) of a
80852 line of text. Any slash or backslash characters in *<pattern>* shall be preceded by a backslash
80853 character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be
80854 considered part of the search pattern. All other characters in the search pattern shall be
80855 considered literal characters.

80856 An alternative format is:

80857 "%s\t%s\t?%s?\n", *<identifier>*, *<filename>*, *<pattern>*80858 which is identical to the first format except that slashes in *<pattern>* shall not be preceded by
80859 escaping backslash characters, and question mark characters in *<pattern>* shall be preceded by
80860 backslash characters.

80861 A second alternative format is:
 80862 "%s\t%s\t%d\n", <identifier>, <filename>, <lineno>

80863 where <lineno> is a decimal line number that could be used by an editor to find <identifier> in
 80864 <filename>.

80865 Neither alternative format shall be produced by *ctags* when it is used as described by
 80866 POSIX.1-200x, but the standard utilities that process tags files shall be able to process those
 80867 formats as well as the first format.

80868 In any of these formats, the file shall be sorted by identifier, based on the collation sequence in
 80869 the POSIX locale.

80870 EXTENDED DESCRIPTION

80871 If the operand identifies C-language source, the *ctags* utility shall attempt to produce an output
 80872 line for each of the following objects:

- 80873 • Function definitions
- 80874 • Type definitions
- 80875 • Macros with arguments

80876 It may also produce output for any of the following objects:

- 80877 • Function prototypes
- 80878 • Structures
- 80879 • Unions
- 80880 • Global variable definitions
- 80881 • Enumeration types
- 80882 • Macros without arguments
- 80883 • **#define** statements
- 80884 • **#line** statements

80885 Any **#if** and **#ifdef** statements shall produce no output. The tag **main** is treated specially in C
 80886 programs. The tag formed shall be created by prefixing **M** to the name of the file, with the
 80887 trailing **.c**, and leading pathname components (if any) removed.

80888 On systems that do not support the C-Language Development Utilities option, *ctags* produces
 80889 unspecified results for C-language source code files. It should write to standard error a message
 80890 identifying this condition and cause a non-zero exit status to be produced.

80891 If the operand identifies FORTRAN source, the *ctags* utility shall produce an output line for each
 80892 function definition. It may also produce output for any of the following objects:

- 80893 • Subroutine definitions
- 80894 • COMMON statements
- 80895 • PARAMETER statements
- 80896 • DATA and BLOCK DATA statements
- 80897 • Statement numbers

80898 On systems that do not support the FORTRAN Development Utilities option, *ctags* produces
 80899 unspecified results for FORTRAN source code files. It should write to standard error a message
 80900 identifying this condition and cause a non-zero exit status to be produced.

80901 It is implementation-defined what other objects (including duplicate identifiers) produce output.

80902 **EXIT STATUS**

80903 The following exit values shall be returned:

80904 0 Successful completion.

80905 >0 An error occurred.

80906 **CONSEQUENCES OF ERRORS**

80907 Default.

80908 **APPLICATION USAGE**

80909 The output with `-x` is meant to be a simple index that can be written out as an off-line readable
 80910 function index. If the input files to `ctags` (such as `.c` files) were not created using the same locale
 80911 as that in effect when `ctags -x` is run, results might not be as expected.

80912 The description of C-language processing says “attempts to” because the C language can be
 80913 greatly confused, especially through the use of `#defines`, and this utility would be of no use if
 80914 the real C preprocessor were run to identify them. The output from `ctags` may be fooled and
 80915 incorrect for various constructs.

80916 **EXAMPLES**

80917 None.

80918 **RATIONALE**

80919 The option list was significantly reduced from that provided by historical implementations. The
 80920 `-F` option was omitted as redundant, since it is the default. The `-B` option was omitted as being
 80921 of very limited usefulness. The `-t` option was omitted since the recognition of `typedefs` is now
 80922 required for C source files. The `-u` option was omitted because the update function was judged
 80923 to be not only inefficient, but also rarely needed.

80924 An early proposal included a `-w` option to suppress warning diagnostics. Since the types of such
 80925 diagnostics could not be described, the option was omitted as being not useful.

80926 The text for `LC_CTYPE` about compatibility with the C locale acknowledges that the ISO C
 80927 standard imposes requirements on the locale used to process C source. This could easily be a
 80928 superset of that known as “the C locale” by way of implementation extensions, or one of a few
 80929 alternative locales for systems supporting different codesets. No statement is made for
 80930 FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar
 80931 locale concept. However, a general rule in this volume of POSIX.1-200x is that any time that
 80932 locales do not match (preparing a file for one locale and processing it in another), the results are
 80933 suspect.

80934 The collation sequence of the tags file is not affected by `LC_COLLATE` because it is typically not
 80935 used by human readers, but only by programs such as `vi` to locate the tag within the source files.
 80936 Using the POSIX locale eliminates some of the problems of coordinating locales between the
 80937 `ctags` file creator and the `vi` file reader.

80938 Historically, the tags file has been used only by `ex` and `vi`. However, the format of the tags file
 80939 has been published to encourage other programs to use the tags in new ways. The format allows
 80940 either patterns or line numbers to find the identifiers because the historical `vi` recognizes either.
 80941 The `ctags` utility does not produce the format using line numbers because it is not useful
 80942 following any source file changes that add or delete lines. The documented search patterns
 80943 match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign
 80944 characters in the search pattern will only behave correctly if anchored to the beginning of the
 80945 line or end of the line by an additional circumflex or dollar-sign character.

80946 Historical implementations also understand the objects used by the languages Pascal and
 80947 sometimes LISP, and they understand the C source output by `lex` and `yacc`. The `ctags` utility is not
 80948 required to accommodate these languages, although implementors are encouraged to do so.

80949 The following historical option was not specified, as `vgrind` is not included in this volume of

80950
80951
80952
80953
80954
80955
80956
80957
80958
80959
80960
80961
80962
80963
80964
80965
80966
80967
80968
80969
80970
80971
80972
80973
80974
80975

POSIX.1-200x:

-v If the **-v** flag is given, an index of the form expected by *vgrind* is produced on the standard output. This listing contains the function name, filename, and page number (assuming 64-line pages). Since the output is sorted into lexicographic order, it may be desired to run the output through *sort -f*. Sample use:

```
ctags -v files | sort -f > index vgrind -x index
```

The special treatment of the tag **main** makes the use of *ctags* practical in directories with more than one program.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *fort77*, *vi*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

First released in Issue 4.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE PASC Interpretation 1003.2 #168 is applied, changing “create” to “write” in the DESCRIPTION.

Issue 7

The *ctags* utility is no longer dependent on support for the User Portability Utilities option.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

80976 **NAME**
 80977 `cut` — cut out selected fields of each line of a file

80978 **SYNOPSIS**
 80979 `cut -b list [-n] [file...]`
 80980 `cut -c list [file...]`
 80981 `cut -f list [-d delim] [-s] [file...]`

80982 **DESCRIPTION**
 80983 The `cut` utility shall cut out bytes (`-b` option), characters (`-c` option), or character-delimited fields
 80984 (`-f` option) from each line in one or more files, concatenate them, and write them to standard
 80985 output.

80986 **OPTIONS**
 80987 The `cut` utility shall conform to XBD [Section 12.2](#) (on page 201).

80988 The application shall ensure that the option-argument *list* (see options `-b`, `-c`, and `-f` below) is a
 80989 comma-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be
 80990 in three forms. The first is two positive numbers separated by a hyphen (*low-high*), which
 80991 represents all fields from the first number to the second number. The second is a positive
 80992 number preceded by a hyphen (*-high*), which represents all fields from field number 1 to that
 80993 number. The third is a positive number followed by a hyphen (*low-*), which represents that
 80994 number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be
 80995 specified in any order, but the bytes, characters, or fields selected shall be written in the order of
 80996 the input data. If an element appears in the selection list more than once, it shall be written
 80997 exactly once.

80998 The following options shall be supported:

80999 `-b list` Cut based on a *list* of bytes. Each selected byte shall be output unless the `-n` option
 81000 is also specified. It shall not be an error to select bytes not present in the input line.

81001 `-c list` Cut based on a *list* of characters. Each selected character shall be output. It shall
 81002 not be an error to select characters not present in the input line.

81003 `-d delim` Set the field delimiter to the character *delim*. The default is the <tab>.

81004 `-f list` Cut based on a *list* of fields, assumed to be separated in the file by a delimiter
 81005 character (see `-d`). Each selected field shall be output. Output fields shall be
 81006 separated by a single occurrence of the field delimiter character. Lines with no field
 81007 delimiters shall be passed through intact, unless `-s` is specified. It shall not be an
 81008 error to select fields not present in the input line.

81009 `-n` Do not split characters. When specified with the `-b` option, each element in *list* of
 81010 the form *low-high* (hyphen-separated numbers) shall be modified as follows:

- 81011 • If the byte selected by *low* is not the first byte of a character, *low* shall be
 81012 decremented to select the first byte of the character originally selected by *low*.
 81013 If the byte selected by *high* is not the last byte of a character, *high* shall be
 81014 decremented to select the last byte of the character prior to the character
 81015 originally selected by *high*, or zero if there is no prior character. If the
 81016 resulting range element has *high* equal to zero or *low* greater than *high*, the list
 81017 element shall be dropped from *list* for that input line without causing an
 81018 error.

81019 Each element in *list* of the form *low-* shall be treated as above with *high* set to the
 81020 number of bytes in the current line, not including the terminating <newline>. Each

81021 element in *list* of the form *-high* shall be treated as above with *low* set to 1. Each
 81022 element in *list* of the form *num* (a single number) shall be treated as above with *low*
 81023 set to *num* and *high* set to *num*.

81024 **-s** Suppress lines with no delimiter characters, when used with the **-f** option. Unless
 81025 specified, lines with no delimiters shall be passed through untouched.

81026 OPERANDS

81027 The following operand shall be supported:

81028 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
 81029 *'-'*, the standard input shall be used.

81030 STDIN

81031 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.
 81032 See the INPUT FILES section.

81033 INPUT FILES

81034 The input files shall be text files, except that line lengths shall be unlimited.

81035 ENVIRONMENT VARIABLES

81036 The following environment variables shall affect the execution of *cut*:

81037 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 81038 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 81039 variables used to determine the values of locale categories.)

81040 *LC_ALL* If set to a non-empty string value, override the values of all the other
 81041 internationalization variables.

81042 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 81043 characters (for example, single-byte as opposed to multi-byte characters in
 81044 arguments and input files).

81045 *LC_MESSAGES*
 81046 Determine the locale that should be used to affect the format and contents of
 81047 diagnostic messages written to standard error.

81048 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81049 ASYNCHRONOUS EVENTS

81050 Default.

81051 STDOUT

81052 The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of
 81053 the following):

81054 "%s\n", <concatenation of bytes>

81055 "%s\n", <concatenation of characters>

81056 "%s\n", <concatenation of fields and field delimiters>

81057 STDERR

81058 The standard error shall be used only for diagnostic messages.

81059 OUTPUT FILES

81060 None.

81061 EXTENDED DESCRIPTION

81062 None.

81063 **EXIT STATUS**

81064 The following exit values shall be returned:

81065 0 All input files were output successfully.

81066 >0 An error occurred.

81067 **CONSEQUENCES OF ERRORS**

81068 Default.

81069 **APPLICATION USAGE**

81070 Earlier versions of the *cut* utility worked in an environment where bytes and characters were
 81071 considered equivalent (modulo <backspace> and <tab> processing in some implementations). In
 81072 the extended world of multi-byte characters, the new **-b** option has been added. The **-n** option
 81073 (used with **-b**) allows it to be used to act on bytes rounded to character boundaries. The
 81074 algorithm specified for **-n** guarantees that:

81075 `cut -b 1-500 -n file > file1`81076 `cut -b 501- -n file > file2`

81077 ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is,
 81078 however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

81079 **EXAMPLES**

81080 Examples of the option qualifier list:

81081 1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

81082 1-3,8 Equivalent to 1,2,3,8.

81083 -5,10 Equivalent to 1,2,3,4,5,10.

81084 3- Equivalent to third to last, inclusive.

81085 The *low-high* forms are not always equivalent when used with **-b** and **-n** and multi-byte
 81086 characters; see the description of **-n**.

81087 The following command:

81088 `cut -d : -f 1,6 /etc/passwd`

81089 reads the System V password file (user database) and produces lines of the form:

81090 `<user ID>:<home directory>`

81091 Most utilities in this volume of POSIX.1-200x work on text files. The *cut* utility can be used to
 81092 turn files with arbitrary line lengths into a set of text files containing the same data. The *paste*
 81093 utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file**
 81094 contains long lines:

81095 `cut -b 1-500 -n file > file1`81096 `cut -b 501- -n file > file2`

81097 creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that
 81098 contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in
 81099 **file** that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from **file1**
 81100 and **file2** using the command:

81101 `paste -d "\0" file1 file2 > file`81102 **RATIONALE**

81103 Some historical implementations do not count <backspace>s in determining character counts
 81104 with the **-c** option. This may be useful for using *cut* for processing *nroff* output. It was
 81105 deliberately decided not to have the **-c** option treat either <backspace>s or <tab>s in any special
 81106 fashion. The *fold* utility does treat these characters specially.

81107 Unlike other utilities, some historical implementations of *cut* exit after not finding an input file,
 81108 rather than continuing to process the remaining *file* operands. This behavior is prohibited by this
 81109 volume of POSIX.1-200x, where only the exit status is affected by this problem.

81110 The behavior of *cut* when provided with either mutually-exclusive options or options that do
 81111 not work logically together has been deliberately left unspecified in favor of global wording in
 81112 [Section 1.4](#) (on page 2235).

81113 The OPTIONS section was changed in response to IEEE PASC Interpretation 1003.2 #149. The
 81114 change represents historical practice on all known systems. The original standard was
 81115 ambiguous on the nature of the output.

81116 The *list* option-arguments are historically used to select the portions of the line to be written, but
 81117 do not affect the order of the data. For example:

```
81118 echo abcdefghi | cut -c6,2,4-7,1
```

81119 yields "abdefg".

81120 A proposal to enhance *cut* with the following option:

81121 **-o** Preserve the selected field order. When this option is specified, each byte, character, or field
 81122 (or ranges of such) shall be written in the order specified by the *list* option-argument, even if
 81123 this requires multiple outputs of the same bytes, characters, or fields.

81124 was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft
 81125 standard.

81126 FUTURE DIRECTIONS

81127 None.

81128 SEE ALSO

81129 [Section 2.5](#) (on page 2249), *grep*, *paste*

81130 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

81131 CHANGE HISTORY

81132 First released in Issue 2.

81133 Issue 6

81134 The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

81135 The normative text is reworded to avoid use of the term “must” for application requirements.

81136 Issue 7

81137 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81138 **NAME**81139 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)81140 **SYNOPSIS**81141 XSI `cxref [-cs] [-o file] [-w num] [-D name[=def]]... [-I dir]...`
81142 `[-U name]... file...`81143 **DESCRIPTION**81144 The *cxref* utility shall analyze a collection of C-language *files* and attempt to build a cross-
81145 reference table. Information from **#define** lines shall be included in the symbol table. A sorted
81146 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*
81147 separately, or with the `-c` option, in combination. Each symbol shall contain an asterisk before
81148 the declaring reference.81149 **OPTIONS**81150 The *cxref* utility shall conform to XBD [Section 12.2](#) (on page 201), except that the order of the `-D`,
81151 `-I`, and `-U` options (which are identical to their interpretation by *c99*) is significant. The
81152 following options shall be supported:

- 81153 `-c` Write a combined cross-reference of all input files.
- 81154 `-s` Operate silently; do not print input filenames.
- 81155 `-o file` Direct output to named *file*.
- 81156 `-w num` Format output no wider than *num* (decimal) columns. This option defaults to 80 if
81157 *num* is not specified or is less than 51.
- 81158 `-D` Equivalent to *c99*.
- 81159 `-I` Equivalent to *c99*.
- 81160 `-U` Equivalent to *c99*.

81161 **OPERANDS**

81162 The following operand shall be supported:

- 81163
- file*
- A pathname of a C-language source file.

81164 **STDIN**

81165 Not used.

81166 **INPUT FILES**

81167 The input files are C-language source files.

81168 **ENVIRONMENT VARIABLES**81169 The following environment variables shall affect the execution of *cxref*:

- 81170 *LANG* Provide a default value for the internationalization variables that are unset or null.
81171 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
81172 variables used to determine the values of locale categories.)
- 81173 *LC_ALL* If set to a non-empty string value, override the values of all the other
81174 internationalization variables.
- 81175 *LC_COLLATE*
81176 Determine the locale for the ordering of the output.

81177 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 81178 characters (for example, single-byte as opposed to multi-byte characters in
 81179 arguments and input files).

81180 **LC_MESSAGES**
 81181 Determine the locale that should be used to affect the format and contents of
 81182 diagnostic messages written to standard error.

81183 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81184 ASYNCHRONOUS EVENTS

81185 Default.

81186 STDOUT

81187 The standard output shall be used for the cross-reference listing, unless the **-o** option is used to
 81188 select a different output file.

81189 The format of standard output is unspecified, except that the following information shall be
 81190 included:

- 81191 • If the **-c** option is not specified, each portion of the listing shall start with the name of the
 81192 input file on a separate line.
- 81193 • The name line shall be followed by a sorted list of symbols, each with its associated
 81194 location pathname, the name of the function in which it appears (if it is not a function
 81195 name itself), and line number references.
- 81196 • Each line number may be preceded by an asterisk (**'*'**) flag, meaning that this is the
 81197 declaring reference. Other single-character flags, with implementation-defined meanings,
 81198 may be included.

81199 STDERR

81200 The standard error shall be used only for diagnostic messages.

81201 OUTPUT FILES

81202 The output file named by the **-o** option shall be used instead of standard output.

81203 EXTENDED DESCRIPTION

81204 None.

81205 EXIT STATUS

81206 The following exit values shall be returned:

- 81207 0 Successful completion.
- 81208 >0 An error occurred.

81209 CONSEQUENCES OF ERRORS

81210 Default.

81211 APPLICATION USAGE

81212 None.

81213 EXAMPLES

81214 None.

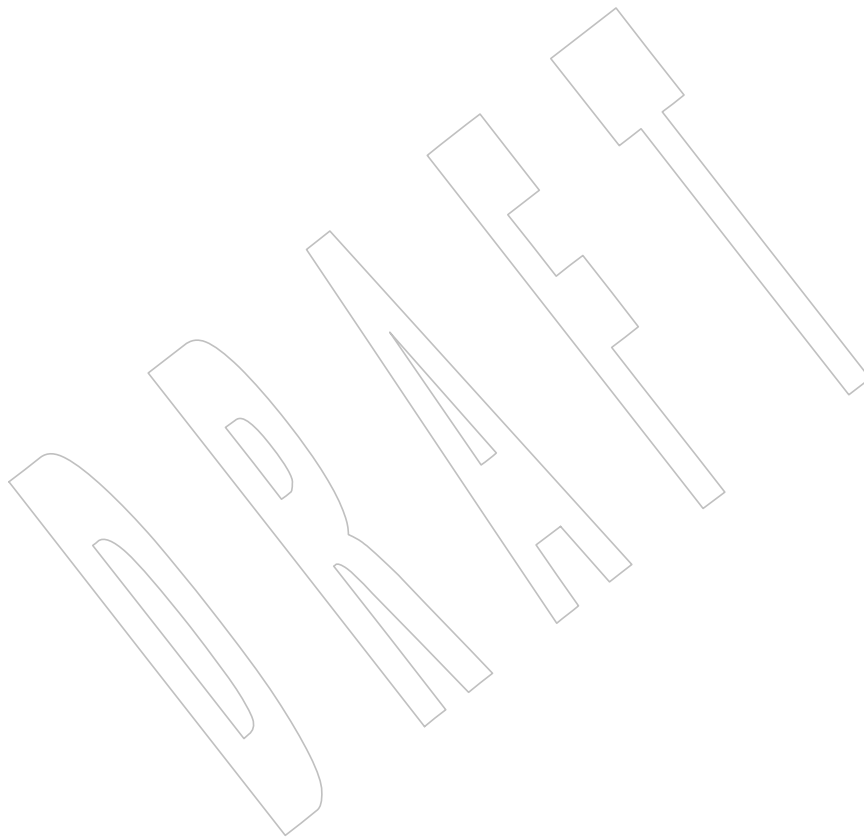
81215 RATIONALE

81216 None.

81217 FUTURE DIRECTIONS

81218 None.

- 81219 **SEE ALSO**
- 81220 [c99](#)
- 81221 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +
- 81222 **CHANGE HISTORY**
- 81223 First released in Issue 2.
- 81224 **Issue 5**
- 81225 In the SYNOPSIS, [-U *dir*] is changed to [-U *name*].
- 81226 **Issue 6**
- 81227 The APPLICATION USAGE section is added.
- 81228 **Issue 7**
- 81229 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



81230 **NAME**

81231 date — write the date and time

81232 **SYNOPSIS**

81233 date [-u] [+format]

81234 XSI date [-u] *mmddhhmm*[[*cc*]*yy*]81235 **DESCRIPTION**

81236 XSI The *date* utility shall write the date and time to standard output or attempt to set the system
 81237 date and time. By default, the current date and time shall be written. If an operand beginning
 81238 with '+' is specified, the output format of *date* shall be controlled by the conversion
 81239 specifications and other text in the operand.

81240 **OPTIONS**81241 The *date* utility shall conform to XBD Section 12.2 (on page 201).

81242 The following option shall be supported:

81243 **-u** Perform operations as if the *TZ* environment variable was set to the string "UTC0",
 81244 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the timezone
 81245 indicated by the *TZ* environment variable or the system default if that variable is
 81246 unset or null.

81247 **OPERANDS**

81248 The following operands shall be supported:

81249 **+format** When the format is specified, each conversion specifier shall be replaced in the
 81250 standard output by its corresponding value. All other characters shall be copied to
 81251 the output without change. The output shall always be terminated with a
 81252 <newline>.

81253 **Conversion Specifications**

81254 %a Locale's abbreviated weekday name.
 81255 %A Locale's full weekday name.
 81256 %b Locale's abbreviated month name.
 81257 %B Locale's full month name.
 81258 %c Locale's appropriate date and time representation.
 81259 %C Century (a year divided by 100 and truncated to an integer) as a decimal
 81260 number [00,99].
 81261 %d Day of the month as a decimal number [01,31].
 81262 %D Date in the format *mm/dd/yy*.
 81263 %e Day of the month as a decimal number [1,31] in a two-digit field with
 81264 leading space character fill.
 81265 %h A synonym for %b.
 81266 %H Hour (24-hour clock) as a decimal number [00,23].
 81267 %I Hour (12-hour clock) as a decimal number [01,12].

81268	%j	Day of the year as a decimal number [001,366].
81269	%m	Month as a decimal number [01,12].
81270	%M	Minute as a decimal number [00,59].
81271	%n	A <newline>.
81272	%P	Locale's equivalent of either AM or PM.
81273	%r	12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.
81274		
81275	%S	Seconds as a decimal number [00,60].
81276	%t	A <tab>.
81277	%T	24-hour clock time [00,23] in the format <i>HH:MM:SS</i> .
81278	%u	Weekday as a decimal number [1,7] (1=Monday).
81279	%U	Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.
81280		
81281		
81282	%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.
81283		
81284		
81285		
81286	%w	Weekday as a decimal number [0,6] (0=Sunday).
81287	%W	Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.
81288		
81289		
81290	%x	Locale's appropriate date representation.
81291	%X	Locale's appropriate time representation.
81292	%y	Year within century [00,99].
81293	%Y	Year with century as a decimal number.
81294	%Z	Timezone name, or no characters if no timezone is determinable.
81295	%%	A percent sign character.
81296		See XBD Section 7.3.5 (on page 144) for the conversion specifier values in the
81297		POSIX locale.

Modified Conversion Specifications

Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate a different format or specification as specified in the *LC_TIME* locale description (see XBD [Section 7.3.5](#), on page 144). If the corresponding keyword (see **era**, **era_year**, **era_d_fmt**, and **alt_digits** in XBD [Section 7.3.5](#), on page 144) is not specified or not supported for the current locale, the unmodified conversion specifier value shall be used.

81305	%Ec	Locale's alternative appropriate date and time representation.
81306	%EC	The name of the base year (period) in the locale's alternative representation.
81307		

81308	%Ex	Locale's alternative date representation.
81309	%EX	Locale's alternative time representation.
81310	%Ey	Offset from %EC (year only) in the locale's alternative representation.
81311	%EY	Full alternative year representation.
81312	%Od	Day of month using the locale's alternative numeric symbols.
81313	%Oe	Day of month using the locale's alternative numeric symbols.
81314	%OH	Hour (24-hour clock) using the locale's alternative numeric symbols.
81315	%OI	Hour (12-hour clock) using the locale's alternative numeric symbols.
81316	%Om	Month using the locale's alternative numeric symbols.
81317	%OM	Minutes using the locale's alternative numeric symbols.
81318	%OS	Seconds using the locale's alternative numeric symbols.
81319	%Ou	Weekday as a number in the locale's alternative representation (Monday = 1).
81320		
81321	%OU	Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
81322		
81323	%OV	Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols.
81324		
81325	%Ow	Weekday as a number in the locale's alternative representation (Sunday = 0).
81326		
81327	%OW	Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
81328		
81329	%Oy	Year (offset from %C) in alternative representation.

81330	XSI	<code>mmdhmm[[cc]yy]</code>
81331		Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first <i>mm</i> is the month (number); <i>dd</i> is the day (number); <i>hh</i> is the hour (number, 24-hour system); the second <i>mm</i> is the minute (number); <i>cc</i> is the century and is the first two digits of the year (this is optional); <i>yy</i> is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if <i>yy</i> is omitted.

81340	Note:	It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
81341		
81342		

81343	STDIN	
81344		Not used.

81345	INPUT FILES	
81346		None.

81347	ENVIRONMENT VARIABLES	
81348		The following environment variables shall affect the execution of <i>date</i> :

81349	LANG	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)
81350		
81351		

- 81352 *LC_ALL* If set to a non-empty string value, override the values of all the other
81353 internationalization variables.
- 81354 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
81355 characters (for example, single-byte as opposed to multi-byte characters in
81356 arguments).
- 81357 *LC_MESSAGES*
81358 Determine the locale that should be used to affect the format and contents of
81359 diagnostic messages written to standard error.
- 81360 *LC_TIME* Determine the format and contents of date and time strings written by *date*.
- 81361 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 81362 *TZ* Determine the timezone in which the time and date are written, unless the *-u*
81363 option is specified. If the *TZ* variable is unset or null and *-u* is not specified, an
81364 unspecified system default timezone is used.

ASYNCHRONOUS EVENTS

- 81365 Default.

STDOUT

- 81368 When no formatting operand is specified, the output in the POSIX locale shall be equivalent to
81369 specifying:
81370 *date* "+%a %b %e %H:%M:%S %Z %Y"

STDERR

- 81372 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

- 81374 None.

EXTENDED DESCRIPTION

- 81376 None.

EXIT STATUS

- 81378 The following exit values shall be returned:
- 81379 0 The date was written successfully.
- 81380 >0 An error occurred.

CONSEQUENCES OF ERRORS

- 81382 Default.

APPLICATION USAGE

- 81384 Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can
81385 contain <newline>s in some locales, so it may be difficult to use the format shown in standard
81386 output for parsing the output of *date* in those locales.
- 81387 The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap
81388 second.
- 81389 Although certain of the conversion specifiers in the POSIX locale (such as the name of the
81390 month) are shown with initial capital letters, this need not be the case in other locales. Programs
81391 using these fields may need to adjust the capitalization if the output is going to be used at the
81392 beginning of a sentence.
- 81393 The date string formatting capabilities are intended for use in Gregorian-style calendars,
81394 possibly with a different starting year (or years). The %x and %c conversion specifications,
81395 however, are intended for local representation; these may be based on a different, non-Gregorian
81396 calendar.

81397 The %C conversion specification was introduced to allow a fallback for the %EC (alternative year
 81398 format base year); it can be viewed as the base of the current subdivision in the Gregorian
 81399 calendar. The century number is calculated as the year divided by 100 and truncated to an
 81400 integer; it should not be confused with the use of ordinal numbers for centuries (for example,
 81401 "twenty-first century".) Both the %Ey and %y can then be viewed as the offset from %EC and %C,
 81402 respectively.

81403 The E and O modifiers modify the traditional conversion specifiers, so that they can always be
 81404 used, even if the implementation (or the current locale) does not support the modifier.

81405 The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as
 81406 these are based on the Gregorian calendar system. Extending the E modifiers to other date
 81407 elements may provide an implementation-defined extension capable of supporting other
 81408 calendar systems, especially in combination with the O modifier.

81409 The O modifier supports time and date formats using the locale's alternative numerical symbols,
 81410 such as Kanji or Hindi digits or ordinal number representation.

81411 Non-European locales, whether they use Latin digits in computational items or not, often have
 81412 local forms of the digits for use in date formats. This is not totally unknown even in Europe; a
 81413 variant of dates uses Roman numerals for the months: the third day of September 1991 would be
 81414 written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking
 81415 countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion
 81416 specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O
 81417 modifier was introduced to support the use for display purposes of non-Latin digits. In the
 81418 LC_TIME category in *localedef*, the optional **alt_digits** keyword is intended for this purpose. As
 81419 an example, assume the following (partial) *localedef* source:

```
81420 alt_digits  " "; "I"; "II"; "III"; "IV"; "V"; "VI"; "VII"; "VIII" \
81421            "IX"; "X"; "XI"; "XII"
81422 d_fmt      "%e.%Om.%Y"
```

81423 With the above date, the command:

```
81424 date "+%x"
```

81425 would yield 3.IX.1991. With the same **d_fmt**, but without the **alt_digits**, the command would
 81426 yield 3.9.1991.

81427 EXAMPLES

- 81428 1. The following are input/output examples of *date* used at arbitrary times in the POSIX
 81429 locale:

```
81430 $ date
81431 Tue Jun 26 09:58:10 PDT 1990
81432 $ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
81433 DATE: 11/02/91
81434 TIME: 13:36:16
```

```
81435 $ date "+TIME: %r"
81436 TIME: 01:36:32 PM
```

- 81437 2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:

```
81438 $ LANG=da_DK.iso_8859-1 date
81439 ons 02 okt 1991 15:03:32 CET
81440 $ LANG=da_DK.iso_8859-1 \
81441     date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
81442 DATO: onsdag den 2. oktober 1991
```

81443 **KLOKKEN: 15:03:56**

81444 3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:

81445 \$ LANG=De_DE.88591 date

81446 **Mi 02.Okt.1991, 15:01:21 MEZ**

81447 \$ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"

81448 **DATUM: Mittwoch, 02. Oktober 1991**

81449 **ZEIT: 15:02:02**

81450 4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:

81451 \$ LANG=Fr_FR.88591 date

81452 **Mer 02 oct 1991 MET 15:03:32**

81453 \$ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"

81454 **JOUR: Mercredi 02 octobre 1991**

81455 **HEURE: 15:03:56**

81456 RATIONALE

81457 Some of the new options for formatting are from the ISO C standard. The `-u` option was
 81458 introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is
 81459 allowed as an equivalent TZ value to be compatible with all of the systems using the BSD
 81460 implementation, where this option originated.

81461 The %e format conversion specification (adopted from System V) was added because the ISO C
 81462 standard conversion specifications did not provide any way to produce the historical default
 81463 *date* output during the first nine days of any month.

81464 There are two varieties of day and week numbering supported (in addition to any others created
 81465 with the locale-dependent %E and %O modifier characters):

- 81466 • The historical variety in which Sunday is the first day of the week and the weekdays
 81467 preceding the first Sunday of the year are considered week 0. These are represented by %w
 81468 and %U. A variant of this is %W, using Monday as the first day of the week, but still
 81469 referring to week 0. This view of the calendar was retained because so many historical
 81470 applications depend on it and the ISO C standard *strptime()* function, on which many *date*
 81471 implementations are based, was defined in this way.
- 81472 • The international standard, based on the ISO 8601:2000 standard where Monday is the first
 81473 weekday and the algorithm for the first week number is more complex: If the week
 81474 (Monday to Sunday) containing January 1 has four or more days in the new year, then it is
 81475 week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These
 81476 are represented by the new conversion specifications %u and %v, added as a result of
 81477 international comments.

81478 FUTURE DIRECTIONS

81479 None.

81480 SEE ALSO

81481 XBD [Section 7.3.5](#) (on page 144), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

81482 XSH [printf](#), [strptime\(\)](#)

81483 CHANGE HISTORY

81484 First released in Issue 2.

81485

Issue 5

81486

Changes are made for Year 2000 alignment.

81487

Issue 6

81488

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

81489

- The %EX modified conversion specification is added.

81490

The Open Group Corrigendum U048/2 is applied, correcting the examples.

81491

81492

The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors for consistency with the *LC_TIME* category.

81493

81494

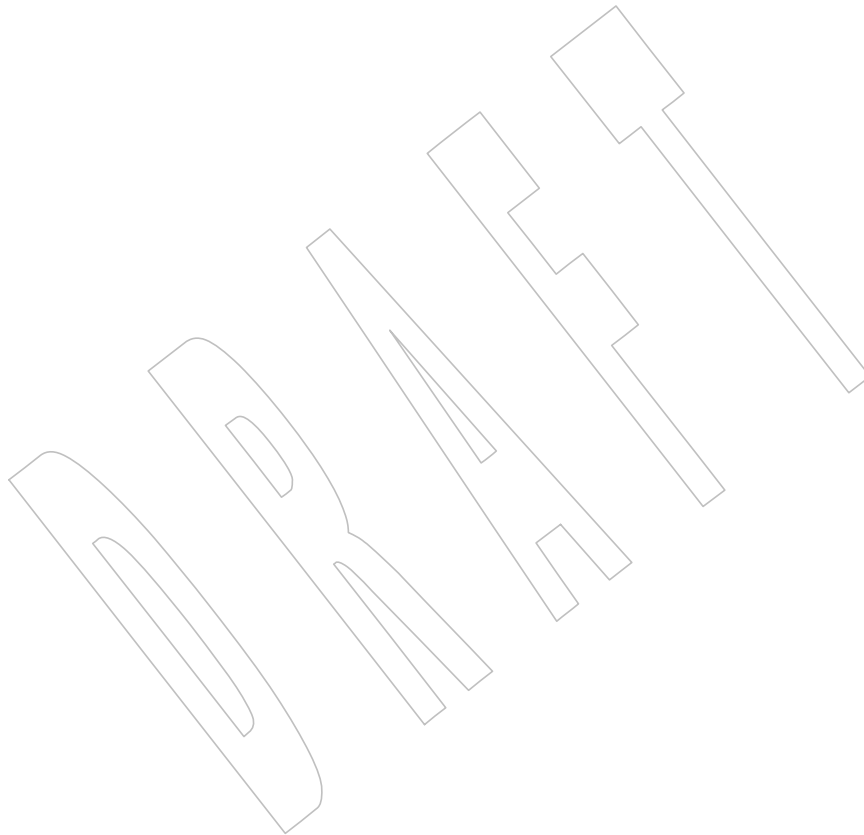
A clarification is made such that the current year is the default if the *yy* argument is omitted when setting the system date and time.

81495

81496

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/19 is applied, correcting the CHANGE HISTORY section.

81497



81498 **NAME**
 81499 dd — convert and copy a file

81500 **SYNOPSIS**
 81501 dd [*operand* . . .]

81502 **DESCRIPTION**
 81503 The *dd* utility shall copy the specified input file to the specified output file with possible
 81504 conversions using specific input and output block sizes. It shall read the input one block at a
 81505 time, using the specified input block size; it shall then process the block of data actually
 81506 returned, which could be smaller than the requested block size. It shall apply any conversions
 81507 that have been specified and write the resulting data to the output in blocks of the specified
 81508 output block size. If the **bs=expr** operand is specified and no conversions other than **sync**,
 81509 **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a
 81510 separate output block; if the read returns less than a full block and the **sync** conversion is not
 81511 specified, the resulting output block shall be the same size as the input block. If the **bs=expr**
 81512 operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the
 81513 input shall be processed and collected into full-sized output blocks until the end of the input is
 81514 reached.

81515 The processing order shall be as follows:

- 81516 1. An input block is read.
- 81517 2. If the input block is shorter than the specified input block size and the **sync** conversion is
 81518 specified, null bytes shall be appended to the input data up to the specified size. (If either
 81519 **block** or **unblock** is also specified, <space>s shall be appended instead of null bytes.) The
 81520 remaining conversions and output shall include the pad characters as if they had been
 81521 read from the input.
- 81522 3. If the **bs=expr** operand is specified and no conversion other than **sync** or **noerror** is
 81523 requested, the resulting data shall be written to the output as a single block, and the
 81524 remaining steps are omitted.
- 81525 4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If
 81526 there is an odd number of bytes in the input block, the last byte in the input record shall
 81527 not be swapped.
- 81528 5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These
 81529 conversions shall operate on the input data independently of the input blocking; an input
 81530 or output fixed-length record may span block boundaries.
- 81531 6. The data resulting from input or conversion or both shall be aggregated into output
 81532 blocks of the specified size. After the end of input is reached, any remaining output shall
 81533 be written as a block without padding if **conv=sync** is not specified; thus, the final output
 81534 block may be shorter than the output block size.

81535 **OPTIONS**
 81536 None.

81537 **OPERANDS**
 81538 All of the operands shall be processed before any input is read. The following operands shall be
 81539 supported:

81540 **if=file** Specify the input pathname; the default is standard input.

81541		of=file	Specify the output pathname; the default is standard output. If the seek=expr conversion is not also specified, the output file shall be truncated before the copy begins if an explicit of=file operand is specified, unless conv=notrunc is specified. If seek=expr is specified, but conv=notrunc is not, the effect of the copy shall be to preserve the blocks in the output file over which <i>dd</i> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy. If the input file is empty and either the size of the seek is greater than the previous size of the output file or the output file did not previously exist, the size of the output file shall be set to the file offset after the seek.)
81542			
81543			
81544			
81545			
81546			
81547			
81548			
81549			
81550			
81551			
81552		ibs=expr	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
81553		obs=expr	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
81554		bs=expr	Set both input and output block sizes to <i>expr</i> bytes, superseding ibs= and obs= . If no conversion other than sync , noerror , and notrunc is specified, each input block shall be copied to the output as a single block without aggregating short blocks.
81555			
81556			
81557		cbs=expr	Specify the conversion block size for block and unblock in bytes by <i>expr</i> (default is zero). If cbs= is omitted or given a value of zero, using block or unblock produces unspecified results.
81558			
81559			
81560	XSI		The application shall ensure that this operand is also specified if the conv= operand is specified with a value of ascii , ebcdic , or ibm . For a conv= operand with an ascii value, the input is handled as described for the unblock value, except that characters are converted to ASCII before any trailing <space>s are deleted. For conv= operands with ebcdic or ibm values, the input is handled as described for the block value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space>s are added.
81561			
81562			
81563			
81564			
81565			
81566			
81567		skip=n	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.
81568			
81569			
81570		seek=n	Skip <i>n</i> blocks (using the specified output block size) from the beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.
81571			
81572			
81573			
81574			
81575		count=n	Copy only <i>n</i> input blocks.
81576		conv=value[,value ...]	
81577			Where <i>values</i> are comma-separated symbols from the following list:
81578	XSI	ascii	Convert EBCDIC to ASCII; see Table 4-6 (on page 2520).
81579	XSI	ebcdic	Convert ASCII to EBCDIC; see Table 4-6 (on page 2520).
81580	XSI	ibm	Convert ASCII to a different EBCDIC set; see Table 4-7 (on page 2521).
81581			
81582			The ascii , ebcdic , and ibm values are mutually-exclusive.
81583		block	Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space>s shall be appended to lines
81584			
81585			
81586			
81587			

81588		that are shorter than their conversion block size to fill the block. Lines
81589		that are longer than the conversion block size shall be truncated to
81590		the largest number of characters that fit into that size; the number of
81591		truncated lines shall be reported (see the STDERR section).
81592		The block and unblock values are mutually-exclusive.
81593	unblock	Convert fixed-length records to variable length. Read a number of
81594		bytes equal to the conversion block size (or the number of bytes
81595		remaining in the input, if less than the conversion block size), delete
81596		all trailing <space>s, and append a <newline>.
81597	lcase	Map uppercase characters specified by the <i>LC_CTYPE</i> keyword
81598		tolower to the corresponding lowercase character. Characters for
81599		which no mapping is specified shall not be modified by this
81600		conversion.
81601		The lcase and ucase symbols are mutually-exclusive.
81602	ucase	Map lowercase characters specified by the <i>LC_CTYPE</i> keyword
81603		toupper to the corresponding uppercase character. Characters for
81604		which no mapping is specified shall not be modified by this
81605		conversion.
81606	swab	Swap every pair of input bytes.
81607	noerror	Do not stop processing on an input error. When an input error
81608		occurs, a diagnostic message shall be written on standard error,
81609		followed by the current input and output block counts in the same
81610		format as used at completion (see the STDERR section). If the sync
81611		conversion is specified, the missing input shall be replaced with null
81612		bytes and processed normally; otherwise, the input block shall be
81613		omitted from the output.
81614	notrunc	Do not truncate the output file. Preserve blocks in the output file not
81615		explicitly written by this invocation of the <i>dd</i> utility. (See also the
81616		preceding of=file operand.)
81617	sync	Pad every input block to the size of the ibs= buffer, appending null
81618		bytes. (If either block or unblock is also specified, append <space>s,
81619		rather than null bytes.)

81620 The behavior is unspecified if operands other than **conv=** are specified more than once.

81621 For the **bs=**, **cbs=**, **ibs=**, and **obs=** operands, the application shall supply an expression
81622 specifying a size in bytes. The expression, *expr*, can be:

- 81623 1. A positive decimal number
- 81624 2. A positive decimal number followed by *k*, specifying multiplication by 1 024
- 81625 3. A positive decimal number followed by *b*, specifying multiplication by 512
- 81626 4. Two or more positive decimal numbers (with or without *k* or *b*) separated by *x*, specifying
81627 the product of the indicated values

81628 All of the operands are processed before any input is read.

81629 XSI The following two tables display the octal number character values used for the **ascii** and **ebcdic**
81630 conversions (first table) and for the **ibm** conversion (second table). In both tables, the ASCII
81631 values are the row and column headers and the EBCDIC values are found at their intersections.
81632 For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The
81633 inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one

correspondence with these tables. The differences between the two tables are highlighted by small boxes drawn around five entries.

Table 4-6 ASCII to EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 '
0050	0115 (0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [0340 \	0275]	0232	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223]	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0137 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0152 ;	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0112 ¢	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0241	0276	0277
0350	0312	0313	0314 ⌋	0315	0316 √	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 ⌈	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

Table 4-7 ASCII to IBM EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 \$	0154 %	0120 &	0175 '
0050	0115 (0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [0340 \	0275]	0137 ^	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117	0320 }	0241 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0232	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0255 [0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0275]	0276	0277
0350	0312	0313	0314 ⌋	0315	0316 ∫	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 ⌈	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377 EO

STDIN
If no `if=` operand is specified, the standard input shall be used. See the `INPUT FILES` section.

81640 **INPUT FILES**

81641 The input file can be any file type.

81642 **ENVIRONMENT VARIABLES**81643 The following environment variables shall affect the execution of *dd*:81644 *LANG* Provide a default value for the internationalization variables that are unset or null. |
81645 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
81646 variables used to determine the values of locale categories.)81647 *LC_ALL* If set to a non-empty string value, override the values of all the other
81648 internationalization variables.81649 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
81650 characters (for example, single-byte as opposed to multi-byte characters in
81651 arguments and input files), the classification of characters as uppercase or
81652 lowercase, and the mapping of characters from one case to the other.81653 *LC_MESSAGES*81654 Determine the locale that should be used to affect the format and contents of
81655 diagnostic messages written to standard error and informative messages written to
81656 standard output.81657 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.81658 **ASYNCHRONOUS EVENTS**81659 For SIGINT, the *dd* utility shall interrupt its current processing, write status information to
81660 standard error, and exit as though terminated by SIGINT. It shall take the standard action for all
81661 other signals; see the ASYNCHRONOUS EVENTS section in [Section 1.4](#) (on page 2235).81662 **STDOUT**81663 If no *of=* operand is specified, the standard output shall be used. The nature of the output
81664 depends on the operands selected.81665 **STDERR**81666 On completion, *dd* shall write the number of input and output blocks to standard error. In the
81667 POSIX locale the following formats shall be used:81668 "%u+%u records in\n", <number of whole input blocks>,
81669 <number of partial input blocks>81670 "%u+%u records out\n", <number of whole output blocks>,
81671 <number of partial output blocks>81672 A partial input block is one for which *read()* returned less than the input block size. A partial
81673 output block is one that was written with fewer bytes than specified by the output block size.81674 In addition, when there is at least one truncated block, the number of truncated blocks shall be
81675 written to standard error. In the POSIX locale, the format shall be:81676 "%u truncated %s\n", <number of truncated blocks>, "record" (if
81677 <number of truncated blocks> is one) "records" (otherwise)

81678 Diagnostic messages may also be written to standard error.

81679 **OUTPUT FILES**81680 If the *of=* operand is used, the output shall be the same as described in the STDOUT section.81681 **EXTENDED DESCRIPTION**

81682 None.

81683 **EXIT STATUS**

81684 The following exit values shall be returned:

81685 0 The input file was copied successfully.

81686 >0 An error occurred.

81687 **CONSEQUENCES OF ERRORS**81688 If an input error is detected and the **noerror** conversion has not been specified, any partial
81689 output block shall be written to the output file, a diagnostic message shall be written, and the
81690 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
81691 be written and the copy operation shall be discontinued.81692 **APPLICATION USAGE**

81693 The input and output block size can be specified to take advantage of raw physical I/O.

81694 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
81695 specified for the *dd* utility perform conversions for the version specified by the tables.81696 **EXAMPLES**

81697 The following command:

81698 `dd if=/dev/rmt0h of=/dev/rmt1h`

81699 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

81700 The following command:

81701 `dd ibs=10 skip=1`

81702 strips the first 10 bytes from standard input.

81703 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
81704 ASCII file *x*:81705 `dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase`81706 **RATIONALE**81707 The **OPTIONS** section is listed as “None” because there are no options recognized by historical
81708 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
81709 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
81710 of this volume of POSIX.1-200x, *dd* retains its curious JCL-like syntax due to the large number of
81711 applications that depend on the historical implementation.81712 A suggested implementation technique for **conv=noerror, sync** is to zero (or <space>-fill, if
81713 **blocking** or **unblocking**) the input buffer before each read and to write the contents of the input
81714 buffer to the output even after an error. In this manner, any data transferred to the input buffer
81715 before the error was detected is preserved. Another point is that a failed read on a regular file or
81716 a disk generally does not increment the file offset, and *dd* must then seek past the block on which
81717 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
81718 tape, however, the tape normally has passed the block containing the error when the error is
81719 reported, and thus no seek is necessary.81720 The default **ibs=** and **obs=** sizes are specified as 512 bytes because there are historical (largely
81721 portable) scripts that assume these values. If they were left unspecified, unusual results could
81722 occur if an implementation chose an odd block size.81723 Historical implementations of *dd* used *creat()* when processing **of=file**. This makes the **seek=**
81724 operand unusable except on special files. The **conv=notrunc** feature was added because more
81725 recent BSD-based implementations use *open()* (without **O_TRUNC**) instead of *creat()*, but they
81726 fail to delete output file contents after the data copied.81727 The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to

81728 mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of
81729 POSIX.1-200x.

81730 Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are
81731 taken from a common print train that does contain them. Other than those characters, the print
81732 train values are not filled in, but appear to provide some of the motivation for the historical
81733 choice of translations reflected here.

81734 The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

81735 The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ
81736 in such a way that:

- 81737 1. EBCDIC 0112 ('ϕ') and 0152 (broken pipe) do not appear in the table.
- 81738 2. EBCDIC 0137 ('¬') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC
81739 0232 (no graphic) is used.
- 81740 3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC
81741 0137 ('¬') is used.
- 81742 4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table
81743 and once in place of 0112 ('ϕ') and 0241 ('~').

81744 In net result:

81745 EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

81746 That displaced EBCDIC 0137 ('¬') in cell 0176.

81747 That displaced EBCDIC 0232 (no graphic) in cell 0136.

81748 That replaced EBCDIC 0152 (broken pipe) in cell 0313.

81749 EBCDIC 0255 ('[') replaced EBCDIC 0112 ('ϕ').

81750 This translation, however, reflects historical practice that (ASCII) '~' and '¬' were often
81751 mapped to each other, as were '[' and 'ϕ'; and ']' and (EBCDIC) '~'.

81752 The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the
81753 **ascii** operand, the input is handled as described for the **unblock** operand except that characters
81754 are converted to ASCII before the trailing <space>s are deleted. For the **ebcdic** and **ibm**
81755 operands, the input is handled as described for the **block** operand except that the characters are
81756 converted to EBCDIC or IBM EBCDIC after the trailing <space>s are added.

81757 The **block** and **unblock** keywords are from historical BSD practice.

81758 The consistent use of the word **record** in standard error messages matches most historical
81759 practice. An earlier version of System V used **block**, but this has been updated in more recent
81760 releases.

81761 Early proposals only allowed two numbers separated by x to be used in a product when
81762 specifying **bs=**, **cbs=**, **ibs=**, and **obs=** sizes. This was changed to reflect the historical practice of
81763 allowing multiple numbers in the product as provided by Version 7 and all releases of System V
81764 and BSD.

81765 A change to the **swab** conversion is required to match historical practice and is the result of IEEE
81766 PASC Interpretations 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.

81767 A change to the handling of SIGINT is required to match historical practice and is the result of
81768 IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.

81769
81770
81771
81772
81773
81774
81775
81776
81777
81778
81779
81780
81781
81782
81783
81784
81785
81786
81787

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 1.4](#) (on page 2235), *sed*, *tr*

XBD [Chapter 8](#) (on page 159)

+

CHANGE HISTORY

First released in Issue 2.

Issue 5

The second paragraph of the **cbs=** description is reworded and marked EX.

The FUTURE DIRECTIONS section is added.

Issue 6

Changes are made to **swab** conversion and SIGINT handling to align with the IEEE P1003.2b draft standard.

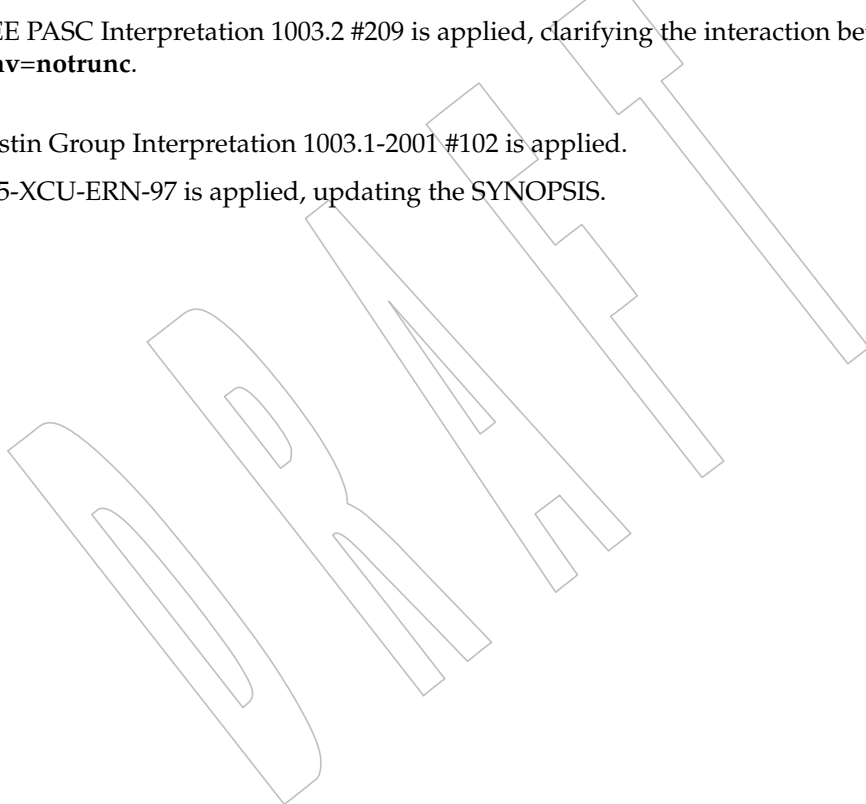
The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between **dd of=file** and **conv=notrunc**.

Issue 7

Austin Group Interpretation 1003.1-2001 #102 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



81788 **NAME**
 81789 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

81790 **SYNOPSIS**
 81791 XSI `delta [-nps] [-g list] [-m mrlist] [-r SID] [-y[comment]] file...`

81792 **DESCRIPTION**
 81793 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that
 81794 were made to the files retrieved by *get* (called the *g-files*, or generated files).

81795 **OPTIONS**
 81796 The *delta* utility shall conform to XBD Section 12.2 (on page 201), except that the `-y` option has an
 81797 optional option-argument. This optional option-argument shall not be presented as a separate
 81798 argument.

81799 The following options shall be supported:

81800 `-r SID` Uniquely identify which delta is to be made to the SCCS file. The use of this option
 81801 shall be necessary only if two or more outstanding *get* commands for editing (*get*
 81802 `-e`) on the same SCCS file were done by the same person (login name). The SID
 81803 value specified with the `-r` option can be either the SID specified on the *get*
 81804 command line or the SID to be made as reported by the *get* utility; see *get* (on page
 81805 2693).

81806 `-s` Suppress the report to standard output of the activity associated with each *file*. See
 81807 the `STDOUT` section.

81808 `-n` Specify retention of the edited *g-file* (normally removed at completion of delta
 81809 processing).

81810 `-g list` Specify a *list* (see *get* for the definition of *list*) of deltas that shall be ignored when
 81811 the file is accessed at the change level (SID) created by this delta.

81812 `-m mrlist` Specify a modification request (MR) number that the application shall supply as
 81813 the reason for creating the new delta. This shall be used if the SCCS file has the `v`
 81814 flag set; see *admin*.

81815 If `-m` is not used and `'-'` is not specified as a file argument, and the standard
 81816 input is a terminal, the prompt described in the `STDOUT` section shall be written
 81817 to standard output before the standard input is read; if the standard input is not a
 81818 terminal, no prompt shall be issued.

81819 MRs in a list shall be separated by `<blank>s` or escaped `<newline>s`. An
 81820 unescaped `<newline>` shall terminate the MR list. The escape character is
 81821 `<backslash>`.

81822 If the `v` flag has a value, it shall be taken to be the name of a program which
 81823 validates the correctness of the MR numbers. If a non-zero exit status is returned
 81824 from the MR number validation program, the *delta* utility shall terminate. (It is
 81825 assumed that the MR numbers were not all valid.)

81826 `-y[comment]` Describe the reason for making the delta. The *comment* shall be an arbitrary group
 81827 of lines that would meet the definition of a text file. Implementations shall support
 81828 *comments* from zero to 512 bytes and may support longer values. A null string
 81829 (specified as either `-y`, `-y" "`, or in response to a prompt for a comment) shall be
 81830 considered a valid *comment*.

81831 If `-y` is not specified and `'-'` is not specified as a file argument, and the standard

81832 input is a terminal, the prompt described in the STDOUT section shall be written
 81833 to standard output before the standard input is read; if the standard input is not a
 81834 terminal, no prompt shall be issued. An unescaped <newline> shall terminate the
 81835 comment text. The escape character is <backslash>.

81836 The `-y` option shall be required if the *file* operand is specified as `'-'`.

81837 **-p** Write (to standard output) the SCCS file differences before and after the delta is
 81838 applied in *diff* format; see *diff*.

81839 OPERANDS

81840 The following operand shall be supported:

81841 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta*
 81842 utility shall behave as though each file in the directory were specified as a named
 81843 file, except that non-SCCS files (last component of the pathname does not begin
 81844 with **s**.) and unreadable files shall be silently ignored.

81845 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;
 81846 each line of the standard input shall be taken to be the name of an SCCS file to be
 81847 processed. Non-SCCS files and unreadable files shall be silently ignored.

81848 STDIN

81849 The standard input shall be a text file used only in the following cases:

- 81850 • To read an *mrlist* or a *comment* (see the `-m` and `-y` options).
- 81851 • A *file* operand shall be specified as `'-'`. In this case, the `-y` option must be used to specify
 81852 the comment, and if the SCCS file has the **v** flag set, the `-m` option must also be used to
 81853 specify the MR list.

81854 INPUT FILES

81855 Input files shall be text files whose data is to be included in the SCCS files. If the first character of
 81856 any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file
 81857 contains more than 99999 lines, the number of lines recorded in the header for this file shall be
 81858 99999 for this delta.

81859 ENVIRONMENT VARIABLES

81860 The following environment variables shall affect the execution of *delta*:

81861 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 81862 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 81863 variables used to determine the values of locale categories.)

81864 *LC_ALL* If set to a non-empty string value, override the values of all the other
 81865 internationalization variables.

81866 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 81867 characters (for example, single-byte as opposed to multi-byte characters in
 81868 arguments and input files).

81869 *LC_MESSAGES*

81870 Determine the locale that should be used to affect the format and contents of
 81871 diagnostic messages written to standard error, and informative messages written
 81872 to standard output.

81873 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81874 *TZ* Determine the timezone in which the time and date are written in the SCCS file. If
 81875 the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

81876 **ASYNCHRONOUS EVENTS**

81877 If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit
 81878 code. The standard action shall be taken for all other signals; see [Section 1.4](#) (on page 2235).

81879 **STDOUT**

81880 The standard output shall be used only for the following messages in the POSIX locale:

- 81881 • Prompts (see the **-m** and **-y** options) in the following formats:

81882 "MRs? "

81883 "comments? "

81884 The MR prompt, if written, shall always precede the comments prompt.

- 81885 • A report of each file's activities (unless the **-s** option is specified) in the following format:

81886 "%s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID> ,
 81887 <number of lines inserted> , <number of lines deleted> ,
 81888 <number of lines unchanged>

81889 **STDERR**

81890 The standard error shall be used only for diagnostic messages.

81891 **OUTPUT FILES**

81892 Any SCCS files updated shall be files of an unspecified format.

81893 **EXTENDED DESCRIPTION**81894 **System Date and Time**

81895 When a *delta* is added to an SCCS file, the system date and time shall be recorded for the new
 81896 delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the
 81897 behavior is unspecified.

81898 **EXIT STATUS**

81899 The following exit values shall be returned:

81900 0 Successful completion.

81901 >0 An error occurred.

81902 **CONSEQUENCES OF ERRORS**

81903 Default.

81904 **APPLICATION USAGE**

81905 Problems can arise if the system date and time have been modified (for example, put forward
 81906 and then back again, or unsynchronized clocks across a network) and can also arise when
 81907 different values of the *TZ* environment variable are used.

81908 Problems of a similar nature can also arise for the operation of the *get* utility, which records the
 81909 date and time in the file body.

81910 **EXAMPLES**

81911 None.

81912 **RATIONALE**

81913 None.

81914 **FUTURE DIRECTIONS**

81915 None.

81916
81917
81918
81919
81920
81921
81922
81923
81924
81925
81926
81927
81928
81929
81930
81931
81932
81933
81934
81935
81936
81937
81938

SEE ALSO

Section 1.4 (on page 2235), *admin*, *diff*, *get*, *prs*, *rm del*

XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

+

CHANGE HISTORY

First released in Issue 2.

Issue 5

The output format description in the STDOUT section is corrected.

Issue 6

The APPLICATION USAGE section is added.

The normative text is reworded to avoid use of the term “must” for application requirements.

The Open Group Base Resolution bwg2001-007 is applied as follows:

- The use of ‘-’ as a file argument is clarified.
- The use of STDIN is added.
- The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that implementations re-signal themselves when catching a normally fatal signal.
- New text is added to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the TZ environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

81939 **NAME**

81940 df — report free disk space

81941 **SYNOPSIS**81942 XSI df [-k] [-P|-t] [*file...*]81943 **DESCRIPTION**

81944 XSI The *df* utility shall write the amount of available space and file slots for file systems on which
 81945 the invoking user has appropriate read access. File systems shall be specified by the *file*
 81946 operands; when none are specified, information shall be written for all file systems. The format
 81947 of the default output from *df* is unspecified, but all space figures are reported in 512-byte units,
 81948 unless the *-k* option is specified. This output shall contain at least the file system names, amount
 81949 of available space on each of these file systems, and the number of free file slots, or *inodes*,
 81950 available; when *-t* is specified, the output shall contain the total allocated space as well.

81951 **OPTIONS**81952 The *df* utility shall conform to XBD Section 12.2 (on page 201).

81953 The following options shall be supported:

81954 *-k* Use 1024-byte units, instead of the default 512-byte units, when writing space
 81955 figures.

81956 *-P* Produce output in the format described in the STDOUT section.

81957 XSI *-t* Include total allocated-space figures in the output.

81958 **OPERANDS**

81959 The following operand shall be supported:

81960 *file* A pathname of a file within the hierarchy of the desired file system. If a file other
 81961 XSI than a FIFO, a regular file, a directory, or a special file representing the device
 81962 containing the file system (for example, */dev/dsk/0s1*) is specified, the results are
 81963 unspecified. If the *file* operand names a file other than a special file containing a file
 81964 system, *df* shall write the amount of free space in the file system containing the
 81965 XSI specified *file* operand. Otherwise, *df* shall write the amount of free space in that
 81966 file system.

81967 **STDIN**

81968 Not used.

81969 **INPUT FILES**

81970 None.

81971 **ENVIRONMENT VARIABLES**81972 The following environment variables shall affect the execution of *df*:

81973 *LANG* Provide a default value for the internationalization variables that are unset or null.
 81974 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 81975 variables used to determine the values of locale categories.)

81976 *LC_ALL* If set to a non-empty string value, override the values of all the other
 81977 internationalization variables.

81978 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 81979 characters (for example, single-byte as opposed to multi-byte characters in
 81980 arguments).

81981 *LC_MESSAGES*
 81982 Determine the locale that should be used to affect the format and contents of
 81983 diagnostic messages written to standard error and informative messages written to
 81984 standard output.

81985 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

81986 ASYNCHRONOUS EVENTS

81987 Default.

81988 STDOUT

81989 When both the *-k* and *-P* options are specified, the following header line shall be written (in the
 81990 POSIX locale):

81991 "Filesystem 1024-blocks Used Available Capacity Mounted on\n"

81992 When the *-P* option is specified without the *-k* option, the following header line shall be written
 81993 (in the POSIX locale):

81994 "Filesystem 512-blocks Used Available Capacity Mounted on\n"

81995 The implementation may adjust the spacing of the header line and the individual data lines so
 81996 that the information is presented in orderly columns.

81997 The remaining output with *-P* shall consist of one line of information for each specified file
 81998 system. These lines shall be formatted as follows:

81999 "%s %d %d %d %d%% %s\n", *<file system name>*, *<total space>*,
 82000 *<space used>*, *<space free>*, *<percentage used>*,
 82001 *<file system root>*

82002 In the following list, all quantities expressed in 512-byte units (1024-byte when *-k* is specified)
 82003 shall be rounded up to the next higher unit. The fields are:

82004 *<file system name>*

82005 The name of the file system, in an implementation-defined format.

82006 *<total space>* The total size of the file system in 512-byte units. The exact meaning of this figure
 82007 is implementation-defined, but should include *<space used>*, *<space free>*, plus any
 82008 space reserved by the system not normally available to a user.

82009 *<space used>* The total amount of space allocated to existing files in the file system, in 512-byte
 82010 units.

82011 *<space free>* The total amount of space available within the file system for the creation of new
 82012 files by unprivileged users, in 512-byte units. When this figure is less than or equal
 82013 to zero, it shall not be possible to create any new files on the file system without
 82014 first deleting others, unless the process has appropriate privileges. The figure
 82015 written may be less than zero.

82016 *<percentage used>*

82017 The percentage of the normally available space that is currently allocated to all files
 82018 on the file system. This shall be calculated using the fraction:

82019
$$\frac{\textit{<space used>}}{\textit{<space used> + <space free>}}$$

82020 expressed as a percentage. This percentage may be greater than 100 if *<space free>*
 82021 is less than zero. The percentage value shall be expressed as a positive integer, with
 82022 any fractional result causing it to be rounded to the next highest integer.

82023 *<file system root>*

82024 The directory below which the file system hierarchy appears.

82025 XSI The output format is unspecified when `-t` is used.

82026 **STDERR**

82027 The standard error shall be used only for diagnostic messages.

82028 **OUTPUT FILES**

82029 None.

82030 **EXTENDED DESCRIPTION**

82031 None.

82032 **EXIT STATUS**

82033 The following exit values shall be returned:

82034 0 Successful completion.

82035 >0 An error occurred.

82036 **CONSEQUENCES OF ERRORS**

82037 Default.

82038 **APPLICATION USAGE**

82039 On most systems, the “name of the file system, in an implementation-defined format” is the
82040 special file on which the file system is mounted.

82041 On large file systems, the calculation specified for percentage used can create huge rounding
82042 errors.

82043 **EXAMPLES**

82044 1. The following example writes portable information about the `/usr` file system:

82045 `df -P /usr`

82046 2. Assuming that `/usr/src` is part of the `/usr` file system, the following produces the same
82047 output as the previous example:

82048 `df -P /usr/src`

82049 **RATIONALE**

82050 The behavior of `df` with the `-P` option is the default action of the 4.2 BSD `df` utility. The uppercase
82051 `-P` was selected to avoid collision with a known industry extension using `-p`.

82052 Historical `df` implementations vary considerably in their default output. It was therefore
82053 necessary to describe the default output in a loose manner to accommodate all known historical
82054 implementations and to add a portable option (`-P`) to provide information in a portable format.

82055 The use of 512-byte units is historical practice and maintains compatibility with `ls` and other
82056 utilities in this volume of POSIX.1-200x. This does not mandate that the file system itself be
82057 based on 512-byte blocks. The `-k` option was added as a compromise measure. It was agreed by
82058 the standard developers that 512 bytes was the best default unit because of its complete
82059 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and
82060 that a `-k` option to switch to 1024-byte units was a good compromise. Users who prefer the
82061 more logical 1024-byte quantity can easily alias `df` to `df -k` without breaking many historical
82062 scripts relying on the 512-byte units.

82063 It was suggested that `df` and the various related utilities be modified to access a `BLOCKSIZE`
82064 environment variable to achieve consistency and user acceptance. Since this is not historical
82065 practice on any system, it is left as a possible area for system extensions and will be re-evaluated
82066 in a future version if it is widely implemented.

82067
82068
82069
82070
82071
82072
82073
82074
82075
82076
82077
82078
82079
82080

FUTURE DIRECTIONS

None.

SEE ALSO

find

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

First released in Issue 2.

Issue 6

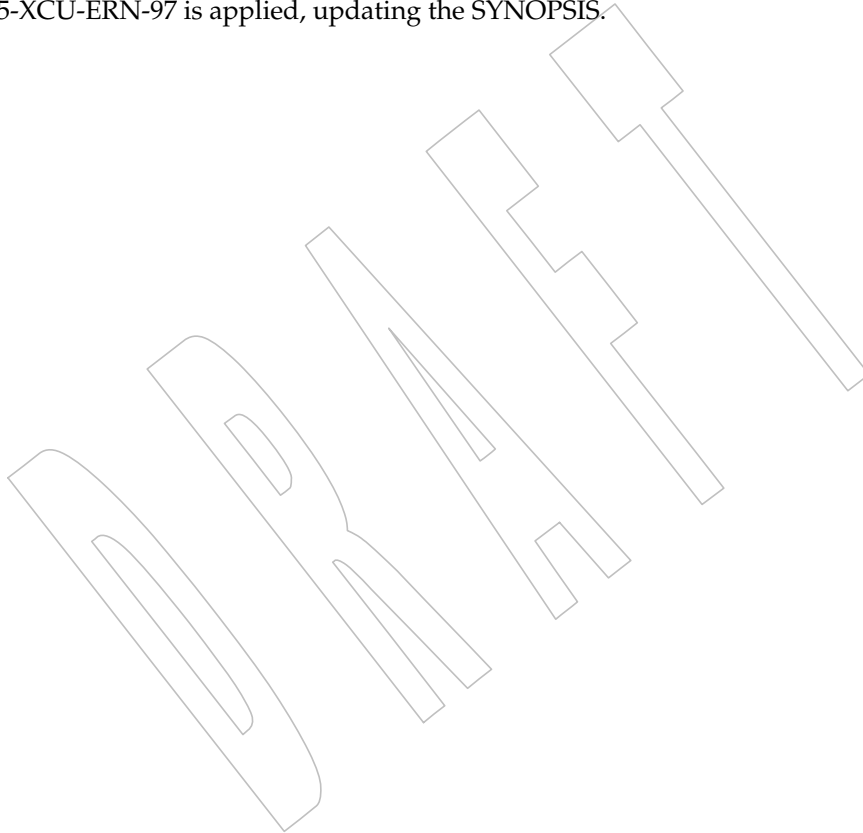
This utility is marked as part of the User Portability Utilities option.

Issue 7

Austin Group Interpretation 1003.1-2001 #099 is applied.

The *df* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



82081 **NAME**
 82082 diff — compare two files

82083 **SYNOPSIS**
 82084 diff [-c|-e|-f|-u|-C n|-U n] [-br] file1 file2

82085 **DESCRIPTION**
 82086 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of
 82087 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be
 82088 produced if the files are identical.

82089 **OPTIONS**
 82090 The *diff* utility shall conform to XBD [Section 12.2](#) (on page 201).

82091 The following options shall be supported:

- 82092 **-b** Cause any amount of white space at the end of a line to be treated as a single
 82093 <newline> (that is, the white-space characters preceding the <newline> are
 82094 ignored) and other strings of white-space characters, not including <newline>s, to
 82095 compare equal.
- 82096 **-c** Produce output in a form that provides three lines of copied context.
- 82097 **-C n** Produce output in a form that provides *n* lines of copied context (where *n* shall be
 82098 interpreted as a positive decimal integer).
- 82099 **-e** Produce output in a form suitable as input for the *ed* utility, which can then be used
 82100 to convert *file1* into *file2*.
- 82101 **-f** Produce output in an alternative form, similar in format to **-e**, but not intended to
 82102 be suitable as input for the *ed* utility, and in the opposite order.
- 82103 **-r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*
 82104 are both directories.

 82105 The *diff* utility shall detect infinite loops; that is, entering a previously visited
 82106 directory that is an ancestor of the last file encountered. When it detects an infinite
 82107 loop, *diff* shall write a diagnostic message to standard error and shall either recover
 82108 its position in the hierarchy or terminate.
- 82109 **-u** Produce output in a form that provides three lines of unified context.
- 82110 **-U n** Produce output in a form that provides *n* lines of unified context (where *n* shall be
 82111 interpreted as a non-negative decimal integer).

82112 **OPERANDS**
 82113 The following operands shall be supported:

82114 *file1, file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is '-', the
 82115 standard input shall be used in its place.

82116 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special
 82117 files, or FIFO special files to any files and shall not compare regular files to directories. Further
 82118 details are as specified in [Diff Directory Comparison Format](#) (on page 2535). The behavior of *diff*
 82119 on other file types is implementation-defined when found in directories.

82120 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file
 82121 contained in the directory file with a filename that is the same as the last component of the non-
 82122 directory file.

82123 **STDIN**

82124 The standard input shall be used only if one of the *file1* or *file2* operands references standard
 82125 input. See the INPUT FILES section.

82126 **INPUT FILES**

82127 The input files may be of any type.

82128 **ENVIRONMENT VARIABLES**

82129 The following environment variables shall affect the execution of *diff*:

82130 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 82131 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 82132 variables used to determine the values of locale categories.)

82133 **LC_ALL** If set to a non-empty string value, override the values of all the other
 82134 internationalization variables.

82135 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 82136 characters (for example, single-byte as opposed to multi-byte characters in
 82137 arguments and input files).

82138 **LC_MESSAGES**

82139 Determine the locale that should be used to affect the format and contents of
 82140 diagnostic messages written to standard error and informative messages written to
 82141 standard output.

82142 **LC_TIME** Determine the locale for affecting the format of file timestamps written with the **-C**
 82143 and **-c** options.

82144 **XSI NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

82145 **TZ** Determine the timezone used for calculating file timestamps written with a context
 82146 format. If **TZ** is unset or null, an unspecified default timezone shall be used.

82147 **ASYNCHRONOUS EVENTS**

82148 Default.

82149 **STDOUT**82150 **Diff Directory Comparison Format**

82151 If both *file1* and *file2* are directories, the following output formats shall be used.

82152 In the POSIX locale, each file that is present in only one directory shall be reported using the
 82153 following format:

82154 "Only in %s: %s\n", <directory pathname>, <filename>

82155 In the POSIX locale, subdirectories that are common to the two directories may be reported with
 82156 the following format:

82157 "Common subdirectories: %s and %s\n", <directory1 pathname>,
 82158 <directory2 pathname>

82159 For each file common to the two directories, if the two files are not to be compared: if the two
 82160 files have the same device ID and file serial number, or are both block special files that refer to
 82161 the same device, or are both character special files that refer to the same device, in the POSIX
 82162 locale the output format is unspecified. Otherwise, in the POSIX locale an unspecified format
 82163 shall be used that contains the pathnames of the two files.

82164 For each file common to the two directories, if the files are compared and are identical, no
 82165 output shall be written. If the two files differ, the following format is written:

82166 "diff %s %s %s\n", <diff_options>, <filename1>, <filename2>

82167 where *<diff_options>* are the options as specified on the command line.

82168 All directory pathnames listed in this section shall be relative to the original command line
82169 arguments. All other names of files listed in this section shall be filenames (pathname
82170 components).

82171 Diff Binary Output Format

82172 In the POSIX locale, if one or both of the files being compared are not text files, an unspecified
82173 format shall be used that contains the pathnames of two files being compared and the string
82174 "differ".

82175 If both files being compared are text files, depending on the options specified, one of the
82176 following formats shall be used to write the differences.

82177 Diff Default Output Format

82178 The default (without *-e*, *-f*, *-c*, *-C*, *-u*, or *-U* options) *diff* utility output shall contain lines of
82179 these forms:

82180 "%da%d\n", *<num1>*, *<num2>*

82181 "%da%d,%d\n", *<num1>*, *<num2>*, *<num3>*

82182 "%dd%d\n", *<num1>*, *<num2>*

82183 "%d,%dd%d\n", *<num1>*, *<num2>*, *<num3>*

82184 "%dc%d\n", *<num1>*, *<num2>*

82185 "%d,%dc%d\n", *<num1>*, *<num2>*, *<num3>*

82186 "%dc%d,%d\n", *<num1>*, *<num2>*, *<num3>*

82187 "%d,%dc%d,%d\n", *<num1>*, *<num2>*, *<num3>*, *<num4>*

82188 These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the
82189 action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d*
82190 and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in
82191 *ed*, identical pairs (where *num1=num2*) are abbreviated as a single number.

82192 Following each of these lines, *diff* shall write to standard output all lines affected in the first file
82193 using the format:

82194 "<Δ%s", *<line>*

82195 and all lines affected in the second file using the format:

82196 ">Δ%s", *<line>*

82197 If there are lines affected in both *file1* and *file2* (as with the *c* subcommand), the changes are
82198 separated with a line consisting of three hyphens:

82199 "---\n"

82200 **Diff -e Output Format**

82201 With the `-e` option, a script shall be produced that shall, when provided as input to `ed`, along
 82202 with an appended `w` (write) command, convert *file1* into *file2*. Only the `a` (append), `c` (change), `d`
 82203 (delete), `i` (insert), and `s` (substitute) commands of `ed` shall be used in this script. Text lines,
 82204 except those consisting of the single character period (`' . '`), shall be output as they appear in the
 82205 file.

82206 **Diff -f Output Format**

82207 With the `-f` option, an alternative format of script shall be produced. It is similar to that
 82208 produced by `-e`, with the following differences:

- 82209 1. It is expressed in reverse sequence; the output of `-e` orders changes from the end of the
 82210 file to the beginning; the `-f` from beginning to end.
- 82211 2. The command form `<lines> <command-letter>` used by `-e` is reversed. For example,
 82212 `10c` with `-e` would be `c10` with `-f`.
- 82213 3. The form used for ranges of line numbers is `<space>`-separated, rather than comma-
 82214 separated.

82215 **Diff -c or -C Output Format**

82216 With the `-c` or `-C` option, the output format shall consist of affected lines along with
 82217 surrounding lines of context. The affected lines shall show which ones need to be deleted or
 82218 changed in *file1*, and those added from *file2*. With the `-c` option, three lines of context, if
 82219 available, shall be written before and after the affected lines. With the `-C` option, the user can
 82220 specify how many lines of context are written. The exact format follows.

82221 The name and last modification time of each file shall be output in the following format:

```
82222 "*** %s %s\n", file1, <file1 timestamp>
82223 "--- %s %s\n", file2, <file2 timestamp>
```

82224 Each `<file>` field shall be the pathname of the corresponding file being compared. The pathname
 82225 written for standard input is unspecified.

82226 In the POSIX locale, each `<timestamp>` field shall be equivalent to the output from the following
 82227 command:

```
82228 date "+%a %b %e %T %Y"
```

82229 without the trailing `<newline>`, executed at the time of last modification of the corresponding
 82230 file (or the current time, if the file is standard input).

82231 Then, the following output formats shall be applied for every set of changes.

82232 First, a line shall be written in the following format:

```
82233 "*****\n"
```

82234 Next, the range of lines in *file1* shall be written in the following format if the range contains two
 82235 or more lines:

```
82236 "*** %d,%d ****\n", <beginning line number>, <ending line number>
```

82237 and the following format otherwise:

```
82238 "*** %d ****\n", <ending line number>
```

82239 The ending line number of an empty range shall be the number of the preceding line, or 0 if the
 82240 range is at the start of the file.

82241 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected

82242 lines shall be written in the following format:

82243 " Δ %s", <unaffected_line>

82244 Deleted lines shall be written as:

82245 "- Δ %s", <deleted_line>

82246 Changed lines shall be written as:

82247 "! Δ %s", <changed_line>

82248 Next, the range of lines in *file2* shall be written in the following format if the range contains two

82249 or more lines:

82250 "--- %d,%d ----\n", <beginning line number>, <ending line number>

82251 and the following format otherwise:

82252 "--- %d ----\n", <ending line number>

82253 Then, lines of context and changed lines shall be written as described in the previous formats.

82254 Lines added from *file2* shall be written in the following format:

82255 "+ Δ %s", <added_line>

82256 **Diff -u or -U Output Format**

82257 The -u or -U options behave like the -c or -C options, except that the context lines are not

82258 repeated; instead, the context, deleted, and added lines are shown together, interleaved. The

82259 exact format follows.

82260 The name and last modification time of each file shall be output in the following format:

82261 "--- Δ %s%s%s Δ %s0, file1, <file1 timestamp>, <file1 frac>, <file1 zone>

82262 "+++ Δ %s%s%s Δ %s0, file2, <file2 timestamp>, <file2 frac>, <file2 zone>

82263 Each <file> field shall be the pathname of the corresponding file being compared, or the single

82264 character '-' if standard input is being compared. However, if the pathname contains a <tab>

82265 or a <newline>, or if it does not consist entirely of characters taken from the portable character

82266 set, the behavior is implementation-defined.

82267 Each <timestamp> field shall be equivalent to the output from the following command:

82268 date '+%Y-%m-%d Δ %H:%M:%S'

82269 without the trailing <newline>, executed at the time of last modification of the corresponding

82270 file (or the current time, if the file is standard input).

82271 Each <frac> field shall be either empty, or a decimal point followed by at least one decimal digit,

82272 indicating the fractional-seconds part (if any) of the file timestamp. The number of fractional

82273 digits shall be at least the number needed to represent the file's timestamp without loss of

82274 information.

82275 Each <zone> field shall be of the form "shhmm", where "shh" is a signed two-digit decimal

82276 number in the range -24 through +25, and "mm" is an unsigned two-digit decimal number in the

82277 range 00 through 59. It represents the timezone of the timestamp as the number of hours (hh)

82278 and minutes (mm) east (+) or west (-) of UTC for the timestamp. If the hours and minutes are

82279 both zero, the sign shall be '+'. However, if the timezone is not an integral number of minutes

82280 away from UTC, the <zone> field is implementation-defined.

82281 Then, the following output formats shall be applied for every set of changes.

82282 First, the range of lines in each file shall be written in the following format:

82283 "@@Δ-%sΔ+%sΔ@" , <file1 range> , <file2 range>

82284 Each <range> field shall be of the form:

82285 "%1d" , <beginning line number>

82286 if the range contains exactly one line, and:

82287 "%1d,%1d" , <beginning line number> , <number of lines>

82288 otherwise. If a range is empty, its beginning line number shall be the number of the line just
82289 before the range, or 0 if the empty range starts the file.

82290 Next, the affected lines along with lines of context shall be written. Each non-empty unaffected
82291 line shall be written in the following format:

82292 "Δ%s" , <unaffected_line>

82293 where the contents of the unaffected line shall be taken from *file1*. It is implementation-defined
82294 whether an empty unaffected line is written as an empty line or a line containing a single
82295 <space> character. This line also represents the same line of *file2*, even though *file2*'s line may
82296 contain different contents due to the **-b**. Deleted lines shall be written as:

82297 "-%s" , <deleted_line>

82298 Added lines shall be written as:

82299 "+%s" , <added_line>

82300 The order of lines written shall be the same as that of the corresponding file. A deleted line shall
82301 never be written immediately after an added line.

82302 If **-U n** is specified, the output shall contain no more than *n* consecutive unaffected lines; and if
82303 the output contains an affected line and this line is adjacent to up to *n* consecutive unaffected
82304 lines in the corresponding file, the output shall contain these unaffected lines. **-u** shall act like
82305 **-U3**.

82306 **STDERR**

82307 The standard error shall be used only for diagnostic messages.

82308 **OUTPUT FILES**

82309 None.

82310 **EXTENDED DESCRIPTION**

82311 None.

82312 **EXIT STATUS**

82313 The following exit values shall be returned:

82314 0 No differences were found.

82315 1 Differences were found.

82316 >1 An error occurred.

82317 **CONSEQUENCES OF ERRORS**

82318 Default.

82319
82320
82321
82322
82323

82324
82325
82326
82327

82328

82329

82330
82331
82332
82333
82334
82335
82336

82337
82338
82339

82340
82341
82342
82343
82344
82345

82346
82347
82348
82349

82350
82351
82352
82353
82354
82355
82356

82357
82358
82359
82360
82361
82362
82363
82364
82365
82366
82367
82368

APPLICATION USAGE

If lines at the end of a file are changed and other lines are added, *diff* output may show this as a delete and add, as a change, or as a change and add; *diff* is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

EXAMPLES

If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**, the command:

```
diff -r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff -r dir1/x/date.out dir2/x/date.out
1c1
< Mon Jul  2 13:12:16 PDT 1990
---
> Tue Jun 19 21:41:39 PDT 1990
```

RATIONALE

The **-h** option was omitted because it was insufficiently specified and does not add to applications portability.

Historical implementations employ algorithms that do not always produce a minimum list of differences; the current language about making every effort is the best this volume of POSIX.1-200x can do, as there is no metric that could be employed to judge the quality of implementations against any and all file contents. The statement “This list should be minimal” clearly implies that implementations are not expected to provide the following output when comparing two 100-line files that differ in only one character on a single line:

```
1,100c1,100
all 100 lines from file1 preceded with "< "
---
all 100 lines from file2 preceded with "> "
```

The “Only in” messages required when the **-r** option is specified are not used by most historical implementations if the **-e** option is also specified. It is required here because it provides useful information that must be provided to update a target directory hierarchy to match a source hierarchy. The “Common subdirectories” messages are written by System V and 4.3 BSD when the **-r** option is specified. They are allowed here but are not required because they are reporting on something that is the same, not reporting a difference, and are not needed to update a target hierarchy.

The **-c** option, which writes output in a format using lines of context, has been included. The format is useful for a variety of reasons, among them being much improved readability and the ability to understand difference changes when the target file has line numbers that differ from another similar, but slightly different, copy. The *patch* utility is most valuable when working with difference listings using a context format. The BSD version of **-c** takes an optional argument specifying the amount of context. Rather than overloading **-c** and breaking the Utility Syntax Guidelines for *diff*, the standard developers decided to add a separate option for specifying a context diff with a specified amount of context (**-C**). Also, the format for context *diffs* was extended slightly in 4.3 BSD to allow multiple changes that are within context lines from each other to be merged together. The output format contains an additional four asterisks after the range of affected lines in the first filename. This was to provide a flag for old programs (like old versions of *patch*) that only understand the old context format. The version of context

82369 described here does not require that multiple changes within context lines be merged, but it does
 82370 not prohibit it either. The extension is upwards-compatible, so any vendors that wish to retain
 82371 the old version of *diff* can do so by adding the extra four asterisks (that is, utilities that currently
 82372 use *diff* and understand the new merged format will also understand the old unmerged format,
 82373 but not *vice versa*).

82374 The **-u** and **-U** options of GNU *diff* have been included. Their output format, designed by
 82375 Wayne Davison, takes up less space than **-c** and **-C** format, and in many cases is easier to read.
 82376 The format's timestamps do not vary by locale, so *LC_TIME* does not affect it. The format's line
 82377 numbers are rendered with the `%1d` format, not `%d`, because the file format notation rules would
 82378 allow extra `<blank>`s to appear around the numbers.

82379 The substitute command was added as an additional format for the **-e** option. This was added
 82380 to provide implementations with a way to fix the classic "dot alone on a line" bug present in
 82381 many versions of *diff*. Since many implementations have fixed this bug, the standard developers
 82382 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing
 82383 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then
 82384 terminate the append command with a period, and then use the substitute command to convert
 82385 the two periods into one period.

82386 The BSD-derived **-r** option was added to provide a mechanism for using *diff* to compare two file
 82387 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not
 82388 easily reproducible with the *find* utility.

82389 The requirement that *diff* not compare files in some circumstances, even though they have the
 82390 same name, is based on the actual output of historical implementations. The specified behavior
 82391 precludes the problems arising from running into FIFOs and other files that would cause *diff* to
 82392 hang waiting for input with no indication to the user that *diff* was hung. An earlier version of
 82393 this standard specified the output format more precisely, but in practice this requirement was
 82394 widely ignored and the benefit of standardization seemed small, so it is now unspecified. In
 82395 most common usage, *diff -r* should indicate differences in the file hierarchies, not the difference
 82396 of contents of devices pointed to by the hierarchies.

82397 Many early implementations of *diff* require seekable files. Since the System Interfaces volume of
 82398 POSIX.1-200x supports named pipes, the standard developers decided that such a restriction
 82399 was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

82400 No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in
 82401 that it prints out all of the differences at the current level, including the statements about all
 82402 common subdirectories before recursing into those subdirectories.

82403 The message:

82404 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

82405 does not vary by locale because it is the representation of a command, not an English sentence.

82406 FUTURE DIRECTIONS

82407 None.

82408 SEE ALSO

82409 *cmp*, *comm*, *ed*, *find*

82410 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

82411 CHANGE HISTORY

82412 First released in Issue 2.

82413	Issue 5	
82414		The FUTURE DIRECTIONS section is added.
82415	Issue 6	
82416		The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:
82417		
82418		<ul style="list-style-type: none"> • The <code>-f</code> option is added.
82419		The output format for <code>-c</code> or <code>-C</code> format is changed to align with changes to the IEEE P1003.2b draft standard resulting from IEEE PASC Interpretation 1003.2 #71.
82420		
82421		The normative text is reworded to avoid use of the term “must” for application requirements.
82422		IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/20 is applied, changing the STDOUT section. This changes the specification of <i>diff</i> <code>-c</code> so that it agrees with existing practice when contexts contain zero lines or one line.
82423		
82424		
82425	Issue 7	
82426		Austin Group Interpretation 1003.1-2001 #115 and #114 are applied. +
82427		SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
82428		SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the <code>-u</code> option. -

DRAFT

82429 **NAME**
 82430 `dirname` — return the directory portion of a pathname

82431 **SYNOPSIS**
 82432 `dirname string`

82433 **DESCRIPTION**
 82434 The *string* operand shall be treated as a pathname, as defined in XBD [Section 3.265](#) (on page 70).
 82435 The string *string* shall be converted to the name of the directory containing the filename
 82436 corresponding to the last pathname component in *string*, performing actions equivalent to the
 82437 following steps in order:

- 82438 1. If *string* is `//`, skip steps 2 to 5.
- 82439 2. If *string* consists entirely of slash characters, *string* shall be set to a single slash character.
 82440 In this case, skip steps 3 to 8.
- 82441 3. If there are any trailing slash characters in *string*, they shall be removed.
- 82442 4. If there are no slash characters remaining in *string*, *string* shall be set to a single period
 82443 character. In this case, skip steps 5 to 8.
- 82444 5. If there are any trailing non-slash characters in *string*, they shall be removed.
- 82445 6. If the remaining *string* is `//`, it is implementation-defined whether steps 7 and 8 are
 82446 skipped or processed.
- 82447 7. If there are any trailing slash characters in *string*, they shall be removed.
- 82448 8. If the remaining *string* is empty, *string* shall be set to a single slash character.

82449 The resulting string shall be written to standard output.

82450 **OPTIONS**
 82451 None.

82452 **OPERANDS**
 82453 The following operand shall be supported:

82454 *string* A string.

82455 **STDIN**
 82456 Not used.

82457 **INPUT FILES**
 82458 None.

82459 **ENVIRONMENT VARIABLES**
 82460 The following environment variables shall affect the execution of *dirname*:

- | | | |
|-------|-----------------|--|
| 82461 | <i>LANG</i> | Provide a default value for the internationalization variables that are unset or null.
(See XBD Section 8.2 (on page 160) for the precedence of internationalization
variables used to determine the values of locale categories.) |
| 82462 | | |
| 82463 | | |
| 82464 | <i>LC_ALL</i> | If set to a non-empty string value, override the values of all the other
internationalization variables. |
| 82465 | | |
| 82466 | <i>LC_CTYPE</i> | Determine the locale for the interpretation of sequences of bytes of text data as
characters (for example, single-byte as opposed to multi-byte characters in
arguments). |
| 82467 | | |
| 82468 | | |

dirname

Utilities

82469 *LC_MESSAGES*
 82470 Determine the locale that should be used to affect the format and contents of
 82471 diagnostic messages written to standard error.

82472 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

82473
 82474 Default.

STDOUT

82475 The *dirname* utility shall write a line to the standard output in the following format:

82476 "%s\n", *<resulting string>*
 82477

STDERR

82478 The standard error shall be used only for diagnostic messages.
 82479

OUTPUT FILES

82480 None.
 82481

EXTENDED DESCRIPTION

82482 None.
 82483

EXIT STATUS

82484 The following exit values shall be returned:
 82485

82486 0 Successful completion.

82487 >0 An error occurred.

CONSEQUENCES OF ERRORS

82488 Default.
 82489

APPLICATION USAGE

82490 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
 82491 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the
 82492 beginning of a pathname unless they can ensure that there are more or less than two or are
 82493 prepared to deal with the implementation-defined consequences.
 82494

EXAMPLES

Command	Results
<i>dirname /</i>	/
<i>dirname //</i>	/ or //
<i>dirname /a/b/</i>	/a
<i>dirname //a//b//</i>	//a
<i>dirname</i>	Unspecified
<i>dirname a</i>	.(\$? = 0)
<i>dirname ""</i>	.(\$? = 0)
<i>dirname /a</i>	/
<i>dirname /a/b</i>	/a
<i>dirname a/b</i>	a

RATIONALE

82507 The *dirname* utility originated in System III. It has evolved through the System V releases to a
 82508 version that matches the requirements specified in this description in System V Release 3. 4.3
 82509 BSD and earlier versions did not include *dirname*.
 82510

82511 The behaviors of *basename* and *dirname* in this volume of POSIX.1-200x have been coordinated so
 82512 that when *string* is a valid pathname:

82513 $\$(basename \textit{string})$

82514 would be a valid filename for the file in the directory:

82515 `$(dirname "string")`

82516 This would not work for the versions of these utilities in early proposals due to the way
82517 processing of trailing slashes was specified. Consideration was given to leaving processing
82518 unspecified if there were trailing slashes, but this cannot be done; XBD [Section 3.265](#) (on page
82519 70) allows trailing slashes. The *basename* and *dirname* utilities have to specify consistent handling
82520 for all valid pathnames.

82521 FUTURE DIRECTIONS

82522 None.

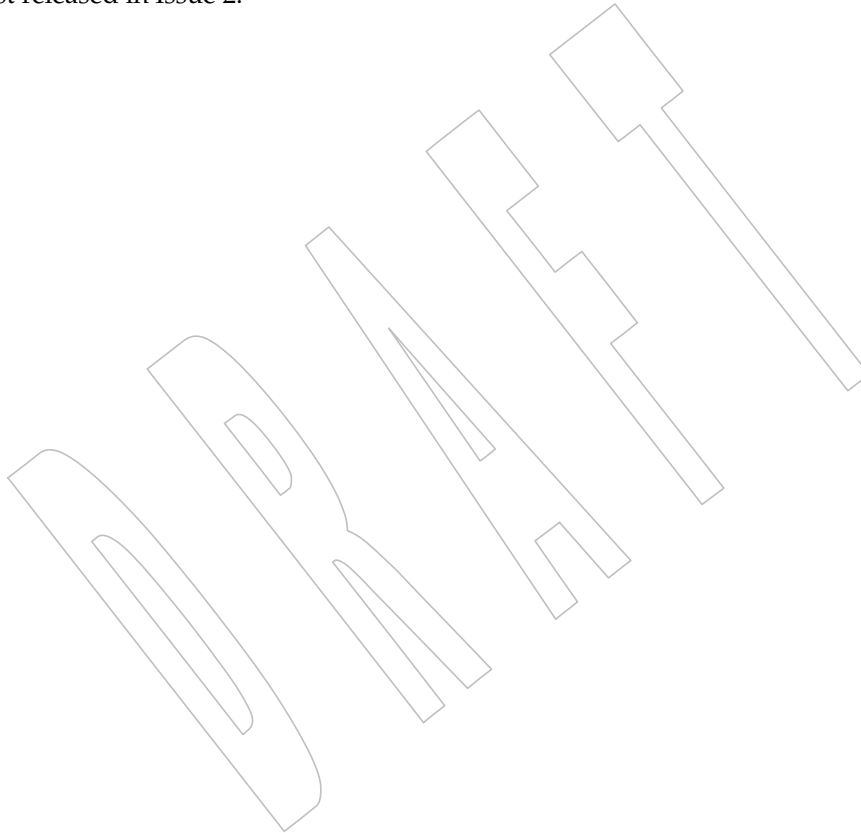
82523 SEE ALSO

82524 [Section 2.5](#) (on page 2249), *basename*

82525 XBD [Section 3.265](#) (on page 70), [Chapter 8](#) (on page 159)

82526 CHANGE HISTORY

82527 First released in Issue 2.



82528 **NAME**
 82529 du — estimate file space usage

82530 **SYNOPSIS**
 82531 du [-a|-s] [-kx] [-H|-L] [*file*...]

82532 **DESCRIPTION**
 82533 By default, the *du* utility shall write to standard output the size of the file space allocated to, and
 82534 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the
 82535 specified files. By default, when a symbolic link is encountered on the command line or in the
 82536 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the
 82537 link), and shall not follow the link to another portion of the file hierarchy. The size of the file
 82538 space allocated to a file of type directory shall be defined as the sum total of space allocated to
 82539 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

82540 When *du* cannot *stat()* files or *stat()* or read directories, it shall report an error condition and the
 82541 final exit status is affected. Files with multiple links shall be counted and written for only one
 82542 entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall
 82543 be written in 512-byte units, rounded up to the next 512-byte unit.

82544 **OPTIONS**
 82545 The *du* utility shall conform to XBD Section 12.2 (on page 201).

82546 The following options shall be supported:

82547 **-a** In addition to the default output, report the size of each file not of type directory in
 82548 the file hierarchy rooted in the specified file. Regardless of the presence of the **-a**
 82549 option, non-directories given as *file* operands shall always be listed.

82550 **-H** If a symbolic link is specified on the command line, *du* shall count the size of the
 82551 file or file hierarchy referenced by the link.

82552 **-k** Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.

82553 **-L** If a symbolic link is specified on the command line or encountered during the
 82554 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy
 82555 referenced by the link.

82556 **-s** Instead of the default output, report only the total sum for each of the specified
 82557 files.

82558 **-x** When evaluating file sizes, evaluate only those files that have the same device as
 82559 the file specified by the *file* operand.

82560 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
 82561 an error. The last option specified shall determine the behavior of the utility.

82562 **OPERANDS**
 82563 The following operand shall be supported:

82564 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current
 82565 directory shall be used.

82566 **STDIN**
 82567 Not used.

82568 **INPUT FILES**

82569 None.

82570 **ENVIRONMENT VARIABLES**82571 The following environment variables shall affect the execution of *du*:

82572 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 82573 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 82574 variables used to determine the values of locale categories.)

82575 **LC_ALL** If set to a non-empty string value, override the values of all the other
 82576 internationalization variables.

82577 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 82578 characters (for example, single-byte as opposed to multi-byte characters in
 82579 arguments).

82580 **LC_MESSAGES**

82581 Determine the locale that should be used to affect the format and contents of
 82582 diagnostic messages written to standard error.

82583 **XSI NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82584 **ASYNCHRONOUS EVENTS**

82585 Default.

82586 **STDOUT**

82587 The output from *du* shall consist of the amount of space allocated to a file and the name of the
 82588 file, in the following format:

82589 "%d %s\n", <size>, <pathname>

82590 **STDERR**

82591 The standard error shall be used only for diagnostic messages.

82592 **OUTPUT FILES**

82593 None.

82594 **EXTENDED DESCRIPTION**

82595 None.

82596 **EXIT STATUS**

82597 The following exit values shall be returned:

82598 0 Successful completion.

82599 >0 An error occurred.

82600 **CONSEQUENCES OF ERRORS**

82601 Default.

82602 **APPLICATION USAGE**

82603 None.

82604 **EXAMPLES**

82605 None.

82606 **RATIONALE**

82607 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
 82608 utilities in this volume of POSIX.1-200x. This does not mandate that the file system itself be
 82609 based on 512-byte blocks. The **-k** option was added as a compromise measure. It was agreed by
 82610 the standard developers that 512 bytes was the best default unit because of its complete
 82611 historical consistency on System V (*versus* the mixed 512/1024-byte usage on BSD systems), and
 82612 that a **-k** option to switch to 1024-byte units was a good compromise. Users who prefer the

- 82613 1 024-byte quantity can easily alias *du* to *du -k* without breaking the many historical scripts
82614 relying on the 512-byte units.
- 82615 The **-b** option was added to an early proposal to provide a resolution to the situation where
82616 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-
82617 defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for
82618 BSD systems.) However, **-b** was later deleted, since the default was eventually decided as
82619 512-byte units.
- 82620 Historical file systems provided no way to obtain exact figures for the space allocation given to
82621 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*
82622 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is
82623 space used by the file system in the storage of the file, but that need not be counted in the space
82624 allocated to the file. A *sparse* file is one in which an *lseek()* call has been made to a position
82625 beyond the end of the file and data has subsequently been written at that point. A file system
82626 need not allocate all the intervening zero-filled blocks to such a file. It is up to the
82627 implementation to define exactly how accurate its methods are.
- 82628 The **-a** and **-s** options were mutually-exclusive in the original version of *du*. The POSIX Shell
82629 and Utilities description is implied by the language in the SVID where **-s** is described as causing
82630 “only the grand total” to be reported. Some systems may produce output for **-sa**, but a Strictly
82631 Conforming POSIX Shell and Utilities Application cannot use that combination.
- 82632 The **-a** and **-s** options were adopted from the SVID except that the System V behavior of not
82633 listing non-directories explicitly given as operands, unless the **-a** option is specified, was
82634 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default
82635 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no
82636 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and
82637 Utilities default behavior shall be to produce such messages. These messages can be turned off
82638 with shell redirection to achieve the System V behavior.
- 82639 The **-x** option is historical practice on recent BSD systems. It has been adopted by this volume of
82640 POSIX.1-200x because there was no other historical method of limiting the *du* search to a single
82641 file hierarchy. This limitation of the search is necessary to make it possible to obtain file space
82642 usage information about a file system on which other file systems are mounted, without having
82643 to resort to a lengthy *find* and *awk* script.
- 82644 **FUTURE DIRECTIONS**
- 82645 None.
- 82646 **SEE ALSO**
- 82647 *ls*
- 82648 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)
- 82649 XSH [fstatat\(\)](#)
- 82650 **CHANGE HISTORY**
- 82651 First released in Issue 2.
- 82652 **Issue 6**
- 82653 This utility is marked as part of the User Portability Utilities option.
- 82654 The APPLICATION USAGE section is added.
- 82655 The obsolescent **-r** option is removed.
- 82656 The Open Group Corrigendum U025/3 is applied. The *du* utility is reinstated, as it had
82657 incorrectly been marked LEGACY in Issue 5.
- 82658 The **-H** and **-L** options for symbolic links are added as described in the IEEE P1003.2b draft
82659 standard.

82660

Issue 7

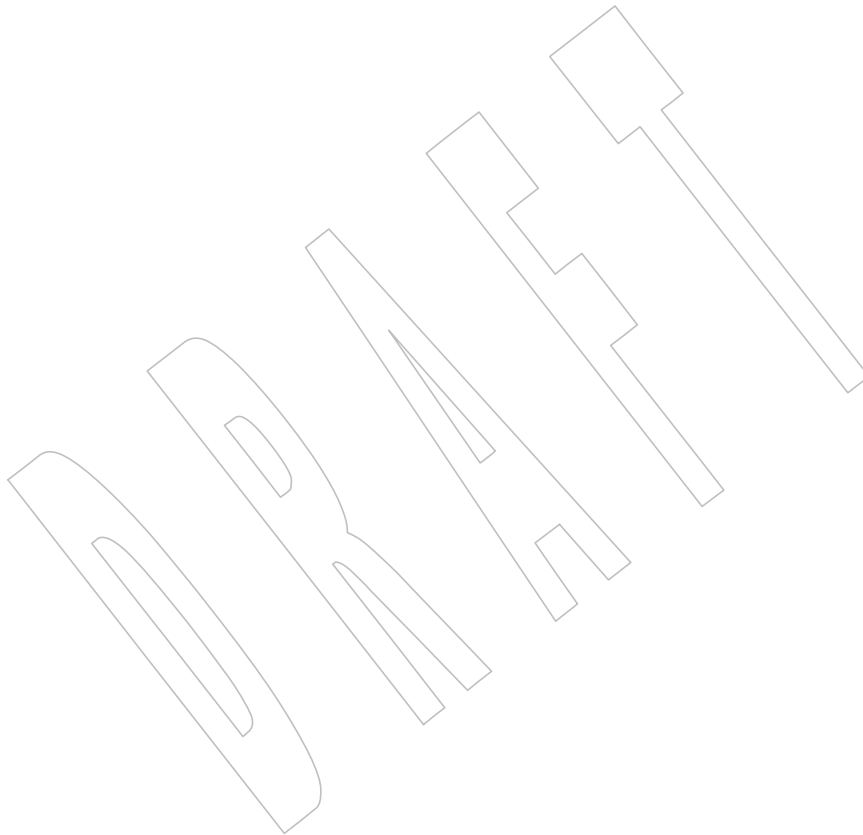
82661

The *du* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

82662

82663

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



82664 **NAME**
 82665 echo — write arguments to standard output

82666 **SYNOPSIS**
 82667 echo [*string...*]

82668 **DESCRIPTION**
 82669 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are
 82670 no arguments, only the <newline> is written.

82671 **OPTIONS**
 82672 The *echo* utility shall not recognize the "--" argument in the manner specified by Guideline 10
 82673 of XBD [Section 12.2](#) (on page 201); "--" shall be recognized as a string operand.
 82674 Implementations shall not support any options.

82675 **OPERANDS**
 82676 The following operands shall be supported:

82677 *string* A string to be written to standard output. If the first operand is *-n*, or if any of the
 82678 operands contain a backslash ('**') character, the results are implementation-
 82679 defined.

82680 XSI On XSI-conformant systems, if the first operand is *-n*, it shall be treated as a string,
 82681 not an option. The following character sequences shall be recognized on XSI-
 82682 conformant systems within any of the arguments:

82683	<i>\a</i>	Write an <alert>.
82684	<i>\b</i>	Write a <backspace>.
82685	<i>\c</i>	Suppress the <newline> that otherwise follows the final argument in the output. All characters following the ' <i>\c</i> ' in the arguments shall be ignored.
82688	<i>\f</i>	Write a <form-feed>.
82689	<i>\n</i>	Write a <newline>.
82690	<i>\r</i>	Write a <carriage-return>.
82691	<i>\t</i>	Write a <tab>.
82692	<i>\v</i>	Write a <vertical-tab>.
82693	<i>\\</i>	Write a backslash character.
82694	<i>\0num</i>	Write an 8-bit value that is the zero, one, two, or three-digit octal number <i>num</i> .
82695		

82696 **STDIN**
 82697 Not used.

82698 **INPUT FILES**
 82699 None.

82700 **ENVIRONMENT VARIABLES**
 82701 The following environment variables shall affect the execution of *echo*:

82702 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 82703 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 82704 variables used to determine the values of locale categories.)

82705 *LC_ALL* If set to a non-empty string value, override the values of all the other
 82706 internationalization variables.

82707 XSI *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 82708 characters (for example, single-byte as opposed to multi-byte characters in
 82709 arguments).

82710 *LC_MESSAGES*
 82711 Determine the locale that should be used to affect the format and contents of
 82712 diagnostic messages written to standard error.

82713 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82714 **ASYNCHRONOUS EVENTS**

82715 Default.

82716 **STDOUT**

82717 The *echo* utility arguments shall be separated by single <space>s and a <newline> shall follow
 82718 XSI the last argument. Output transformations shall occur based on the escape sequences in the
 82719 input. See the OPERANDS section.

82720 **STDERR**

82721 The standard error shall be used only for diagnostic messages.

82722 **OUTPUT FILES**

82723 None.

82724 **EXTENDED DESCRIPTION**

82725 None.

82726 **EXIT STATUS**

82727 The following exit values shall be returned:

82728 0 Successful completion.

82729 >0 An error occurred.

82730 **CONSEQUENCES OF ERRORS**

82731 Default.

82732 **APPLICATION USAGE**

82733 It is not possible to use *echo* portably across all POSIX systems unless both *-n* (as the first
 82734 argument) and escape sequences are omitted.

82735 The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo*
 82736 utility as follows (assuming that *IFS* has its standard value or is unset):

- 82737 • The historic System V *echo* and the requirements on XSI implementations in this volume of
 82738 POSIX.1-200x are equivalent to:

```
82739            printf "%b\n" "$*"

```

- 82740 • The BSD *echo* is equivalent to:

```
82741            if [ "$1" = "X-n" ]

```

```
82742            then

```

```
82743                shift

```

```
82744                printf "%s" "$*"

```

```
82745            else

```

```
82746                printf "%s\n" "$*"

```

82747 f i
82748 New applications are encouraged to use *printf* instead of *echo*.

EXAMPLES

82749 None.
82750

RATIONALE

82751 The *echo* utility has not been made obsolescent because of its extremely widespread use in
82752 historical applications. Conforming applications that wish to do prompting without <newline>s
82753 or that could possibly be expecting to echo a *-n*, should use the *printf* utility derived from the
82754 Ninth Edition system.
82755

82756 As specified, *echo* writes its arguments in the simplest of ways. The two different historical
82757 versions of *echo* vary in fatally incompatible ways.

82758 The BSD *echo* checks the first argument for the string *-n* which causes it to suppress the
82759 <newline> that would otherwise follow the final argument in the output.

82760 The System V *echo* does not support any options, but allows escape sequences within its
82761 operands, as described for XSI implementations in the OPERANDS section.

82762 The *echo* utility does not support Utility Syntax Guideline 10 because historical applications
82763 depend on *echo* to echo *all* of its arguments, except for the *-n* option in the BSD version.

FUTURE DIRECTIONS

82764 None.
82765

SEE ALSO

82766 *printf*
82767
82768 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

CHANGE HISTORY

82769 First released in Issue 2.
82770

Issue 5

82771 In the OPTIONS section, the last sentence is changed to indicate that implementations “do not”
82772 support any options; in the previous issue this said “need not”.
82773

Issue 6

82774 The following new requirements on POSIX implementations derive from alignment with the
82775 Single UNIX Specification:
82776

- 82777 • A set of character sequences is defined as *string* operands.
- 82778 • *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- 82779 • In the OPTIONS section, implementations shall not support any options.

82780 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/21 is applied, so that the *echo* utility can
82781 accommodate historical BSD behavior.

Issue 7

82782 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
82783

82784 **NAME**

82785 ed — edit text

82786 **SYNOPSIS**82787 ed [-p *string*] [-s] [*file*]82788 **DESCRIPTION**

82789 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*. In
 82790 command mode the input characters shall be interpreted as commands, and in input mode they
 82791 shall be interpreted as text. See the EXTENDED DESCRIPTION section.

82792 If an operand is '-', the results are unspecified.

82793 **OPTIONS**

82794 The *ed* utility shall conform to XBD Section 12.2 (on page 201), except for the unspecified usage
 82795 of '-'.

82796 The following options shall be supported:

82797 **-p *string*** Use *string* as the prompt string when in command mode. By default, there shall be
 82798 no prompt string.

82799 **-s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the '!'
 82800 prompt after a *!command*.

82801 **OPERANDS**

82802 The following operand shall be supported:

82803 ***file*** If the *file* argument is given, *ed* shall simulate an **e** command on the file named by
 82804 the pathname, *file*, before accepting commands from the standard input.

82805 **STDIN**

82806 The standard input shall be a text file consisting of commands, as described in the EXTENDED
 82807 DESCRIPTION section.

82808 **INPUT FILES**

82809 The input files shall be text files.

82810 **ENVIRONMENT VARIABLES**82811 The following environment variables shall affect the execution of *ed*:

82812 **HOME** Determine the pathname of the user's home directory.

82813 **LANG** Provide a default value for the internationalization variables that are unset or null.
 82814 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 82815 variables used to determine the values of locale categories.)

82816 **LC_ALL** If set to a non-empty string value, override the values of all the other
 82817 internationalization variables.

82818 **LC_COLLATE**

82819 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 82820 character collating elements within regular expressions.

82821 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 82822 characters (for example, single-byte as opposed to multi-byte characters in
 82823 arguments and input files) and the behavior of character classes within regular
 82824 expressions.

82825 *LC_MESSAGES*
 82826 Determine the locale that should be used to affect the format and contents of
 82827 diagnostic messages written to standard error and informative messages written to
 82828 standard output.

82829 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

82830 ASYNCHRONOUS EVENTS

82831 The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS
 82832 section in [Section 1.4](#), on page 2235) with the following exceptions:

82833 SIGINT The *ed* utility shall interrupt its current activity, write the string "?\n" to standard
 82834 output, and return to command mode (see the EXTENDED DESCRIPTION
 82835 section).

82836 SIGHUP If the buffer is not empty and has changed since the last write, the *ed* utility shall
 82837 attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the
 82838 current directory shall be used; if that fails, the file named **ed.hup** in the directory
 82839 named by the *HOME* environment variable shall be used. In any case, the *ed* utility
 82840 shall exit without writing the file to the currently remembered pathname and
 82841 without returning to command mode.

82842 SIGQUIT The *ed* utility shall ignore this event.

82843 STDOUT

82844 Various editing commands and the prompting feature (see **-p**) write to standard output, as
 82845 described in the EXTENDED DESCRIPTION section.

82846 STDERR

82847 The standard error shall be used only for diagnostic messages.

82848 OUTPUT FILES

82849 The output files shall be text files whose formats are dependent on the editing commands given.

82850 EXTENDED DESCRIPTION

82851 The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have
 82852 no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.

82853 Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a
 82854 single-character *command*, possibly followed by parameters to that command. These addresses
 82855 specify one or more lines in the buffer. Every command that requires addresses has default
 82856 addresses, so that the addresses very often can be omitted. If the **-p** option is specified, the
 82857 prompt string shall be written to standard output before each command is read.

82858 In general, only one command can appear on a line. Certain commands allow text to be input.
 82859 This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be
 82860 in *input mode*. In this mode, no commands shall be recognized; all input is merely collected.
 82861 Input mode is terminated by entering a line consisting of two characters: a period ('.')
 82862 followed by a <newline>. This line is not considered part of the input text.

82863 Regular Expressions in ed

82864 The *ed* utility shall support basic regular expressions, as described in XBD [Section 9.3](#) (on page
 82865 169). Since regular expressions in *ed* are always matched against single lines (excluding the
 82866 terminating <newline>s), never against any larger section of text, there is no way for a regular
 82867 expression to match a <newline>.

82868 A null RE shall be equivalent to the last RE encountered.

82869 Regular expressions are used in addresses to specify lines, and in some commands (for example,
 82870 the **s** substitute command) to specify portions of a line to be substituted.

82871

Addresses in ed

82872

Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. If the edit buffer is not empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

82873

82874

Addresses shall be constructed as follows:

82876

1. The period character (' . ') shall address the current line.

82877

2. The dollar sign character (' \$ ') shall address the last line of the edit buffer.

82878

3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

82879

4. The apostrophe-x character pair (" ' x ") shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. It shall be an error if the character has not been set to mark a line or if the line that was marked is not currently present in the edit buffer.

82880

82881

82882

82883

5. A BRE enclosed by slash characters (' / ') shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall address the next line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second slash can be omitted at the end of a command line. Within the BRE, a backslash-slash pair (" \ / ") shall represent a literal slash instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched.

82884

82885

82886

82887

82888

82889

82900

82901

82902

6. A BRE enclosed by question-mark characters (' ? ') shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of question-mark characters (" ?? ") shall address the previous line for which the line excluding the terminating <newline> matches the last BRE encountered. In addition, the second question-mark can be omitted at the end of a command line. Within the BRE, a backslash-question-mark pair (" \ ? ") shall represent a literal question mark instead of the BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched.

82893

82894

82895

82896

82897

82898

82899

82900

82901

82902

7. A plus-sign (' + ') or hyphen character (' - ') followed by a decimal number shall address the current line plus or minus the number. A plus-sign or hyphen character not followed by a decimal number shall address the current line plus or minus 1.

82903

82904

82905

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

82906

82907

82908

- A plus-sign or hyphen character followed by a decimal number shall add or subtract, respectively, the indicated number of lines to or from the address. A plus-sign or hyphen character not followed by a decimal number shall add or subtract 1 to or from the address.

82909

82910

- A decimal number shall add the indicated number of lines to the address.

82911

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a matching line.

82912

82913

82914

82915

82916

Commands accept zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more

82917

82918 than the required number of addresses are provided to a command, the addresses specified first
82919 shall be evaluated and then discarded until the maximum number of valid addresses remain, for
82920 the specified command.

82921 Addresses shall be separated from each other by a comma (',') or semicolon character (';').
82922 In the case of a semicolon separator, the current line ('.') shall be set to the first address, and
82923 only then will the second address be calculated. This feature can be used to determine the
82924 starting line for forwards and backwards searches; see rules 5. and 6.

82925 Addresses can be omitted on either side of the comma or semicolon separator, in which case the
82926 resulting address pairs shall be as follows:

Specified	Resulting
,	1 , \$
, addr	1 , addr
addr ,	addr , addr
;	. ; \$
; addr	. ; addr
addr ;	addr ; addr

82934 Any <blank>s included between addresses, address separators, or address offsets shall be
82935 ignored.

82936 Commands in ed

82937 In the following list of *ed* commands, the default addresses are shown in parentheses. The
82938 number of addresses shown in the default shall be the number expected by the command. The
82939 parentheses are not part of the address; they show that the given addresses are the default.

82940 It is generally invalid for more than one command to appear on a line. However, any command
82941 (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for
82942 the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be
82943 written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used
82944 with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but
82945 it is unspecified whether the suffix writes the current line again in the requested format or
82946 whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l**
82947 suffix) shall either write just the current line or write it twice—once as specified for **p** and once
82948 as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

82949 Each address component can be preceded by zero or more <blank>s. The command letter can
82950 be preceded by zero or more <blank>s. If a suffix letter (**l**, **n**, or **p**) is given, the application shall
82951 ensure that it immediately follows the command.

82952 The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the
82953 command letter by one or more <blank>s.

82954 If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed*
82955 shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.
82956 The *ed* utility shall write the string:

82957 " ?\n "

82958 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to
82959 standard output and shall continue in command mode with the current line number unchanged.
82960 If the **e** or **q** command is repeated with no intervening command, it shall take effect.

82961 If a terminal disconnect (see XBD Chapter 11 (on page 185), Modem Disconnect and Closing a
82962 Device Terminal), is detected:

- 82963
- If accompanied by a SIGHUP signal, the *ed* utility shall operate as described in the ASYNCHRONOUS EVENTS section for a SIGHUP signal.
- 82964
- 82965
- If not accompanied by a SIGHUP signal, the *ed* utility shall act as if an end-of-file had been detected on standard input.
- 82966

82967 If an end-of-file is detected on standard input:

- 82968
- If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode. It is unspecified if any partially entered lines (that is, input text without a terminating <newline>) are discarded from the input text.
- 82969
- 82970
- If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.
- 82971

82972 If the closing delimiter of an RE or of a replacement string (for example, ' / ') in a **g**, **G**, **s**, **v**, or **V**

82973 command would be the last character before a <newline>, that delimiter can be omitted, in

82974 which case the addressed line shall be written. For example, the following pairs of commands

82975 are equivalent:

82976 `s/s1/s2` `s/s1/s2/p`

82977 `g/s1` `g/s1/p`

82978 `?s1` `?s1?`

82979 If an invalid command is entered, *ed* shall write the string:

82980 `"?\n"`

82981 (followed by an explanatory message if *help mode* has been enabled via the **H** command) to

82982 standard output and shall continue in command mode with the current line number unchanged.

82983 Append Command

82984 *Synopsis:* `(.)a`

82985 `<text>`

82986 `.`

82987 The **a** command shall read the given text and append it after the addressed line; the current line

82988 number shall become the address of the last inserted line or, if there were none, the addressed

82989 line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at

82990 the beginning of the buffer.

82991 Change Command

82992 *Synopsis:* `(.,.)c`

82993 `<text>`

82994 `.`

82995 The **c** command shall delete the addressed lines, then accept input text that replaces these lines;

82996 the current line shall be set to the address of the last line input; or, if there were none, at the line

82997 after the last line deleted; if the lines deleted were originally at the end of the buffer, the current

82998 line number shall be set to the address of the new last line; if no lines remain in the buffer, the

82999 current line number shall be set to zero. Address 0 shall be valid for this command; it shall be

83000 interpreted as if address 1 were specified.

83001

Delete Command

83002

Synopsis: (. , .)d

83003

The **d** command shall delete the addressed lines from the buffer. The address of the line after the last line deleted shall become the current line number; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero.

83004

83005

83006

83007

Edit Command

83008

Synopsis: e [*file*]

83009

The **e** command shall delete the entire contents of the buffer and then read in the file named by the pathname *file*. The current line number shall be set to the address of the last line of the buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **f** command). The number of bytes read shall be written to standard output, unless the **-s** option was specified, in the following format:

83010

83011

83012

83013

83014

"%d\n", <number of bytes read>

83015

The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**, and **w** commands. If *file* is replaced by **'!'**, the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current *file*. All marks shall be discarded upon the completion of a successful **e** command. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

83016

83017

83018

83019

83020

83021

Edit Without Checking Command

83022

Synopsis: E [*file*]

83023

The **E** command shall possess all properties and restrictions of the **e** command except that the editor shall not check to see whether any changes have been made to the buffer since the last **w** command.

83024

83025

83026

Filename Command

83027

Synopsis: f [*file*]

83028

If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether the name is changed or not, it shall then write the (possibly new) currently remembered pathname to the standard output in the following format:

83029

83030

83031

"%s\n", <pathname>

83032

The current line number shall be unchanged.

83033

Global Command

83034

Synopsis: (1,\$)g/RE/command list

83035

In the **g** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given RE. Then, going sequentially from the beginning of the file to the end of the file, the given *command list* shall be executed for each marked line, with the current line number set to the address of that line. Any line modified by the *command list* shall be unmarked. When the **g** command completes, the current line number shall have the value assigned by the last command in the *command list*. If there were no matching lines, the current line number shall not be changed. A single command or the first of a list of commands shall appear on the same line as the global command. All lines of a multi-line list except the last line shall be ended with a backslash preceding the terminating <newline>; the **a**, **i**, and **c**

83036

83037

83038

83039

83040

83041

83042

83043

83044 commands and associated input are permitted. The `'.'` terminating input mode can be omitted
 83045 if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the `p`
 83046 command. The use of the `g`, `G`, `v`, `V`, and `!` commands in the *command list* produces undefined
 83047 results. Any character other than `<space>` or `<newline>` can be used instead of a slash to delimit
 83048 the RE. Within the RE, the RE delimiter itself can be used as a literal character if it is preceded by
 83049 a backslash.

83050 Interactive Global Command

83051 *Synopsis:* `(1, $)G/RE/`

83052 In the `G` command, the first step shall be to mark every line for which the line excluding the
 83053 terminating `<newline>` matches the given RE. Then, for every such line, that line shall be
 83054 written, the current line number shall be set to the address of that line, and any one command
 83055 (other than one of the `a`, `c`, `i`, `g`, `G`, `v`, and `V` commands) shall be read and executed. A `<newline>`
 83056 shall act as a null command (causing no action to be taken on the current line); an `'&'` shall
 83057 cause the re-execution of the most recent non-null command executed within the current
 83058 invocation of `G`. Note that the commands input as part of the execution of the `G` command can
 83059 address and affect any lines in the buffer. Any line modified by the command shall be
 83060 unmarked. The final value of the current line number shall be the value set by the last command
 83061 successfully executed. (Note that the last command successfully executed shall be the `G`
 83062 command itself if a command fails or the null command is specified.) If there were no matching
 83063 lines, the current line number shall not be changed. The `G` command can be terminated by a
 83064 SIGINT signal. Any character other than `<space>` or `<newline>` can be used instead of a slash to
 83065 delimit the RE and the replacement. Within the RE, the RE delimiter itself can be used as a
 83066 literal character if it is preceded by a backslash.

83067 Help Command

83068 *Synopsis:* `h`

83069 The `h` command shall write a short message to standard output that explains the reason for the
 83070 most recent `'?'` notification. The current line number shall be unchanged.

83071 Help-Mode Command

83072 *Synopsis:* `H`

83073 The `H` command shall cause *ed* to enter a mode in which help messages (see the `h` command)
 83074 shall be written to standard output for all subsequent `'?'` notifications. The `H` command
 83075 alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on,
 83076 the `H` command also explains the previous `'?'` notification, if there was one. The current line
 83077 number shall be unchanged.

83078 Insert Command

83079 *Synopsis:* `(.)i`
 83080 `<text>`
 83081 `.`

83082 The `i` command shall insert the given text before the addressed line; the current line is set to the
 83083 last inserted line or, if there was none, to the addressed line. This command differs from the `a`
 83084 command only in the placement of the input text. Address 0 shall be valid for this command; it
 83085 shall be interpreted as if address 1 were specified.

83086

Join Command

83087

Synopsis: (. , . +1) j

83088

83089

83090

83091

The **j** command shall join contiguous lines by removing the appropriate <newline>s. If exactly one address is given, this command shall do nothing. If lines are joined, the current line number shall be set to the address of the joined line; otherwise, the current line number shall be unchanged.

83092

Mark Command

83093

Synopsis: (.) kx

83094

83095

83096

The **k** command shall mark the addressed line with name *x*, which the application shall ensure is a lowercase letter from the portable character set. The address " 'x'" shall then refer to this line; the current line number shall be unchanged.

83097

List Command

83098

Synopsis: (. , .) l

83099

83100

83101

83102

83103

83104

The **l** command shall write to standard output the addressed lines in a visually unambiguous form. The characters listed in XBD Table 5-1 (on page 108) (' \ ' , ' \ a ' , ' \ b ' , ' \ f ' , ' \ r ' , ' \ t ' , ' \ v ') shall be written as the corresponding escape sequence; the ' \ n ' in that table is not applicable. Non-printable characters not in the table shall be written as one three-digit octal number (with a preceding backslash character) for each byte in the character (most significant byte first).

83105

83106

83107

83108

83109

83110

Long lines shall be folded, with the point of folding indicated by <newline> preceded by a backslash; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a ' \$ ' , and ' \$ ' characters within the text shall be written with a preceding backslash. An **l** command can be appended to any other command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of the last line written.

83111

Move Command

83112

Synopsis: (. , .) maddress

83113

83114

83115

83116

The **m** command shall reposition the addressed lines after the line addressed by *address*. Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning of the buffer. It shall be an error if *address* falls within the range of moved lines. The current line number shall be set to the address of the last line moved.

83117

Number Command

83118

Synopsis: (. , .) n

83119

83120

83121

The **n** command shall write to standard output the addressed lines, preceding each line by its line number and a <tab>; the current line number shall be set to the address of the last line written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

83122 **Print Command**83123 *Synopsis:* (. , .)p

83124 The **p** command shall write to standard output the addressed lines; the current line number shall
 83125 be set to the address of the last line written. The **p** command can be appended to any command
 83126 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

83127 **Prompt Command**83128 *Synopsis:* P

83129 The **P** command shall cause *ed* to prompt with an asterisk ('*') (or *string*, if **-p** is specified) for
 83130 all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall
 83131 be initially on if the **-p** option is specified; otherwise, off. The current line number shall be
 83132 unchanged.

83133 **Quit Command**83134 *Synopsis:* q

83135 The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire
 83136 buffer was written, the user shall be warned, as described previously.

83137 **Quit Without Checking Command**83138 *Synopsis:* Q

83139 The **Q** command shall cause *ed* to exit without checking whether changes have been made in the
 83140 buffer since the last **w** command.

83141 **Read Command**83142 *Synopsis:* (\$)r [*file*]

83143 The **r** command shall read in the file named by the pathname *file* and append it after the
 83144 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be
 83145 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed
 83146 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file
 83147 to be read at the beginning of the buffer. If the read is successful, and **-s** was not specified, the
 83148 number of bytes read shall be written to standard output in the following format:

83149 "%d\n", <number of bytes read>

83150 The current line number shall be set to the address of the last line read in. If *file* is replaced by
 83151 '!', the rest of the line shall be taken to be a shell command line whose output is to be read.
 83152 Such a shell command line shall not be remembered as the current pathname.

83153 **Substitute Command**83154 *Synopsis:* (. , .)s/RE/replacement/flags

83155 The **s** command shall search each addressed line for an occurrence of the specified RE and
 83156 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the
 83157 following description of the **g** suffix. It is an error if the substitution fails on every addressed
 83158 line. Any character other than <space> or <newline> can be used instead of a slash to delimit the
 83159 RE and the replacement. Within the RE, the RE delimiter itself can be used as a literal character
 83160 if it is preceded by a backslash. The current line shall be set to the address of the last line on
 83161 which a substitution occurred.

83162 An ampersand ('&') appearing in the *replacement* shall be replaced by the string matching the
 83163 RE on the current line. The special meaning of '&' in this context can be suppressed by

83164 preceding it by backslash. As a more general feature, the characters ' $\backslash n$ ', where n is a digit,
 83165 shall be replaced by the text matched by the corresponding back-reference expression. If the
 83166 corresponding back-reference expression does not match, then the characters ' $\backslash n$ ' shall be
 83167 replaced by the empty string. When the character ' $\%$ ' is the only character in the *replacement*, the
 83168 *replacement* used in the most recent substitute command shall be used as the *replacement* in the
 83169 current substitute command; if there was no previous substitute command, the use of ' $\%$ ' in this
 83170 manner shall be an error. The ' $\%$ ' shall lose its special meaning when it is in a replacement
 83171 string of more than one character or is preceded by a backslash. For each backslash (' \backslash ')
 83172 encountered in scanning *replacement* from beginning to end, the following character shall lose its
 83173 special meaning (if any). It is unspecified what special meaning is given to any character other
 83174 than '&', ' \backslash ', ' $\%$ ', or digits.

83175 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the
 83176 <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as
 83177 part of a **g** or **v** *command list*. The current line number shall be set to the address of the last line
 83178 on which a substitution is performed. If no substitution is performed, the current line number
 83179 shall be unchanged. If a line is split, a substitution shall be considered to have been performed
 83180 on each of the new lines for the purpose of determining the new current line number. A
 83181 substitution shall be considered to have been performed even if the replacement string is
 83182 identical to the string that it replaces.

83183 The application shall ensure that the value of *flags* is zero or more of:

- 83184 *count* Substitute for the *count*th occurrence only of the RE found on each addressed line.
- 83185 **g** Globally substitute for all non-overlapping instances of the RE rather than just the first
 83186 one. If both **g** and *count* are specified, the results are unspecified.
- 83187 **l** Write to standard output the final line in which a substitution was made. The line shall
 83188 be written in the format specified for the **l** command.
- 83189 **n** Write to standard output the final line in which a substitution was made. The line shall
 83190 be written in the format specified for the **n** command.
- 83191 **p** Write to standard output the final line in which a substitution was made. The line shall
 83192 be written in the format specified for the **p** command.

83193 Copy Command

83194 *Synopsis:* (*. . .*)*taddress*

83195 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
 83196 shall be placed after address *address* (which can be 0); the current line number shall be set to the
 83197 address of the last line added.

83198 Undo Command

83199 *Synopsis:* **u**

83200 The **u** command shall nullify the effect of the most recent command that modified anything in
 83201 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
 83202 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
 83203 changes were made by the global command (such as with **g/RE/p**), the **u** command shall have
 83204 no effect. The current line number shall be set to the value it had immediately before the
 83205 command being undone started.

83206

Global Non-Matched Command

83207

Synopsis: (1,\$)v/RE/command list

83208

This command shall be equivalent to the global command **g** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

83209

83210

83211

Interactive Global Not-Matched Command

83212

Synopsis: (1,\$)V/RE/

83213

This command shall be equivalent to the interactive global command **G** except that the lines that are marked during the first step shall be those for which the line excluding the terminating <newline> does not match the RE.

83214

83215

83216

Write Command

83217

Synopsis: (1,\$)w [file]

83218

The **w** command shall write the addressed lines into the file named by the pathname *file*. The command shall create the file, if it does not exist, or shall replace the contents of the existing file. The currently remembered pathname shall not be changed unless there is no remembered pathname. If no pathname is given, the currently remembered pathname, if any, shall be used (see the **e** and **f** commands); the current line number shall be unchanged. If the command is successful, the number of bytes written shall be written to standard output, unless the **-s** option was specified, in the following format:

83219

83220

83221

83222

83223

83224

83225

"%d\n", <number of bytes written>

83226

If *file* begins with **'!'**, the rest of the line shall be taken to be a shell command line whose standard input shall be the addressed lines. Such a shell command line shall not be remembered as the current pathname. This usage of the write command with **'!'** shall not be considered as a "last **w** command that wrote the entire buffer", as described previously; thus, this alone shall not prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands.

83227

83228

83229

83230

83231

83232

Line Number Command

83233

Synopsis: (\$) =

83234

The line number of the addressed line shall be written to standard output in the following format:

83235

83236

"%d\n", <line number>

83237

The current line number shall be unchanged by this command.

83238

Shell Escape Command

83239

Synopsis: !command

83240

The remainder of the line after the **'!'** shall be sent to the command interpreter to be interpreted as a shell command line. Within the text of that shell command line, the unescaped character **'%'** shall be replaced with the remembered pathname; if a **'!'** appears as the first character of the command, it shall be replaced with the text of the previous shell command executed via **'!'**. Thus, **"!!"** shall repeat the previous **!command**. If any replacements of **'%'** or **'!'** are performed, the modified line shall be written to the standard output before *command* is executed. The **!** command shall write:

83241

83242

83243

83244

83245

83246

83247

"! \n"

83248 to standard output upon completion, unless the `-s` option is specified. The current line number
83249 shall be unchanged.

83250 Null Command

83251 *Synopsis:* (`. +1`)

83252 An address alone on a line shall cause the addressed line to be written. A `<newline>` alone shall
83253 be equivalent to `" +1p"`. The current line number shall be set to the address of the written line.

83254 EXIT STATUS

83255 The following exit values shall be returned:

- 83256 0 Successful completion without any file or command errors.
- 83257 >0 An error occurred.

83258 CONSEQUENCES OF ERRORS

83259 When an error in the input script is encountered, or when an error is detected that is a
83260 consequence of the data (not) present in the file or due to an external condition such as a read or
83261 write error:

- 83262 • If the standard input is a terminal device file, all input shall be flushed, and a new
83263 command read.
- 83264 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

83265 APPLICATION USAGE

83266 Because of the extremely terse nature of the default error messages, the prudent script writer
83267 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
83268 clue as to the cause is made available.

83269 In earlier versions of this standard, an obsolescent `-` option was described. This is no longer
83270 specified. Applications should use the `-s` option. Using `-` as a *file* operand now produces
83271 unspecified results. This allows implementations to continue to support the former required
83272 behavior.

83273 EXAMPLES

83274 None.

83275 RATIONALE

83276 The initial description of this utility was adapted from the SVID. It contains some features not
83277 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
83278 BSD *ed* utilities include, but need not be limited to:

- 83279 • The BSD `-` option does not suppress the `' ! '` prompt after a `!` command.
- 83280 • BSD does not support the special meanings of the `' % '` and `' ! '` characters within a `!`
83281 command.
- 83282 • BSD does not support the *addresses* `' ; '` and `' , '`.
- 83283 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this
83284 volume of POSIX.1-200x.
- 83285 • BSD does not support the `' ! '` character part of the **e**, **r**, or **w** commands.
- 83286 • A failed **g** command in BSD sets the line number to the last line searched if there are no
83287 matches.
- 83288 • BSD does not default the *command list* to the **p** command.

- 83289 • BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- 83290 • On BSD, if there is no inserted text, the insert command changes the current line to the
- 83291 referenced line `-1`; that is, the line before the specified line.
- 83292 • On BSD, the `join` command with only a single address changes the current line to that
- 83293 address.
- 83294 • BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p**
- 83295 command.
- 83296 • BSD does not support the `undo` of the commands **j**, **m**, **r**, **s**, or **t**.
- 83297 • The Version 7 `ed` command **W**, and the BSD `ed` commands **W**, **wq**, and **z** are not present in
- 83298 this volume of POSIX.1-200x.

83299 The `-s` option was added to allow the functionality of the removed `-` option in a manner |
 83300 compatible with the Utility Syntax Guidelines.

83301 In early proposals there was a limit, `{ED_FILE_MAX}`, that described the historical limitations of
 83302 some `ed` utilities in their handling of large files; some of these have had problems with files
 83303 larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a
 83304 `split` command in this volume of POSIX.1-200x. Since this limit was removed, this volume of
 83305 POSIX.1-200x requires that implementations document the file size limits imposed by `ed` in the
 83306 conformance document. The limit `{ED_LINE_MAX}` was also removed; therefore, the global
 83307 limit `{LINE_MAX}` is used for input and output lines.

83308 The manner in which the `I` command writes non-printable characters was changed to avoid the
 83309 historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous
 83310 because most terminals simply replace overstruck characters, making the `I` format not useful for
 83311 its intended purpose of unambiguously understanding the content of the line. The historical
 83312 backslash escapes were also ambiguous. (The string `"a\0011"` could represent a line containing
 83313 those six characters or a line containing the three characters `'a'`, a byte with a binary value of 1,
 83314 and a 1.) In the format required here, a backslash appearing in the line is written as `"\\"` so that
 83315 the output is truly unambiguous. The method of marking the ends of lines was adopted from
 83316 the `ex` editor and is required for any line ending in `<space>s`; the `'$'` is placed on all lines so
 83317 that a real `'$'` at the end of a line cannot be misinterpreted.

83318 Earlier versions of this standard allowed for implementations with bytes other than eight bits, |
 83319 but this has been modified in this version.

83320 The description of how a NUL is written was removed. The NUL character cannot be in text
 83321 files, and this volume of POSIX.1-200x should not dictate behavior in the case of undefined,
 83322 erroneous input.

83323 Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands
 83324 are not patterns.

83325 Early proposals stated that the `-p` option worked only when standard input was associated with
 83326 a terminal device. This has been changed to conform to historical implementations, thereby
 83327 allowing applications to interpose themselves between a user and the `ed` utility.

83328 The form of the substitute command that uses the **n** suffix was limited in some historical
 83329 documentation (where this was described incorrectly as “backreferencing”). This limit has been
 83330 omitted because there is no reason why an editor processing lines of `{LINE_MAX}` length should
 83331 have this restriction. The command `s/x/X/2047` should be able to substitute the 2 047th occurrence
 83332 of `'x'` on a line.

83333 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made
 83334 unspecified because BSD-based systems allow this, whereas System V does not.

83335 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file

83336 have been deleted. Since this volume of POSIX.1-200x refers to the **q** command in this instance,
83337 such behavior is not allowed.

83338 Some historical implementations returned exit status zero even if command errors had occurred;
83339 this is not allowed by this volume of POSIX.1-200x.

83340 Some historical implementations contained a bug that allowed a single period to be entered in
83341 input mode as `<backslash> <period> <newline>`. This is not allowed by *ed* because there is no
83342 description of escaping any of the characters in input mode; backslashes are entered into the
83343 buffer exactly as typed. The typical method of entering a single period has been to precede it
83344 with another character and then use the substitute command to delete that character.

83345 It is difficult under some modes of some versions of historical operating system terminal drivers
83346 to distinguish between an end-of-file condition and terminal disconnect. POSIX.1-200x does not
83347 require implementations to distinguish between the two situations, which permits historical
83348 implementations of the *ed* utility on historical platforms to conform. Implementations are
83349 encouraged to distinguish between the two, if possible, and take appropriate action on terminal
83350 disconnect.

83351 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the
83352 start of the edit buffer. When the buffer was empty the command `.=` returned zero. POSIX.1-200x
83353 requires conformance to historical practice.

83354 For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands
83355 must also accept an address of 0, in which case `0i` is treated as `1i` and likewise for the **c**
83356 command.

83357 All of the following are valid addresses:

83358	<code>+++</code>	Three lines after the current line.
83359	<code>/pattern/-</code>	One line before the next occurrence of pattern.
83360	<code>-2</code>	Two lines before the current line.
83361	<code>3 ---- 2</code>	Line one (note the intermediate negative address).
83362	<code>1 2 3</code>	Line six.

83363 Any number of addresses can be provided to commands taking addresses; for example,
83364 `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses
83365 accepted by the **print** command. This, in combination with the semicolon delimiter, permits
83366 users to create commands based on ordered patterns in the file. For example, the command
83367 `"3;/foo/;+2p"` will display the first line after line 3 that contains the pattern *foo*, plus the next
83368 two lines. Note that the address `"3;"` must still be evaluated before being discarded, because
83369 the search origin for the `"/foo/"` command depends on this.

83370 Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or
83371 semicolon separators; for example, `","` or `";;"` were considered an error. For consistency of
83372 address specification, this restriction is removed. The following table lists some of the address
83373 forms now possible:

	Address	Addr1	Addr2	Status	Comment
83374	7,	7	7	Historical	
83375	7,5,	5	5	Historical	
83376	7,5,9	5	9	Historical	
83377	7,9	7	9	Historical	
83378	7,+	7	8	Historical	
83379	,	1	\$	Historical	
83380	,7	1	7	Extension	
83381	,,	\$	\$	Extension	
83382	,i	\$	\$	Extension	
83383	7i	7	7	Historical	
83384	7i5i	5	5	Historical	
83385	7i5i9	5	9	Historical	
83386	7i5,9	5	9	Historical	
83387	7i\$;4	\$	4	Historical	Valid, but erroneous.
83388	7i9	7	9	Historical	
83389	7i+	7	8	Historical	
83390	i	.	\$	Historical	
83391	i7	.	7	Extension	
83392	ii	\$	\$	Extension	
83393	i,	\$	\$	Extension	
83394					

Historically, values could be added to addresses by including them after one or more <blank>s; for example, "3 - 5p" wrote the seventh line of the file, and "/foo/ 5" was the same as "5 /foo/". However, only absolute values could be added; for example, "5 /foo/" was an error. POSIX.1-200x requires conformance to historical practice.

Historically, *ed* accepted the '^' character as an address, in which case it was identical to the hyphen character. POSIX.1-200x does not require or prohibit this behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 1.4](#) (on page 2235), *ex*, *sed*, *sh*, *vi*

[XBD Table 5-1](#) (on page 108), [Chapter 8](#) (on page 159), [Section 9.3](#) (on page 169), [Chapter 11](#) (on page 185), [Section 12.2](#) (on page 201)

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the OPTIONS section, the meaning of `-s` and `-` is clarified.

A second FUTURE DIRECTION is added.

Issue 6

The obsolescent single-minus form is removed.

A second APPLICATION USAGE note is added.

The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, and changes to processing when end-of-file is detected and when terminal disconnect is detected.

The normative text is reworded to avoid use of the term "must" for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/22 is applied, adding the text: "Any line

83421

modified by the *command list* shall be unmarked.” to the **G** command. This change corresponds to a similar change made to the **g** command in the first version of this standard.

83422

83423

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/7 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

83424

83425

Issue 7

83426

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is ‘-’.

83427

83428

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

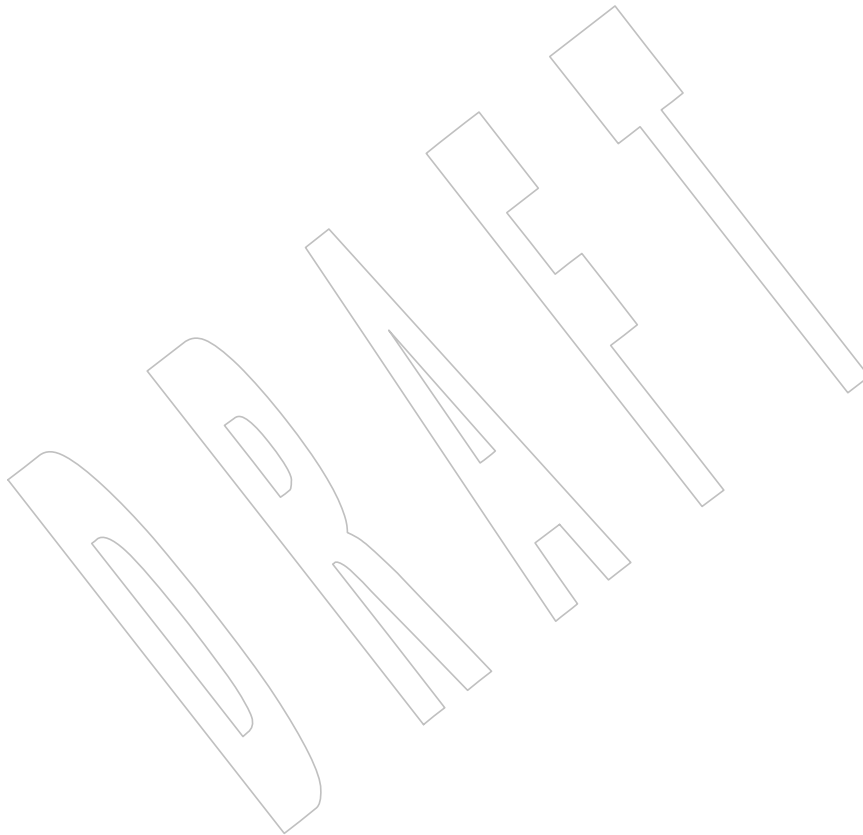
83429

SD5-XCU-ERN-94 is applied, updating text in the EXTENDED DESCRIPTION where a terminal disconnect is detected (in Commands in *ed*).

83430

83431

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



83432 **NAME**83433 `env` — set the environment for command invocation83434 **SYNOPSIS**83435 `env [-i] [name=value]... [utility [argument...]]`83436 **DESCRIPTION**83437 The `env` utility shall obtain the current environment, modify it according to its arguments, then
83438 invoke the utility named by the `utility` operand with the modified environment.83439 Optional arguments shall be passed to `utility`.83440 If no `utility` operand is specified, the resulting environment shall be written to the standard
83441 output, with one `name=value` pair per line.83442 If the first argument is `'-'`, the results are unspecified.83443 **OPTIONS**83444 The `env` utility shall conform to XBD Section 12.2 (on page 201), except for the unspecified usage
83445 of `'-'`.

83446 The following options shall be supported:

83447 **-i** Invoke `utility` with exactly the environment specified by the arguments; the
83448 inherited environment shall be ignored completely.83449 **OPERANDS**

83450 The following operands shall be supported:

83451 `name=value` Arguments of the form `name=value` shall modify the execution environment, and
83452 shall be placed into the inherited environment before the `utility` is invoked.83453 `utility` The name of the utility to be invoked. If the `utility` operand names any of the
83454 special built-in utilities in Section 2.14 (on page 2280), the results are undefined.83455 `argument` A string to pass as an argument for the invoked utility.83456 **STDIN**

83457 Not used.

83458 **INPUT FILES**

83459 None.

83460 **ENVIRONMENT VARIABLES**83461 The following environment variables shall affect the execution of `env`:83462 `LANG` Provide a default value for the internationalization variables that are unset or null.
83463 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
83464 variables used to determine the values of locale categories.)83465 `LC_ALL` If set to a non-empty string value, override the values of all the other
83466 internationalization variables.83467 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
83468 characters (for example, single-byte as opposed to multi-byte characters in
83469 arguments).83470 `LC_MESSAGES`83471 Determine the locale that should be used to affect the format and contents of
83472 diagnostic messages written to standard error.

env*Utilities*

83473 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

83474 **PATH** Determine the location of the *utility*, as described in XBD Chapter 8 (on page 159).
 83475 If *PATH* is specified as a *name=value* operand to *env*, the *value* given shall be used in
 83476 the search for *utility*.

ASYNCHRONOUS EVENTS

83477 Default.

STDOUT

83480 If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be
 83481 written in the form:

83482 "%s=%s\n", <name>, <value>

83483 If the *utility* operand is specified, the *env* utility shall not write to standard output.

STDERR

83484 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

83486 None.

EXTENDED DESCRIPTION

83488 None.

EXIT STATUS

83491 If *utility* is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env*
 83492 utility shall exit with one of the following values:

83493 0 The *env* utility completed successfully.

83494 1–125 An error occurred in the *env* utility.

83495 126 The utility specified by *utility* was found but could not be invoked.

83496 127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

83497 Default.

APPLICATION USAGE

83500 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 83501 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
 83502 utility exited with an error indication”. The value 127 was chosen because it is not commonly
 83503 used for other meanings; most utilities use small values for “normal error conditions” and the
 83504 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 83505 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 83506 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 83507 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 83508 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 83509 any other reason.

83510 Historical implementations of the *env* utility use the *execvp()* or *execlp()* functions defined in the
 83511 System Interfaces volume of POSIX.1-200x to invoke the specified utility; this provides better
 83512 performance and keeps users from having to escape characters with special meaning to the shell.
 83513 Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are
 83514 not found.

EXAMPLES

The following command:

```
env -i PATH=/mybin:"$PATH" $(getconf V7_ENV) mygrep xyz myfile
```

invokes the command *mygrep* with a new *PATH* value as the only entry in its environment other than any variables required by the implementation for conformance. In this case, *PATH* is used to locate *mygrep*, which is expected to reside in **/mybin**.

RATIONALE

As with all other utilities that invoke other utilities, this volume of POSIX.1-200x only specifies what *env* does with standard input, standard output, standard error, input files, and output files. If a utility is executed, it is not constrained by the specification of input and output by *env*.

The *-i* option was added to allow the functionality of the removed *-* option in a manner compatible with the Utility Syntax Guidelines. It is possible to create a non-conforming environment using the *-i* option, as it may remove environment variables required by the implementation for conformance. The following will preserve these environment variables as well as preserve the *PATH* for conforming utilities:

```
IFS='
# The preceding value should be <space><tab><newline>.
# Set IFS to its default value.

set -f
# disable pathname expansion

\unalias -a
# Unset all possible aliases.
# Note that unalias is escaped to prevent an alias
# being used for unalias.
# This step is not strictly necessary, since aliases are not inherited,
# and the ENV environment variable is only used by interactive shells,
# the only way any aliases can exist in a script is if it defines them
# itself.

unset -f env getconf
# Ensure env and getconf are not user functions.

env -i $(getconf V7_ENV) PATH="$(getconf PATH)" command
```

Some have suggested that *env* is redundant since the same effect is achieved by:

```
name=value ... utility [ argument ... ]
```

The example is equivalent to *env* when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The *env* utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of *env*.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.14](#) (on page 2280), [Section 2.5](#) (on page 2249)

[XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

83558
83559
83560
83561
83562
83563
83564
83565
83566**CHANGE HISTORY**

First released in Issue 2.

Issue 7

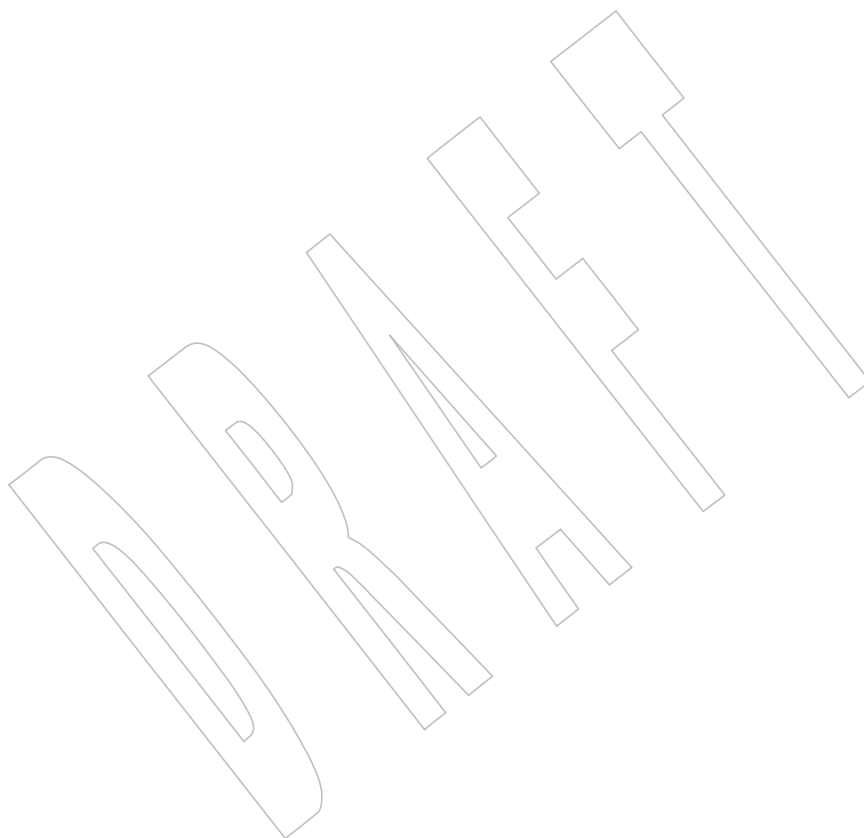
Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is '- '.

Austin Group Interpretation 1003.1-2001 #047 is applied, providing RATIONALE on how to use the *env* utility to preserve a conforming environment.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The EXAMPLES section is revised to change the use of *env* -i.

+



83567 **NAME**
 83568 `ex` — text editor

83569 **SYNOPSIS**
 83570 UP `ex [-rR] [-s|-v] [-c command] [-t tagstring] [-w size] [file...]`

83571 DESCRIPTION

83572 The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and
 83573 visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open**
 83574 and **visual** commands and in *vi*.

83575 If an operand is `'-'`, the results are unspecified.

83576 This section uses the term *edit buffer* to describe the current working text. No specific
 83577 implementation is implied by this term. All editing changes are performed on the edit buffer,
 83578 and no changes to it shall affect any file until an editor command writes the file.

83579 Certain terminals do not have all the capabilities necessary to support the complete *ex* definition,
 83580 such as the full-screen editing commands (*visual mode* or *open mode*). When these commands
 83581 cannot be supported on such terminals, this condition shall not produce an error message such
 83582 as “not an editor command” or report a syntax error. The implementation may either accept the
 83583 commands and produce results on the screen that are the result of an unsuccessful attempt to
 83584 meet the requirements of this volume of POSIX.1-200x or report an error describing the terminal-
 83585 related deficiency.

83586 OPTIONS

83587 The *ex* utility shall conform to XBD [Section 12.2](#) (on page 201), except for the unspecified usage
 83588 of `'-'`, and that `'+'` may be recognized as an option delimiter as well as `'-'`.

83589 The following options shall be supported:

83590 `-c command` Specify an initial command to be executed in the first edit buffer loaded from an
 83591 existing file (see the EXTENDED DESCRIPTION section). Implementations may
 83592 support more than a single `-c` option. In such implementations, the specified
 83593 commands shall be executed in the order specified on the command line.

83594 `-r` Recover the named files (see the EXTENDED DESCRIPTION section). Recovery
 83595 information for a file shall be saved during an editor or system crash (for example,
 83596 when the editor is terminated by a signal which the editor can catch), or after the
 83597 use of an *ex* **preserve** command.

83598 A *crash* in this context is an unexpected failure of the system or utility that requires
 83599 restarting the failed system or utility. A system crash implies that any utilities
 83600 running at the time also crash. In the case of an editor or system crash, the number
 83601 of changes to the edit buffer (since the most recent **preserve** command) that will be
 83602 recovered is unspecified.

83603 If no *file* operands are given and the `-t` option is not specified, all other options, the
 83604 *EXINIT* variable, and any *.exrc* files shall be ignored; a list of all recoverable files
 83605 available to the invoking user shall be written, and the editor shall exit normally
 83606 without further action.

83607 `-R` Set **readonly** edit option.

83608 `-s` Prepare *ex* for batch use by taking the following actions:

- 83609 • Suppress writing prompts and informational (but not diagnostic) messages.
- 83610 • Ignore the value of *TERM* and any implementation default terminal type and
- 83611 assume the terminal is a type incapable of supporting open or visual modes;
- 83612 see the **visual** command and the description of *vi*.
- 83613 • Suppress the use of the *EXINIT* environment variable and the reading of any
- 83614 **.exrc** file; see the EXTENDED DESCRIPTION section.
- 83615 • Suppress autoindentation, ignoring the value of the **autoindent** edit option.
- 83616 **-t tagstring** Edit the file containing the specified *tagstring*; see *ctags*. The tags feature
- 83617 represented by **-t tagstring** and the **tag** command is optional. It shall be provided
- 83618 on any system that also provides a conforming implementation of *ctags*; otherwise,
- 83619 the use of **-t** produces undefined results. On any system, it shall be an error to
- 83620 specify more than a single **-t** option.
- 83621 **-v** Begin in visual mode (see *vi*).
- 83622 **-w size** Set the value of the *window* editor option to *size*.

OPERANDS

The following operand shall be supported:

- 83623 *file* A pathname of a file to be edited.

STDIN

The standard input consists of a series of commands and input text, as described in the EXTENDED DESCRIPTION section. The implementation may limit each line of standard input to a length of {LINE_MAX}.

If the standard input is not a terminal device, it shall be as if the **-s** option had been specified.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

INPUT FILES

Input files shall be text files or files that would be text files except for an incomplete last line that is not longer than {LINE_MAX}-1 bytes in length and contains no NUL characters. By default, any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other forms of files may optionally be allowed by *ex* implementations.

The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED DESCRIPTION section.

By default, the editor shall read lines from the files to be edited without interpreting any of those lines as any form of editor command.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ex*:

- 83644 **COLUMNS** Override the system-selected horizontal screen size. See XBD Chapter 8 (on page |
- 83645 159) for valid values and results when it is unset or null.
- 83646 **EXINIT** Determine a list of *ex* commands that are executed on editor start-up. See the |
- 83647 EXTENDED DESCRIPTION section for more details of the initialization phase.
- 83648 **HOME** Determine a pathname of a directory that shall be searched for an editor start-up |
- 83649 file named **.exrc**; see the EXTENDED DESCRIPTION section.
- 83650 **LANG** Provide a default value for the internationalization variables that are unset or null. |
- 83651 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
- 83652 variables used to determine the values of locale categories.)

83653		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
83654			
83655		<i>LC_COLLATE</i>	Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.
83656			
83657			
83658		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as uppercase or lowercase letters, the case conversion of letters, and the detection of word boundaries.
83659			
83660			
83661			
83662			
83663		<i>LC_MESSAGES</i>	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
83664			
83665			
83666		<i>LINES</i>	Override the system-selected vertical screen size, used as the number of lines in a screenful and the vertical screen size in visual mode. See XBD Chapter 8 (on page 159) for valid values and results when it is unset or null.
83667			
83668			
83669	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
83670		<i>PATH</i>	Determine the search path for the shell command specified in the <i>ex</i> editor commands ! , shell , read , and write , and the open and visual mode command ! ; see the description of command search and execution in Section 2.9.1.1 (on page 2264).
83671			
83672			
83673		<i>SHELL</i>	Determine the preferred command line interpreter for use as the default value of the shell edit option.
83674			
83675		<i>TERM</i>	Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type shall be used.
83676			

ASYNCHRONOUS EVENTS

The following term is used in this and following sections to specify command and asynchronous event actions:

complete write

A complete write is a write of the entire contents of the edit buffer to a file of a type other than a terminal device, or the saving of the edit buffer caused by the user executing the *ex* **preserve** command. Writing the contents of the edit buffer to a temporary file that will be removed when the editor exits shall not be considered a complete write.

The following actions shall be taken upon receipt of signals:

SIGINT If the standard input is not a terminal device, *ex* shall not write the file or return to command or text input mode, and shall exit with a non-zero exit status.

Otherwise, if executing an open or visual text input mode command, *ex* in receipt of **SIGINT** shall behave identically to its receipt of the <ESC> character.

Otherwise:

1. If executing an *ex* text input mode command, all input lines that have been completely entered shall be resolved into the edit buffer, and any partially entered line shall be discarded.
2. If there is a currently executing command, it shall be aborted and a message displayed. Unless otherwise specified by the *ex* or *vi* command descriptions, it is unspecified whether any lines modified by the executing command appear modified, or as they were before being modified by the executing command, in the buffer.

- 83700 If the currently executing command was a motion command, its associated
83701 command shall be discarded.
- 83702 3. If in open or visual command mode, the terminal shall be alerted.
- 83703 4. The editor shall then return to command mode.
- 83704 SIGCONT The screen shall be refreshed if in open or visual mode.
- 83705 SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt
83706 to save the edit buffer so that it can be recovered later using the **-r** option or the *ex*
83707 **recover** command. The editor shall not write the file or return to command or text
83708 input mode, and shall terminate with a non-zero exit status.
- 83709 SIGTERM Refer to SIGHUP.
- 83710 The action taken for all other signals is unspecified.

STDOUT

83711 The standard output shall be used only for writing prompts to the user, for informational
83712 messages, and for writing lines from the file.
83713

STDERR

83714 The standard error shall be used only for diagnostic messages.
83715

OUTPUT FILES

83716 The output from *ex* shall be text files.
83717

EXTENDED DESCRIPTION

83718 Only the *ex* mode of the editor is described in this section. See *vi* for additional editing
83719 capabilities available in *ex*.
83720

83721 When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such
83722 as inverse video), the message shall be written in standout mode. If the terminal does not
83723 support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the
83724 error message.

83725 By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the
83726 **prompt** command. Text input mode can be entered by the **append**, **insert**, or **change** commands;
83727 it can be exited (and command mode re-entered) by typing a period (**.**) alone at the beginning
83728 of a line.

Initialization in *ex* and *vi*

83729 The following symbols are used in this and following sections to specify locations in the edit
83730 buffer:
83731

alternate and current pathnames

83732 Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex*
83733 commands that take filenames as arguments shall set them as follows:
83734

- 83735 1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag**
83736 command replaces the contents of the edit buffer.
- 83737 a. If the command replaces the contents of the edit buffer, the current pathname
83738 shall be set to the *file* argument or the file indicated by the tag, and the
83739 alternate pathname shall be set to the previous value of the current pathname.
- 83740 b. Otherwise, the alternate pathname shall be set to the *file* argument.
- 83741 2. If a *file* argument is specified to the *ex* **next** command:

- 83742 a. If the command replaces the contents of the edit buffer, the current pathname
83743 shall be set to the first *file* argument, and the alternate pathname shall be set to
83744 the previous value of the current pathname.
- 83745 3. If a *file* argument is specified to the *ex file* command, the current pathname shall be
83746 set to the *file* argument, and the alternate pathname shall be set to the previous value
83747 of the current pathname.
- 83748 4. If a *file* argument is specified to the *ex read* and *write* commands (that is, when
83749 reading or writing a file, and not to the program named by the *shell* edit option), or a
83750 *file* argument is specified to the *ex xit* command:
- 83751 a. If the current pathname has no value, the current pathname shall be set to the
83752 *file* argument.
- 83753 b. Otherwise, the alternate pathname shall be set to the *file* argument.

83754 If the alternate pathname is set to the previous value of the current pathname when the
83755 current pathname had no previous value, then the alternate pathname shall have no value
83756 as a result.

83757 *current line*

83758 The line of the edit buffer referenced by the cursor. Each command description specifies the
83759 current line after the command has been executed, as the *current line value*. When the edit
83760 buffer contains no lines, the current line shall be zero; see [Addressing in ex](#) (on page 2579).

83761 *current column*

83762 The current display line column occupied by the cursor. (The columns shall be numbered
83763 beginning at 1.) Each command description specifies the current column after the command
83764 has been executed, as the *current column value*. This column is an *ideal* column that is
83765 remembered over the lifetime of the editor. The actual display line column upon which the
83766 cursor rests may be different from the current column; see the cursor positioning discussion
83767 in [Command Descriptions in vi](#) (on page 3216).

83768 *set to non-<blank>*

83769 A description for a current column value, meaning that the current column shall be set to
83770 the last display line column on which is displayed any part of the first non-<blank> of the
83771 line. If the line has no non-<blank> non-<newline>s, the current column shall be set to the
83772 last display line column on which is displayed any part of the last non-<newline> in the
83773 line. If the line is empty, the current column shall be set to column position 1.

83774 The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual
83775 mode, the length of lines in the edit buffer may be limited to the number of characters that will
83776 fit in the display. If either limit is exceeded during editing, an error message shall be written. If
83777 either limit is exceeded by a line read in from a file, an error message shall be written and the
83778 edit session may be terminated.

83779 If the editor stops running due to any reason other than a user command, and the edit buffer has
83780 been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous
83781 event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

83782 During initialization (before the first file is copied into the edit buffer or any user commands
83783 from the terminal are processed) the following shall occur:

- 83784 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands
83785 contained in that variable.
- 83786 2. If the *EXINIT* variable is not set, and all of the following are true:

- 83787 a. The *HOME* environment variable is not null and not empty.
- 83788 b. The file *.exrc* in the directory referred to by the *HOME* environment variable:
- 83789 1. Exists
- 83790 2. Is owned by the same user ID as the real user ID of the process or the
- 83791 process has appropriate privileges
- 83792 3. Is not writable by anyone other than the owner

83793 the editor shall execute the *ex* commands contained in that file.

- 83794 3. If and only if all of the following are true:
- 83795 a. The current directory is not referred to by the *HOME* environment variable.
- 83796 b. A command in the *EXINIT* environment variable or a command in the *.exrc* file in
- 83797 the directory referred to by the *HOME* environment variable sets the editor option
- 83798 **exrc**.
- 83799 c. The *.exrc* file in the current directory:
- 83800 1. Exists
- 83801 2. Is owned by the same user ID as the real user ID of the process, or by one of
- 83802 a set of implementation-defined user IDs
- 83803 3. Is not writable by anyone other than the owner

83804 the editor shall attempt to execute the *ex* commands contained in that file.

83805 Lines in any *.exrc* file that are blank lines shall be ignored. If any *.exrc* file exists, but is not read

83806 for ownership or permission reasons, it shall be an error.

83807 After the *EXINIT* variable and any *.exrc* files are processed, the first file specified by the user

83808 shall be edited, as follows:

- 83809 1. If the user specified the **-t** option, the effect shall be as if the *ex tag* command was entered
- 83810 with the specified argument, with the exception that if tag processing does not result in a
- 83811 file to edit, the effect shall be as described in step 3. below.
- 83812 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
- 83813 the *ex edit* command was entered with the first of those arguments as its *file* argument.
- 83814 3. Otherwise, the effect shall be as if the *ex edit* command was entered with a nonexistent
- 83815 filename as its *file* argument. It is unspecified whether this action shall set the current
- 83816 pathname. In an implementation where this action does not set the current pathname, any
- 83817 editor command using the current pathname shall fail until an editor command sets the
- 83818 current pathname.

83819 If the **-r** option was specified, the first time a file in the initial argument list or a file specified by

83820 the **-t** option is edited, if recovery information has previously been saved about it, that

83821 information shall be recovered and the editor shall behave as if the contents of the edit buffer

83822 have already been modified. If there are multiple instances of the file to be recovered, the one

83823 most recently saved shall be recovered, and an informational message that there are previous

83824 versions of the file that can be recovered shall be written. If no recovery information about a file

83825 is available, an informational message to this effect shall be written, and the edit shall proceed as

83826 usual.

83827 If the **-c** option was specified, the first time a file that already exists (including a file that might

83828 not exist but for which recovery information is available, when the **-r** option is specified)

83829 replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of

83830 the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified

83831 with the `-c` option shall be executed. In this case, the current line and current column shall not
 83832 be set as described for the command associated with the replacement or initialization of the edit
 83833 buffer contents. However, if the `-t` option or a **tag** command is associated with this action, the `-c`
 83834 option commands shall be executed and then the movement to the tag shall be performed.

83835 The current argument list shall initially be set to the filenames specified by the user on the
 83836 command line. If no filenames are specified by the user, the current argument list shall be empty.
 83837 If the `-t` option was specified, it is unspecified whether any filename resulting from tag
 83838 processing shall be prepended to the current argument list. In the case where the filename is
 83839 added as a prefix to the current argument list, the current argument list reference shall be set to
 83840 that filename. In the case where the filename is not added as a prefix to the current argument
 83841 list, the current argument list reference shall logically be located before the first of the filenames
 83842 specified on the command line (for example, a subsequent *ex next* command shall edit the first
 83843 filename from the command line). If the `-t` option was not specified, the current argument list
 83844 reference shall be to the first of the filenames on the command line.

83845 Addressing in *ex*

83846 Addressing in *ex* relates to the current line and the current column; the address of a line is its
 83847 1-based line number, the address of a column is its 1-based count from the beginning of the line.
 83848 Generally, the current line is the last line affected by a command. The current line number is the
 83849 address of the current line. In each command description, the effect of the command on the
 83850 current line number and the current column is described.

83851 Addresses are constructed as follows:

- 83852 1. The character `'.'` (period) shall address the current line.
- 83853 2. The character `'$'` shall address the last line of the edit buffer.
- 83854 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
- 83855 4. The address `"'x"` refers to the line marked with the mark name character `'x'`, which
 83856 shall be a lowercase letter from the portable character set or one of the characters `'\'` or
 83857 `'\''`. It shall be an error if the line that was marked is not currently present in the edit
 83858 buffer or the mark has not been set. Lines can be marked with the *ex mark* or *k*
 83859 commands, or the *vi m* command.
- 83860 5. A regular expression enclosed by slashes (`'/'`) shall address the first line found by
 83861 searching forwards from the line following the current line toward the end of the edit
 83862 buffer and stopping at the first line for which the line excluding the terminating
 83863 `<newline>` matches the regular expression. As stated in [Regular Expressions in *ex*](#) (on
 83864 page 2608), an address consisting of a null regular expression delimited by slashes `"/"`
 83865 shall address the next line for which the line excluding the terminating `<newline>`
 83866 matches the last regular expression encountered. In addition, the second slash can be
 83867 omitted at the end of a command line. If the **wrapsan** edit option is set, the search shall
 83868 wrap around to the beginning of the edit buffer and continue up to and including the
 83869 current line, so that the entire edit buffer is searched. Within the regular expression, the
 sequence `"\"` shall represent a literal slash instead of the regular expression delimiter.
- 83871 6. A regular expression enclosed in question marks (`'?'`) shall address the first line found
 83872 by searching backwards from the line preceding the current line toward the beginning of
 83873 the edit buffer and stopping at the first line for which the line excluding the terminating
 83874 `<newline>` matches the regular expression. An address consisting of a null regular
 83875 expression delimited by question marks `"??"` shall address the previous line for which
 83876 the line excluding the terminating `<newline>` matches the last regular expression
 83877 encountered. In addition, the second question mark can be omitted at the end of a
 83878 command line. If the **wrapsan** edit option is set, the search shall wrap around from the
 83879 beginning of the edit buffer to the end of the edit buffer and continue up to and including

83880 the current line, so that the entire edit buffer is searched. Within the regular expression,
83881 the sequence "\?" shall represent a literal question mark instead of the RE delimiter.

83882 7. A plus sign ('+') or a minus sign ('-') followed by a decimal number shall address the
83883 current line plus or minus the number. A '+' or '-' not followed by a decimal number
83884 shall address the current line plus or minus 1.

83885 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
83886 Address offsets are constructed as follows:

- 83887 1. A '+' or '-' immediately followed by a decimal number shall add (subtract) the
83888 indicated number of lines to (from) the address. A '+' or '-' not followed by a decimal
83889 number shall add (subtract) 1 to (from) the address.
- 83890 2. A decimal number shall add the indicated number of lines to the address.

83891 It shall not be an error for an intermediate address value to be less than zero or greater than the
83892 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
83893 greater than the last line in the edit buffer.

83894 Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in
83895 [Command Descriptions in ex](#) (on page 2586). If more than the required number of addresses are
83896 provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than
83897 the required number of addresses are provided to a command, the addresses specified first shall
83898 be evaluated and then discarded until the maximum number of valid addresses remain.

83899 Addresses shall be separated from each other by a comma (',') or a semicolon (';'). If no
83900 address is specified before or after a comma or semicolon separator, it shall be as if the address
83901 of the current line was specified before or after the separator. In the case of a semicolon
83902 separator, the current line ('.') shall be set to the first address, and only then will the next
83903 address be calculated. This feature can be used to determine the starting line for forwards and
83904 backwards searches (see rules 5. and 6.).

83905 A percent sign ('%') shall be equivalent to entering the two addresses "1,\$".

83906 Any delimiting <blank>s between addresses, address separators, or address offsets shall be
83907 discarded.

83908 **Command Line Parsing in ex**

83909 The following symbol is used in this and following sections to describe parsing behavior:

83910 *escape* If a character is referred to as "backslash-escaped" or "<control>-V-escaped," it
83911 shall mean that the character acquired or lost a special meaning by virtue of being
83912 preceded, respectively, by a backslash or <control>-V character. Unless otherwise
83913 specified, the escaping character shall be discarded at that time and shall not be
83914 further considered for any purpose.

83915 Command-line parsing shall be done in the following steps. For each step, characters already
83916 evaluated shall be ignored; that is, the phrase "leading character" refers to the next character
83917 that has not yet been evaluated.

- 83918 1. Leading colon characters shall be skipped.
- 83919 2. Leading <blank>s shall be skipped.
- 83920 3. If the leading character is a double-quote character, the characters up to and including the
83921 next non-backslash-escaped <newline> shall be discarded, and any subsequent characters
83922 shall be parsed as a separate command.

- 83923 4. Leading characters that can be interpreted as addresses shall be evaluated; see
83924 [Addressing in ex](#) (on page 2579).
- 83925 5. Leading <blank>s shall be skipped.
- 83926 6. If the next character is a vertical-line character or a <newline>:
- 83927 a. If the next character is a <newline>:
- 83928 1. If *ex* is in open or visual mode, the current line shall be set to the last
83929 address specified, if any.
- 83930 2. Otherwise, if the last command was terminated by a vertical-line character,
83931 no action shall be taken; for example, the command "| |<newline>" shall
83932 execute two implied commands, not three.
- 83933 3. Otherwise, step 6.b. shall apply.
- 83934 b. Otherwise, the implied command shall be the **print** command. The last #, **p**, and **I**
83935 flags specified to any *ex* command shall be remembered and shall apply to this
83936 implied command. Executing the *ex* **number**, **print**, or **list** command shall set the
83937 remembered flags to #, nothing, and **I**, respectively, plus any other flags specified
83938 for that execution of the **number**, **print**, or **list** command.
- 83939 If *ex* is not currently performing a **global** or **v** command, and no address or count
83940 is specified, the current line shall be incremented by 1 before the command is
83941 executed. If incrementing the current line would result in an address past the last
83942 line in the edit buffer, the command shall fail, and the increment shall not happen.
- 83943 c. The <newline> or vertical-line character shall be discarded and any subsequent
83944 characters shall be parsed as a separate command.
- 83945 7. The command name shall be comprised of the next character (if the character is not
83946 alphabetic), or the next character and any subsequent alphabetic characters (if the
83947 character is alphabetic), with the following exceptions:
- 83948 a. Commands that consist of any prefix of the characters in the command name
83949 **delete**, followed immediately by any of the characters 'l', 'p', '+', '-', or '#'
83950 shall be interpreted as a **delete** command, followed by a <blank>, followed by the
83951 characters that were not part of the prefix of the **delete** command. The maximum
83952 number of characters shall be matched to the command name **delete**; for example,
83953 "de1" shall not be treated as "de" followed by the flag **I**.
- 83954 b. Commands that consist of the character 'k', followed by a character that can be
83955 used as the name of a mark, shall be equivalent to the mark command followed by
83956 a <blank>, followed by the character that followed the 'k'.
- 83957 c. Commands that consist of the character 's', followed by characters that could be
83958 interpreted as valid options to the **s** command, shall be the equivalent of the **s**
83959 command, without any pattern or replacement values, followed by a <blank>,
83960 followed by the characters after the 's'.
- 83961 8. The command name shall be matched against the possible command names, and a
83962 command name that contains a prefix matching the characters specified by the user shall
83963 be the executed command. In the case of commands where the characters specified by the
83964 user could be ambiguous, the executed command shall be as follows:

83965
83966
83967
83968
83969
83970

a	append	n	next	t	t
c	change	p	print	u	undo
ch	change	pr	print	un	undo
e	edit	r	read	v	v
m	move	re	read	w	write
ma	mark	s	s		

83971
83972
83973

Implementation extensions with names causing similar ambiguities shall not be checked for a match until all possible matches for commands specified by POSIX.1-200x have been checked.

83974
83975
83976
83977
83978

9. If the command is a **!** command, or if the command is a **read** command followed by zero or more `<blank>s` and a **!**, or if the command is a **write** command followed by one or more `<blank>s` and a **!**, the rest of the command shall include all characters up to a non-backslash-escaped `<newline>`. The `<newline>` shall be discarded and any subsequent characters shall be parsed as a separate *ex* command.

83979
83980

10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in open or visual mode, the next part of the command shall be parsed as follows:

83981
83982

- a. Any `'!'` character immediately following the command shall be skipped and be part of the command.

83983

- b. Any leading `<blank>s` shall be skipped and be part of the command.

83984
83985
83986

- c. If the next character is a `'+'`, characters up to the first non-backslash-escaped `<newline>` or non-backslash-escaped `<blank>` shall be skipped and be part of the command.

83987
83988

- d. The rest of the command shall be determined by the steps specified in paragraph 12.

83989
83990

11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the command shall be parsed as follows:

83991

- a. Any leading `<blank>s` shall be skipped and be part of the command.

83992
83993

- b. If the next character is not an alphanumeric, double-quote, `<newline>`, backslash, or vertical-line character:

83994

1. The next character shall be used as a command delimiter.

83995
83996
83997

2. If the command is a **global**, **open**, or **v** command, characters up to the first non-backslash-escaped `<newline>`, or first non-backslash-escaped delimiter character, shall be skipped and be part of the command.

83998
83999
84000

3. If the command is an **s** command, characters up to the first non-backslash-escaped `<newline>`, or second non-backslash-escaped delimiter character, shall be skipped and be part of the command.

84001
84002

- c. If the command is a **global** or **v** command, characters up to the first non-backslash-escaped `<newline>` shall be skipped and be part of the command.

84003
84004

- d. Otherwise, the rest of the command shall be determined by the steps specified in paragraph 12.

84005

12. Otherwise:

84006
84007
84008

- a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command, characters up to the first non-`<control>-V`-escaped `<newline>`, vertical-line, or double-quote character shall be skipped and be part of the command.

- 84009 b. Otherwise, characters up to the first non-backslash-escaped <newline>, vertical-
84010 line, or double-quote character shall be skipped and be part of the command.
- 84011 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
84012 ended at a vertical-line character, any subsequent characters, up to the next non-
84013 backslash-escaped <newline> shall be used as input text to the command.
- 84014 d. If the command was ended by a double-quote character, all subsequent characters,
84015 up to the next non-backslash-escaped <newline>, shall be discarded.
- 84016 e. The terminating <newline> or vertical-line character shall be discarded and any
84017 subsequent characters shall be parsed as a separate *ex* command.

84018 Command arguments shall be parsed as described by the Synopsis and Description of each
84019 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,
84020 which must follow the command name without intervening <blank>s, and where it would
84021 otherwise be ambiguous. For example, *count* and *flag* arguments need not be <blank>-separated
84022 because "d22p" is not ambiguous, but *file* arguments to the *ex next* command must be separated
84023 by one or more <blank>s. Any <blank> in command arguments for the **abbreviate**,
84024 **unabbreviate**, **map**, and **unmap** commands can be <control>-V-escaped, in which case the
84025 <blank> shall not be used as an argument delimiter. Any <blank> in the command argument for
84026 any other command can be backslash-escaped, in which case that <blank> shall not be used as
84027 an argument delimiter.

84028 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,
84029 any character can be <control>-V-escaped. All such escaped characters shall be treated literally
84030 and shall have no special meaning. Within command arguments for all other *ex* commands that
84031 are not regular expressions or replacement strings, any character that would otherwise have a
84032 special meaning can be backslash-escaped. Escaped characters shall be treated literally, without
84033 special meaning as shell expansion characters or **'!'**, **'%'**, and **'#'** expansion characters. See
84034 [Regular Expressions in ex](#) (on page 2608) and [Replacement Strings in ex](#) (on page 2608) for
84035 descriptions of command arguments that are regular expressions or replacement strings.

84036 Non-backslash-escaped **'%'** characters appearing in *file* arguments to any *ex* command shall be
84037 replaced by the current pathname; unescaped **'#'** characters shall be replaced by the alternate
84038 pathname. It shall be an error if **'%'** or **'#'** characters appear unescaped in an argument and
84039 their corresponding values are not set.

84040 Non-backslash-escaped **'!'** characters in the arguments to either the *ex !* command or the open
84041 and visual mode **!** command, or in the arguments to the *ex read* command, where the first
84042 non-<blank> after the command name is a **'!'** character, or in the arguments to the *ex write*
84043 command where the command name is followed by one or more <blank>s and the first
84044 non-<blank> after the command name is a **'!'** character, shall be replaced with the arguments
84045 to the last of those three commands as they appeared after all unescaped **'%'**, **'#'**, and **'!'**
84046 characters were replaced. It shall be an error if **'!'** characters appear unescaped in one of these
84047 commands and there has been no previous execution of one of these commands.

84048 If an error occurs during the parsing or execution of an *ex* command:

- 84049 • An informational message to this effect shall be written. Execution of the *ex* command shall
84050 stop, and the cursor (for example, the current line and column) shall not be further
84051 modified.
- 84052 • If the *ex* command resulted from a map expansion, all characters from that map expansion
84053 shall be discarded, except as otherwise specified by the **map** command.
- 84054 • Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment
84055 variable, a **.exrc** file, a **:source** command, a **-c** option, or a **+command** specified to an *ex edit*,
84056 *ex next*, or *visual* command, no further commands from the source of the commands shall
84057 be executed.

- 84058 • Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v**
- 84059 command, no further commands caused by the execution of the buffer or the **global** or **v**
- 84060 command shall be executed.
- 84061 • Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and
- 84062 including the next non-backslash-escaped <newline> shall be discarded.

84063

Input Editing in *ex*

84064

The following symbol is used in this and the following sections to specify command actions:

84065

word In the POSIX locale, a word consists of a maximal sequence of letters, digits, and underscores, delimited at both ends by characters other than letters, digits, or underscores, or by the beginning or end of a line or the edit buffer.

84066

84067

84068

When accepting input characters from the user, in either *ex* command mode or *ex* text input mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces volume of POSIX.1-200x.

84069

84070

84071

If in *ex* text input mode:

84072

1. If the **number** edit option is set, *ex* shall prompt for input using the line number that would be assigned to the line if it is entered, in the format specified for the *ex* **number** command.

84073

84074

84075

2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters, as described by the **autoindent** edit option. **autoindent** characters shall follow the line number, if any.

84076

84077

84078

If in *ex* command mode:

84079

1. If the **prompt** edit option is set, input shall be prompted for using a single ' : ' character; otherwise, there shall be no prompt.

84080

84081

The input characters in the following sections shall have the following effects on the input line.

84082

Scroll

84083

Synopsis: eof

84084

See the description of the *stty* eof character in *stty*.

84085

If in *ex* command mode:

84086

If the eof character is the first character entered on the line, the line shall be evaluated as if it contained two characters: a <control>-D and a <newline>.

84087

84088

Otherwise, the eof character shall have no special meaning.

84089

If in *ex* text input mode:

84090 If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be
 84091 modified so that a part of the next text input character will be displayed on the first
 84092 column in the line after the previous **shiftwidth** edit option column boundary, and the
 84093 user shall be prompted again for input for the same line.

84094 Otherwise, if the cursor follows a '0', which follows an **autoindent** character, and the '0'
 84095 was the previous text input character, the '0' and all **autoindent** characters in the line
 84096 shall be discarded, and the user shall be prompted again for input for the same line.

84097 Otherwise, if the cursor follows a '^', which follows an **autoindent** character, and the '^'
 84098 was the previous text input character, the '^' and all **autoindent** characters in the line
 84099 shall be discarded, and the user shall be prompted again for input for the same line. In
 84100 addition, the **autoindent** level for the next input line shall be derived from the same line
 84101 from which the **autoindent** level for the current input line was derived.

84102 Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character
 84103 shall be discarded.

84104 Otherwise, the *eof* character shall have no special meaning.

84105 <newline>

84106 *Synopsis:* <newline>
 84107 <control>-J

84108 If in *ex* command mode:

84109 Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for
 84110 this purpose.

84111 If in *ex* text input mode:

84112 Terminate the current line. If there are no characters other than **autoindent** characters on
 84113 the line, all characters on the line shall be discarded.

84114 Prompt for text input on a new line after the current line. If the **autoindent** edit option is
 84115 set, an appropriate number of **autoindent** characters shall be added as a prefix to the line
 84116 as described by the *ex* **autoindent** edit option.

84117 <backslash>

84118 *Synopsis:* <backslash>

84119 Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any
 84120 special meaning that it may have to the editor during text input mode. The backslash character
 84121 shall be retained and evaluated when the command line is parsed, or retained and included
 84122 when the input text becomes part of the edit buffer.

84123 <control>-V

84124 *Synopsis:* <control>-V

84125 Allow the entry of any subsequent character as a literal character, removing any special meaning
 84126 that it may have to the editor during text input mode. The <control>-V character shall be
 84127 discarded before the command line is parsed or the input text becomes part of the edit buffer.

84128 If the "literal next" functionality is performed by the underlying system, it is implementation-
 84129 defined whether a character other than <control>-V performs this function.

84130

<control>-W

84131

Synopsis: <control>-W

84132

84133

84134

84135

Discard the <control>-W, and the word previous to it in the input line, including any <blank>s following the word and preceding the <control>-W. If the “word erase” functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-W performs this function.

84136

Command Descriptions in ex

84137

84138

The following symbols are used in this section to represent command modifiers. Some of these modifiers can be omitted, in which case the specified defaults shall be used.

84139

84140

1addr A single line address, given in any of the forms described in [Addressing in ex](#) (on page 2579); the default shall be the current line (‘.’), unless otherwise specified.

84141

84142

If the line address is zero, it shall be an error, unless otherwise specified in the following command descriptions.

84143

84144

84145

If the edit buffer is empty, and the address is specified with a command other than **=**, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be an error.

84146

84147

84148

84149

84150

2addr Two addresses specifying an inclusive range of lines. If no addresses are specified, the default for *2addr* shall be the current line only (“.”), unless otherwise specified in the following command descriptions. If one address is specified, *2addr* shall specify that line only, unless otherwise specified in the following command descriptions.

84151

It shall be an error if the first address is greater than the second address.

84152

84153

84154

If the edit buffer is empty, and the two addresses are specified with a command other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall be an error.

84155

84156

84157

84158

count A positive decimal number. If *count* is specified, it shall be equivalent to specifying an additional address to the command, unless otherwise specified by the following command descriptions. The additional address shall be equal to the last address specified to the command (either explicitly or by default) plus *count*−1.

84159

84160

If this would result in an address greater than the last line of the edit buffer, it shall be corrected to equal the last line of the edit buffer.

84161

84162

84163

84164

flags One or more of the characters ‘+’, ‘-’, ‘#’, ‘p’, or ‘l’ (ell). The flag characters can be <blank>-separated, and in any order or combination. The characters ‘#’, ‘p’, and ‘l’ shall cause lines to be written in the format specified by the **print** command with the specified *flags*.

84165

The lines to be written are as follows:

84166

84167

84168

84169

84170

1. All edit buffer lines written during the execution of the *ex* **&**, **~**, **list**, **number**, **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.
2. After the completion of an *ex* command with a flag as an argument, the current line shall be written as specified by *flags*, unless the current line was the last line written by the command.

84171

84172

84173

84174

The characters ‘+’ and ‘-’ cause the value of the current line after the execution of the *ex* command to be adjusted by the offset address as described in [Addressing in ex](#) (on page 2579). This adjustment shall occur before the current line is written as described in 2. above.

84175 The default for *flags* shall be none.

84176 *buffer* One of a number of named areas for holding text. The named buffers are specified
84177 by the alphanumeric characters of the POSIX locale. There shall also be one
84178 “unnamed” buffer. When no buffer is specified for editor commands that use a
84179 buffer, the unnamed buffer shall be used. Commands that store text into buffers
84180 shall store the text as it was before the command took effect, and shall store text
84181 occurring earlier in the file before text occurring later in the file, regardless of how
84182 the text region was specified. Commands that store text into buffers shall store the
84183 text into the unnamed buffer as well as any specified buffer.

84184 In *ex* commands, buffer names are specified as the name by itself. In open or visual
84185 mode commands the name is preceded by a double quote (' " ') character.

84186 If the specified buffer name is an uppercase character, and the buffer contents are
84187 to be modified, the buffer shall be appended to rather than being overwritten. If
84188 the buffer is not being modified, specifying the buffer name in lowercase and
84189 uppercase shall have identical results.

84190 There shall also be buffers named by the numbers 1 through 9. In open and visual
84191 mode, if a region of text including characters from more than a single line is being
84192 modified by the *vi* *c* or *d* commands, the motion character associated with the *c* or
84193 *d* commands specifies that the buffer text shall be in line mode, or the commands
84194 *%*, *'*, */*, *?*, *(*, *)*, *N*, *n*, *{*, or *}* are used to define a region of text for the *c* or *d*
84195 commands, the contents of buffers 1 through 8 shall be moved into the buffer named by the
84196 next numerically greater value, the contents of buffer 9 shall be discarded, and the
84197 region of text shall be copied into buffer 1. This shall be in addition to copying the
84198 text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can
84199 be specified as a source buffer for open and visual mode commands; however,
84200 specifying a numeric buffer as the write target of an open or visual mode
84201 command shall have unspecified results.

84202 The text of each buffer shall have the characteristic of being in either line or
84203 character mode. Appending text to a non-empty buffer shall set the mode to match
84204 the characteristic of the text being appended. Appending text to a buffer shall
84205 cause the creation of at least one additional line in the buffer. All text stored into
84206 buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers
84207 as the source of text specify individually how buffers of different modes are
84208 handled. Each open or visual mode command that uses buffers for any purpose
84209 specifies individually the mode of the text stored into the buffer and how buffers
84210 of different modes are handled.

84211 *file* Command text used to derive a pathname. The default shall be the current
84212 pathname, as defined previously, in which case, if no current pathname has yet
84213 been established it shall be an error, except where specifically noted in the
84214 individual command descriptions that follow. If the command text contains any of
84215 the characters *'* *~*, *'* *{*, *'* *[*, *'* ***, *'* *?*, *'* *\$*, *'* *'*, *'* *'*, *'* *"*, and *'* **, it shall be
84216 subjected to the process of “shell expansions”, as described below; if more than a
84217 single pathname results and the command expects only one, it shall be an error.

84218 The process of shell expansions in the editor shall be done as follows. The *ex* utility
84219 shall pass two arguments to the program named by the shell edit option; the first
84220 shall be *-c*, and the second shall be the string “echo” and the command text as a
84221 single argument. The standard output and standard error of that command shall
84222 replace the command text.

84223 ! A character that can be appended to the command name to modify its operation,
 84224 as detailed in the individual command descriptions. With the exception of the *ex*
 84225 **read**, **write**, and **!** commands, the '!' character shall only act as a modifier if there
 84226 are no <blank>s between it and the command name.

84227 *remembered search direction*

84228 The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in
 84229 the edit buffer based on a remembered search direction, which is initially unset,
 84230 and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* / and ? commands.

84231 Abbreviate

84232 *Synopsis:* ab[*breviate*][*lhs rhs*]

84233 If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

84234 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
 84235 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-
 84236 defined.

84237 In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character
 84238 shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

84239 In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a
 84240 <control>-V character is entered after a word character, a check shall be made for a set of
 84241 characters matching *lhs*, in the text input entered during this command. If it is found, the effect
 84242 shall be as if *rhs* was entered instead of *lhs*.

84243 The set of characters that are checked is defined as follows:

- 84244 1. If there are no characters inserted before the word and non-word or <ESC> characters
 84245 that triggered the check, the set of characters shall consist of the word character.
- 84246 2. If the character inserted before the word and non-word or <ESC> characters that
 84247 triggered the check is a word character, the set of characters shall consist of the characters
 84248 inserted immediately before the triggering characters that are word characters, plus the
 84249 triggering word character.
- 84250 3. If the character inserted before the word and non-word or <ESC> characters that
 84251 triggered the check is not a word character, the set of characters shall consist of the
 84252 characters that were inserted before the triggering characters that are neither <blank>s
 84253 nor word characters, plus the triggering word character.

84254 It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate**
 84255 commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the
 84256 effect of the command shall be as if the replacement had not occurred.

84257 *Current line:* Unchanged.

84258 *Current column:* Unchanged.

84259 Append

84260 *Synopsis:* [*laddr*] a[*ppend*][!]

84261 Enter *ex* text input mode; the input text shall be placed after the specified line. If line zero is
 84262 specified, the text shall be placed at the beginning of the edit buffer.

84263 This command shall be affected by the **number** and **autoindent** edit options; following the
 84264 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
 84265 duration of this command only.

84266 *Current line:* Set to the last input line; if no lines were input, set to the specified line, or to the first

84267 line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

84268 *Current column*: Set to non-<blank>.

84269 Arguments

84270 *Synopsis*: ar[gs]

84271 Write the current argument list, with the current argument-list entry, if any, between ' [' and
84272 '] ' characters.

84273 *Current line*: Unchanged.

84274 *Current column*: Unchanged.

84275 Change

84276 *Synopsis*: [2addr] c[hange][!][count]

84277 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall
84278 be copied into the unnamed buffer, which shall become a line mode buffer.

84279 This command shall be affected by the **number** and **autoindent** edit options; following the
84280 command name with ' ! ' shall cause the **autoindent** edit option setting to be toggled for the
84281 duration of this command only.

84282 *Current line*: Set to the last input line; if no lines were input, set to the line before the first
84283 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to
84284 zero if the edit buffer is empty.

84285 *Current column*: Set to non-<blank>.

84286 Change Directory

84287 *Synopsis*: chd[ir][!][directory]

84288 cd[!][directory]

84289 Change the current working directory to *directory*.

84290 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null
84291 and non-empty value, *directory* shall default to the value named in the *HOME* environment
84292 variable. If the *HOME* environment variable is empty or is undefined, the default value of
84293 *directory* is implementation-defined.

84294 If no ' ! ' is appended to the command name, and the edit buffer has been modified since the
84295 last complete write, and the current pathname does not begin with a ' / ', it shall be an error.

84296 *Current line*: Unchanged.

84297 *Current column*: Unchanged.

84298 Copy

84299 *Synopsis*: [2addr] co[py] 1addr [flags]

84300 [2addr] t 1addr [flags]

84301 Copy the specified lines after the specified destination line; line zero specifies that the lines shall
84302 be placed at the beginning of the edit buffer.

84303 *Current line*: Set to the last line copied.

84304 *Current column*: Set to non-<blank>.

84305 **Delete**
 84306 *Synopsis:* `[2addr] d[delete][buffer][count][flags]`

84307 Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a
 84308 line-mode buffer.

84309 Flags can immediately follow the command name; see [Command Line Parsing in ex](#) (on page
 84310 2580).

84311 *Current line:* Set to the line following the deleted lines, or to the last line in the edit buffer if that
 84312 line is past the end of the edit buffer, or to zero if the edit buffer is empty.

84313 *Current column:* Set to non-<blank>.

84314 **Edit**

84315 *Synopsis:* `e[dit][!][+command][file]`
 84316 `ex[!][+command][file]`

84317 If no '!' is appended to the command name, and the edit buffer has been modified since the
 84318 last complete write, it shall be an error.

84319 If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*,
 84320 and set the current pathname to *file*. If *file* is not specified, replace the current contents of the
 84321 edit buffer with the current contents of the file named by the current pathname. If for any reason
 84322 the current contents of the file cannot be accessed, the edit buffer shall be empty.

84323 The *+command* option shall be <blank>-delimited; <blank>s within *+command* can be escaped by
 84324 preceding them with a backslash character. The *+command* shall be interpreted as an *ex*
 84325 command immediately after the contents of the edit buffer have been replaced and the current
 84326 line and column have been set.

84327 If the edit buffer is empty:

84328 *Current line:* Set to 0.

84329 *Current column:* Set to 1.

84330 Otherwise, if executed while in *ex* command mode or if the *+command* argument is specified:

84331 *Current line:* Set to the last line of the edit buffer.

84332 *Current column:* Set to non-<blank>.

84333 Otherwise, if *file* is omitted or results in the current pathname:

84334 *Current line:* Set to the first line of the edit buffer.

84335 *Current column:* Set to non-<blank>.

84336 Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if
 84337 the file was previously edited, the line and column may be set as follows:

84338 *Current line:* Set to the last value held when that file was last edited. If this value is not a valid
 84339 line in the new edit buffer, set to the first line of the edit buffer.

84340 *Current column:* If the current line was set to the last value held when the file was last edited, set
 84341 to the last value held when the file was last edited. Otherwise, or if the last value is not a valid
 84342 column in the new edit buffer, set to non-<blank>.

84343 Otherwise:

84344 *Current line:* Set to the first line of the edit buffer.

84345 *Current column:* Set to non-<blank>.

84346 **File**84347 *Synopsis:* `f[file][file]`84348 If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and
84349 the current pathname shall be set to *file*.84350 Write an informational message. If the file has a current pathname, it shall be included in this
84351 message; otherwise, the message shall indicate that there is no current pathname. If the edit
84352 buffer contains lines, the current line number and the number of lines in the edit buffer shall be
84353 included in this message; otherwise, the message shall indicate that the edit buffer is empty. If
84354 the edit buffer has been modified since the last complete write, this fact shall be included in this
84355 message. If the **readonly** edit option is set, this fact shall be included in this message. The
84356 message may contain other unspecified information.84357 *Current line:* Unchanged.84358 *Current column:* Unchanged.84359 **Global**84360 *Synopsis:* `[2addr] g[lobal] /pattern/ [commands]`84361 `[2addr] v /pattern/ [commands]`84362 The optional **'!'** character after the **global** command shall be the same as executing the **v**
84363 command.84364 If *pattern* is empty (for example, `"/"/`) or not specified, the last regular expression used in the
84365 editor command shall be used as the *pattern*. The *pattern* can be delimited by slashes (shown in
84366 the Synopsis), as well as any non-alphanumeric or non-`<blank>` other than backslash, vertical
84367 line, double quote, or `<newline>`.

84368 If no lines are specified, the lines shall default to the entire file.

84369 The **global** and **v** commands are logically two-pass operations. First, mark the lines within the
84370 specified lines for which the line excluding the terminating `<newline>` matches (**global**) or does
84371 not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by
84372 *commands*, with the current line (`'.'`) set to each marked line. If an error occurs during this
84373 process, or the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an
84374 error message shall be written and no more commands resulting from the execution of this
84375 command shall be processed.84376 Multiple *ex* commands can be specified by entering multiple commands on a single line using a
84377 vertical line to delimit them, or one per line, by escaping each `<newline>` with a backslash.

84378 If no commands are specified:

- 84379
1. If in *ex* command mode, it shall be as if the **print** command were specified.
 2. Otherwise, no command shall be executed.
- 84380

84381 For the **append**, **change**, and **insert** commands, the input text shall be included as part of the
84382 command, and the terminating period can be omitted if the command ends the list of
84383 commands. The **open** and **visual** commands can be specified as one of the commands, in which
84384 case each marked line shall cause the editor to enter open or visual mode. If open or visual mode
84385 is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open
84386 or visual mode reentered, until the list of marked lines is exhausted.84387 The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted
84388 by commands executed for lines occurring earlier in the file than the marked lines. In this case,
84389 no commands shall be executed for the deleted lines.84390 If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

84391 The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v**
84392 command.

84393 *Current line*: If no commands executed, set to the last marked line. Otherwise, as specified for the
84394 executed *ex* commands.

84395 *Current column*: If no commands are executed, set to non-<blank>; otherwise, as specified for the
84396 individual *ex* commands.

84397 **Insert**

84398 *Synopsis*: `[1addr] i[nsert][!]`

84399 Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero
84400 or 1, the text shall be placed at the beginning of the edit buffer.

84401 This command shall be affected by the **number** and **autoindent** edit options; following the
84402 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
84403 duration of this command only.

84404 *Current line*: Set to the last input line; if no lines were input, set to the line before the specified
84405 line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero
84406 if the edit buffer is empty.

84407 *Current column*: Set to non-<blank>.

84408 **Join**

84409 *Synopsis*: `[2addr] j[oin][!][count][flags]`

84410 If *count* is specified:

84411 If no address was specified, the **join** command shall behave as if *2addr* were the current
84412 line and the current line plus *count* ($.. + count$).

84413 If one address was specified, the **join** command shall behave as if *2addr* were the specified
84414 address and the specified address plus *count* ($addr, addr + count$).

84415 If two addresses were specified, the **join** command shall behave as if an additional
84416 address, equal to the last address plus *count* -1 ($addr1, addr2, addr2 + count - 1$), was
84417 specified.

84418 If this would result in a second address greater than the last line of the edit buffer, it shall
84419 be corrected to be equal to the last line of the edit buffer.

84420 If no *count* is specified:

84421 If no address was specified, the **join** command shall behave as if *2addr* were the current
84422 line and the next line ($.. + 1$).

84423 If one address was specified, the **join** command shall behave as if *2addr* were the specified
84424 address and the next line ($addr, addr + 1$).

84425 Join the text from the specified lines together into a single line, which shall replace the specified
84426 lines.

84427 If a '!' character is appended to the command name, the **join** shall be without modification of
84428 any line, independent of the current locale.

84429 Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for
84430 each subsequent line, proceed as follows:

- 84431 1. Discard leading <space>s from the line to be joined.
- 84432 2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
- 84433 3. If the current line ends in a <blank>, or the first character of the line to be joined is a ') ' character, join the lines without further modification.
- 84434
- 84435 4. If the last character of the current line is a ' . ' , join the lines with two <space>s between
- 84436 them.
- 84437 5. Otherwise, join the lines with a single <space> between them.

84438 *Current line*: Set to the first line specified.

84439 *Current column*: Set to non-<blank>.

84440 **List**

84441 *Synopsis*: `[2addr] l[ist][count][flags]`

84442 This command shall be equivalent to the *ex* command:

84443 `[2addr] p[rint][count] l[flags]`

84444 See [Print](#) (on page 2597).

84445 **Map**

84446 *Synopsis*: `map[!][lhs rhs]`

84447 If *lhs* and *rhs* are not specified:

- 84448 1. If ' ! ' is specified, write the current list of text input mode maps.
- 84449 2. Otherwise, write the current list of command mode maps.
- 84450 3. Do nothing more.

84451 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

84452

84453

84454 If the character ' ! ' is appended to the **map** command name, the mapping shall be effective during open or visual text input mode rather than **open** or **visual** command mode. This allows *lhs* to have two different **map** definitions at the same time: one for command mode and one for text input mode.

84455

84460 For command mode mappings:

84461 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as
84462 part of the arguments to the command), the action shall be as if the corresponding *rhs* had
84463 been entered.

84464 If any character in the command, other than the first, is escaped using a <control>-V
84465 character, that character shall not be part of a match to an *lhs*.

84466 It is unspecified whether implementations shall support **map** commands where the *lhs* is
84467 more than a single character in length, where the first character of the *lhs* is printable.

84468 If *lhs* contains more than one character and the first character is '#', followed by a
84469 sequence of digits corresponding to a numbered function key, then when this function key
84470 is typed it shall be mapped to *rhs*. Characters other than digits following a '#' character
84471 also represent the function key named by the characters in the *lhs* following the '#' and
84472 may be mapped to *rhs*. It is unspecified how function keys are named or what function
84473 keys are supported.

84474 For text input mode mappings:

84475 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
84476 action shall be as if the corresponding *rhs* had been entered.

84477 If any character in the input text is escaped using a <control>-V character, that character
84478 shall not be part of a match to an *lhs*.

84479 It is unspecified whether the *lhs* text entered for subsequent **map** or **unmap** commands is
84480 replaced with the *rhs* text for the purposes of the screen display; regardless of whether or
84481 not the display appears as if the corresponding *rhs* text was entered, the effect of the
84482 command shall be as if the *lhs* text was entered.

84483 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
84484 possibly matching characters before treating the already entered characters as not matching the
84485 *lhs*.

84486 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
84487 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
84488 characters shall not be remapped.

84489 On block-mode terminals, the mapping need not occur immediately (for example, it may occur
84490 after the terminal transmits a group of characters to the system), but it shall achieve the same
84491 results as if it occurred immediately.

84492 *Current line*: Unchanged.

84493 *Current column*: Unchanged.

84494 **Mark**

84495 *Synopsis*: `[laddr] ma[rk] character`

84496 `[laddr] k character`

84497 Implementations shall support *character* values of a single lowercase letter of the POSIX locale
84498 and the characters '' and '''; support of other characters is implementation-defined.

84499 If executing the *vi m* command, set the specified mark to the current line and 1-based numbered
84500 character referenced by the current column, if any; otherwise, column position 1.

84501 Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank>
84502 non-<newline> in the line, if any; otherwise, the last non-<newline> in the line, if any;
84503 otherwise, column position 1.

84504 The mark shall remain associated with the line until the mark is reset or the line is deleted. If a

84505 deleted line is restored by a subsequent **undo** command, any marks previously associated with
 84506 the line, which have not been reset, shall be restored as well. Any use of a mark not associated
 84507 with a current line in the edit buffer shall be an error.

84508 The marks ' and ' shall be set as described previously, immediately before the following events
 84509 occur in the editor:

- 84510 1. The use of '\$' as an *ex* address
- 84511 2. The use of a positive decimal number as an *ex* address
- 84512 3. The use of a search command as an *ex* address
- 84513 4. The use of a mark reference as an *ex* address
- 84514 5. The use of the following open and visual mode commands: <control>-], %, (,), [,], {, }
- 84515 6. The use of the following open and visual mode commands: ', **G**, **H**, **L**, **M**, **z** if the current
 84516 line will change as a result of the command
- 84517 7. The use of the open and visual mode commands: /, ?, **N**, ', **n** if the current line or column
 84518 will change as a result of the command
- 84519 8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

84520 For rules 1., 2., 3., and 4., the ' and ' marks shall not be set if the *ex* command is parsed as
 84521 specified by rule 6.a. in [Command Line Parsing in ex](#) (on page 2580).

84522 For rules 5., 6., and 7., the ' and ' marks shall not be set if the commands are used as motion
 84523 commands in open and visual mode.

84524 For rules 1., 2., 3., 4., 5., 6., 7., and 8., the ' and ' marks shall not be set if the command fails.

84525 The ' and ' marks shall be set as described previously, each time the contents of the edit buffer
 84526 are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex**
 84527 mode and the edit buffer is not empty, before any commands or movements (including
 84528 commands or movements specified by the **-c** or **-t** options or the **+command** argument) are
 84529 executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the **vi**
 84530 **m** command; otherwise, as if executing the **ex mark** command.

84531 When changing from **ex** mode to open or visual mode, if the ' and ' marks are not already set,
 84532 the ' and ' marks shall be set as described previously.

84533 *Current line*: Unchanged.

84534 *Current column*: Unchanged.

84535 **Move**

84536 *Synopsis*: `[2addr] m[ove] 1addr [flags]`

84537 Move the specified lines after the specified destination line. A destination of line zero specifies
 84538 that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the
 84539 destination line is within the range of lines to be moved.

84540 *Current line*: Set to the last of the moved lines.

84541 *Current column*: Set to non-<blank>.

84542

Next

84543

Synopsis: n[ext][!][+command][file ...]

84544

84545

84546

If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

84547

If one or more files is specified:

84548

1. Set the argument list to the specified filenames.

84549

2. Set the current argument list reference to be the first entry in the argument list.

84550

3. Set the current pathname to the first filename specified.

84551

Otherwise:

84552

1. It shall be an error if there are no more filenames in the argument list after the filename currently referenced.

84553

84554

2. Set the current pathname and the current argument list reference to the filename after the filename currently referenced in the argument list.

84555

84556

84557

84558

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

84559

This command shall be affected by the **autowrite** and **wriean** edit options.

84560

84561

84562

84563

The `+command` option shall be `<blank>`-delimited; `<blank>`s can be escaped by preceding them with a backslash character. The `+command` shall be interpreted as an `ex` command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

84564

Current line: Set as described for the **edit** command.

84565

Current column: Set as described for the **edit** command.

84566

Number

84567

Synopsis: [2addr] nu[mber][count][flags]

84568

[2addr] #[count][flags]

84569

These commands shall be equivalent to the `ex` command:

84570

[2addr] p[rint][count] #[flags]

84571

See [Print](#) (on page 2597).

84572

Open

84573

Synopsis: [1addr] o[pen] /pattern/ [flags]

84574

84575

84576

This command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

84577

Enter open mode.

84578

84579

84580

84581

84582

The trailing delimiter can be omitted from `pattern` at the end of the command line. If `pattern` is empty (for example, `"/ /"`) or not specified, the last regular expression used in the editor shall be used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis), as well as any alphanumeric, or non-`<blank>` other than backslash, vertical line, double quote, or `<newline>`.

84583 *Current line*: Set to the specified line.

84584 *Current column*: Set to non-<blank>.

84585 **Preserve**

84586 *Synopsis*: `pre[serve]`

84587 Save the edit buffer in a form that can later be recovered by using the `-r` option or by using the
84588 *ex* **recover** command. After the file has been preserved, a mail message shall be sent to the user.
84589 This message shall be readable by invoking the *mailx* utility. The message shall contain the name
84590 of the file, the time of preservation, and an *ex* command that could be used to recover the file.
84591 Additional information may be included in the mail message.

84592 *Current line*: Unchanged.

84593 *Current column*: Unchanged.

84594 **Print**

84595 *Synopsis*: `[2addr] p[rint][count][flags]`

84596 Write the addressed lines. The behavior is unspecified if the number of columns on the display is
84597 less than the number of columns required to write any single character in the lines being written.

84598 Non-printable characters, except for the <tab>, shall be written as implementation-defined
84599 multi-character sequences.

84600 If the # flag is specified or the **number** edit option is set, each line shall be preceded by its line
84601 number in the following format:

84602 `"%6dΔΔ", <line number>`

84603 If the l flag is specified or the **list** edit option is set:

- 84604 1. The characters listed in XBD Table 5-1 (on page 108) shall be written as the corresponding
84605 escape sequence.
- 84606 2. Non-printable characters not in XBD Table 5-1 (on page 108) shall be written as one three-
84607 digit octal number (with a preceding backslash) for each byte in the character (most
84608 significant byte first).
- 84609 3. The end of each line shall be marked with a '\$', and literal '\$' characters within the line
84610 shall be written with a preceding backslash.

84611 Long lines shall be folded; the length at which folding occurs is unspecified, but should be
84612 appropriate for the output terminal, considering the number of columns of the terminal.

84613 If a line is folded, and the l flag is not specified and the **list** edit option is not set, it is unspecified
84614 whether a multi-column character at the folding position is separated; it shall not be discarded.

84615 *Current line*: Set to the last written line.

84616 *Current column*: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

84617

Put

84618

Synopsis: [laddr] pu[t][buffer]

84619

Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

84620

84621

84622

Current line: Set to the last line entered into the edit buffer.

84623

Current column: Set to non-<blank>.

84624

Quit

84625

Synopsis: q[uit][!]

84626

If no '!' is appended to the command name:

84627

1. If the edit buffer has been modified since the last complete write, it shall be an error.

84628

2. If there are filenames in the argument list after the filename currently referenced, and the last command was not a **quit**, **wq**, **xit**, or **ZZ** (see [Exit](#), on page 3248) command, it shall be an error.

84629

84630

84631

Otherwise, terminate the editing session.

84632

Read

84633

Synopsis: [laddr] r[ead][!][file]

84634

If '!' is not the first non-<blank> to follow the command name, a copy of the specified file shall be appended into the edit buffer after the specified line; line zero specifies that the copy shall be placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If no *file* is named, the current pathname shall be the default. If there is no current pathname, then *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be an error. Specifying a *file* that is not of type regular shall have unspecified results.

84635

84636

84637

84638

84639

84640

Otherwise, if *file* is preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters expanded as described in [Command Line Parsing in ex](#) (on page 2580).

84641

84642

The *ex* utility shall then pass two arguments to the program named by the shell edit option; the first shall be **-c** and the second shall be the expanded arguments to the **read** command as a single argument. The standard input of the program shall be set to the standard input of the *ex* program when it was invoked. The standard error and standard output of the program shall be appended into the edit buffer after the specified line.

84643

84644

84645

84646

84647

Each line in the copied file or program output (as delimited by <newline>s or the end of the file or output if it is not immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s.

84648

84649

84650

84651

The special meaning of the '!' following the **read** command can be overridden by escaping it with a backslash character.

84652

84653

Current line: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into the edit buffer.

84654

84655

84656

Current column: Set to non-<blank>.

84657

Recover

84658

Synopsis: `rec[over][!] file`

84659

If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

84660

84661

If no *file* operand is specified, then the current pathname shall be used. If there is no current pathname or *file* operand, it shall be an error.

84662

84663

If no recovery information has previously been saved about *file*, the **recover** command shall behave identically to the **edit** command, and an informational message to this effect shall be written.

84664

84665

84666

Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. The editor shall behave as if the contents of the edit buffer have already been modified.

84667

84668

84669

84670

84671

Current file: Set as described for the **edit** command.

84672

Current column: Set as described for the **edit** command.

84673

Rewind

84674

Synopsis: `rew[ind][!]`

84675

If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

84676

84677

84678

If the argument list is empty, it shall be an error.

84679

The current argument list reference and the current pathname shall be set to the first filename in the argument list.

84680

84681

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

84682

84683

84684

This command shall be affected by the **autowrite** and **wriean** edit options.

84685

Current line: Set as described for the **edit** command.

84686

Current column: Set as described for the **edit** command.

84687

Set

84688

Synopsis: `se[t][option[=value]] ...][nooption ...][option? ...][all]`

84689

When no arguments are specified, write the value of the **term** edit option and those options whose values have been changed from the default settings; when the argument *all* is specified, write all of the option values.

84690

84691

84692

Giving an option name followed by the character `'?'` shall cause the current value of that option to be written. The `'?'` can be separated from the option name by zero or more `<blank>`s. The `'?'` shall be necessary only for Boolean valued options. Boolean options can be given values by the form **set option** to turn them on or **set nooption** to turn them off; string and numeric options can be assigned by the form **set option=*value***. Any `<blank>`s in strings can be included as is by preceding each `<blank>` with an escaping backslash. More than one option can be set or listed by a single set command by specifying multiple arguments, each separated from the next by one or more `<blank>`s.

84693

84694

84695

84696

84697

84698

84699

84700 See [Edit Options in ex](#) (on page 2609) for details about specific options.

84701 *Current line*: Unchanged.

84702 *Current column*: Unchanged.

84703 Shell

84704 *Synopsis*: `sh[ell]`

84705 Invoke the program named in the **shell** edit option with the single argument **-i** (interactive
84706 mode). Editing shall be resumed when the program exits.

84707 *Current line*: Unchanged.

84708 *Current column*: Unchanged.

84709 Source

84710 *Synopsis*: `so[urce] file`

84711 Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

84712 *Current line*: As specified for the individual *ex* commands.

84713 *Current column*: As specified for the individual *ex* commands.

84714 Substitute

84715 *Synopsis*: `[2addr] s[substitute][/pattern/repl][options][count][flags]`

84716 `[2addr] &[options][count][flags]`

84717 `[2addr] ~[options][count][flags]`

84718 Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See
84719 [Regular Expressions in ex](#) (on page 2608) and [Replacement Strings in ex](#) (on page 2608).) Any
84720 non-alphabetic, non-`<blank>` delimiter other than `'\'`, `'|'`, double quote, or `<newline>` can be
84721 used instead of `'/'`. Backslash characters can be used to escape delimiters, backslash
84722 characters, and other special characters.

84723 The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If
84724 both *pattern* and *repl* are not specified or are empty (for example, `"/"/`), the last **s** command
84725 shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in
84726 the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be
84727 replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an
84728 **s** command shall be used.

84729 Entering a `<carriage-return>` in *repl* (which requires an escaping backslash in *ex* mode and an
84730 escaping `<control>-V` in open or *vi* mode) shall split the line at that point, creating a new line in
84731 the edit buffer. The `<carriage-return>` shall be discarded.

84732 If *options* includes the letter **'g'** (**global**), all non-overlapping instances of the pattern in the line
84733 shall be replaced.

84734 If *options* includes the letter **'c'** (**confirm**), then before each substitution the line shall be written;
84735 the written line shall reflect all previous substitutions. On the following line, `<space>`s shall be
84736 written beneath the characters from the line that are before the *pattern* to be replaced, and `'^'`
84737 characters written beneath the characters included in the *pattern* to be replaced. The *ex* utility
84738 shall then wait for a response from the user. An affirmative response shall cause the substitution
84739 to be done, while any other input shall not make the substitution. An affirmative response shall
84740 consist of a line with the affirmative response (as defined by the current locale) at the beginning
84741 of the line. This line shall be subject to editing in the same way as the *ex* command line.

84742 If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the

84743 user shall be preserved in the edit buffer after the interrupt.

84744 If the remembered search direction is not set, the **s** command shall set it to forward.

84745 In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&**
84746 command were replaced by:

84747 *s/pattern/repl/*

84748 where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

84749 In the third Synopsis, the **~** command shall repeat the previous substitution, as if the '**~**' were
84750 replaced by:

84751 *s/pattern/repl/*

84752 where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from
84753 the previous substitution (including **&** and **~**) command.

84754 These commands shall be affected by the *LC_MESSAGES* environment variable.

84755 *Current line*: Set to the last line in which a substitution occurred, or, unchanged if no substitution
84756 occurred.

84757 *Current column*: Set to non-<blank>.

84758 Suspend

84759 *Synopsis:* **su**[*suspend*][!]

84760 **st**[*op*][!]

84761 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the
84762 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell
84763 (see the description of **set -m**).

84764 These commands shall be affected by the **autowrite** and **writeany** edit options.

84765 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

84766 Tag

84767 *Synopsis:* **ta**[*g*][!] *tagstring*

84768 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see
84769 *ctags*) description.

84770 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in
84771 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from
84772 beginning to end. If no reference is found, it shall be an error and an error message to this effect
84773 shall be written. If the reference is not found, or if an error occurs while processing a file referred
84774 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first
84775 occurrence of such an error.

84776 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression
84777 used in the editor; for example, for the purposes of the **s** command.

84778 If the *tagstring* is in a file with a different name than the current pathname, set the current
84779 pathname to the name of that file, and replace the contents of the edit buffer with the contents of
84780 that file. In this case, if no '**!**' is appended to the command name, and the edit buffer has been
84781 modified since the last complete write, it shall be an error, unless the file is successfully written
84782 as specified by the **autowrite** option.

84783 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

84784 *Current line*: If the tags file contained a line number, set to that line number. If the line number is

84785 larger than the last line in the edit buffer, an error message shall be written and the current line
84786 shall be set as specified for the **edit** command.

84787 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no
84788 matching pattern is found, an error message shall be written and the current line shall be set as
84789 specified for the **edit** command.

84790 *Current column*: If the tags file contained a line-number reference and that line-number was not
84791 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern
84792 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

84793 **Unabbreviate**

84794 *Synopsis*: `una[bbrev] lhs`

84795 If *lhs* is not an entry in the current list of abbreviations (see [Abbreviate](#), on page 2588), it shall be
84796 an error. Otherwise, delete *lhs* from the list of abbreviations.

84797 *Current line*: Unchanged.

84798 *Current column*: Unchanged.

84799 **Undo**

84800 *Synopsis*: `u[ndo]`

84801 Reverse the changes made by the last command that modified the contents of the edit buffer,
84802 including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands
84803 resulting from buffer executions and mapped character expansions, are considered single
84804 commands.

84805 If no action that can be undone preceded the **undo** command, it shall be an error.

84806 If the **undo** command restores lines that were marked, the mark shall also be restored unless it
84807 was reset subsequent to the deletion of the lines.

84808 *Current line*:

- 84809 1. If lines are added or changed in the file, set to the first line added or changed.
- 84810 2. Set to the line before the first line deleted, if it exists.
- 84811 3. Set to 1 if the edit buffer is not empty.
- 84812 4. Set to zero.

84813 *Current column*: Set to non-<blank>.

84814 **Unmap**

84815 *Synopsis*: `unm[ap][!] lhs`

84816 If '!' is appended to the command name, and if *lhs* is not an entry in the list of text input mode
84817 map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map
84818 definitions.

84819 If no '!' is appended to the command name, and if *lhs* is not an entry in the list of command
84820 mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode
84821 map definitions.

84822 *Current line*: Unchanged.

84823 *Current column*: Unchanged.

84824

Version

84825

Synopsis: `ve[rsion]`

84826

Write a message containing version information for the editor. The format of the message is unspecified.

84827

84828

Current line: Unchanged.

84829

Current column: Unchanged.

84830

Visual

84831

Synopsis: `[laddr] vi[sual][type][count][flags]`

84832

If *ex* is currently in open or visual mode, the *Synopsis* and behavior of the visual command shall be the same as the **edit** command, as specified by **Edit** (on page 2590).

84833

84834

Otherwise, this command need not be supported on block-mode terminals or terminals with insufficient capabilities. If standard input, standard output, or standard error are not terminal devices, the results are unspecified.

84835

84836

84837

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window**, on page 2615). If the '^' type character was also specified, the **window** edit option shall be set before being used by the type character.

84838

84839

84840

Enter visual mode. If *type* is not specified, it shall be as if a *type* of '+' was specified. The *type* shall cause the following effects:

84841

84842

+ Place the beginning of the specified line at the top of the display.

84843

- Place the end of the specified line at the bottom of the display.

84844

. Place the beginning of the specified line in the middle of the display.

84845

^ If the specified line is less than or equal to the value of the **window** edit option, set the line to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place the beginning of this line as close to the bottom of the displayed lines as possible, while still displaying the value of the **window** edit option number of lines.

84846

84847

84848

84849

Current line: Set to the specified line.

84850

Current column: Set to non-<blank>.

84851

Write

84852

Synopsis: `[2addr] w[rite][!][>>][file]`

84853

`[2addr] w[rite][!][file]`

84854

`[2addr] wq[!][>>][file]`

84855

If no lines are specified, the lines shall default to the entire file.

84856

The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!** shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the **quit** shall not be attempted.

84857

84858

84859

If the command name is not followed by one or more <blank>s, or *file* is not preceded by a '!' character, the **write** shall be to a file.

84860

84861

1. If the >> argument is specified, and the file already exists, the lines shall be appended to the file instead of replacing its contents. If the >> argument is specified, and the file does not already exist, it is unspecified whether the write shall proceed as if the >> argument had not been specified or if the write shall fail.

84862

84863

84864

- 84865 2. If the **readonly** edit option is set (see [readonly](#), on page 2612), the **write** shall fail.
- 84866 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.
- 84867 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
84868 the **write** command shall fail.
- 84869 5. If the current pathname is used, and the current pathname has been changed by the **file**
84870 or **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,
84871 subsequent **writes** shall not fail for this reason (unless the current pathname is changed
84872 again).
- 84873 6. If the whole edit buffer is not being written, and the file to be written exists, the **write**
84874 shall fail.

84875 For rules 1., 2., 3., and 5., the **write** can be forced by appending the character '**!**' to the
84876 command name.

84877 For rules 2., 3., and 5., the **write** can be forced by setting the **wri~~te~~any** edit option.

84878 Additional, implementation-defined tests may cause the **write** to fail.

84879 If the edit buffer is empty, a file without any contents shall be written.

84880 An informational message shall be written noting the number of lines and bytes written.

84881 Otherwise, if the command is followed by one or more <blank>s, and the file is preceded by
84882 '**!**', the rest of the line after the '**!**' shall have '%', '#', and '**!**' characters expanded as
84883 described in [Command Line Parsing in ex](#) (on page 2580).

84884 The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the
84885 first shall be **-c** and the second shall be the expanded arguments to the **write** command as a
84886 single argument. The specified lines shall be written to the standard input of the command. The
84887 standard error and standard output of the program, if any, shall be written as described for the
84888 **print** command. If the last character in that output is not a <newline>, a <newline> shall be
84889 written at the end of the output.

84890 The special meaning of the '**!**' following the **write** command can be overridden by escaping it
84891 with a backslash character.

84892 *Current line*: Unchanged.

84893 *Current column*: Unchanged.

84894 **Write and Exit**

84895 *Synopsis*: [*2addr*] **x[it][!][file]**

84896 If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to
84897 the **quit** command, or if a '**!**' is appended to the command name, to **quit!**.

84898 Otherwise, **xit** shall be equivalent to the **wq** command, or if a '**!**' is appended to the command
84899 name, to **wq!**.

84900 *Current line*: Unchanged.

84901 *Current column*: Unchanged.

84902 **Yank**84903 *Synopsis:* `[2addr] ya[nk][buffer][count]`84904 Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall
84905 become a line-mode buffer.84906 *Current line:* Unchanged.84907 *Current column:* Unchanged.84908 **Adjust Window**84909 *Synopsis:* `[1addr] z[!][type ...][count][flags]`84910 If no line is specified, the current line shall be the default; if *type* is omitted as well, the current
84911 line value shall first be incremented by 1. If incrementing the current line would cause it to be
84912 greater than the last line in the edit buffer, it shall be an error.84913 If there are <blank>s between the *type* argument and the preceding *z* command name or
84914 optional '!' character, it shall be an error.84915 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
84916 **window**, on page 2615). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit
84917 option, or if ! was specified, the number of lines in the display minus 1.84918 If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise,
84919 *count* lines starting with the line specified by the *type* argument shall be written.84920 The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

84921 – The specified line shall be decremented by the following value:

84922 $((\text{number of ``-'' characters}) \times \text{count}) - 1$ 84923 If the calculation would result in a number less than 1, it shall be an error. Write lines from
84924 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
84925 buffer has been written.

84926 + The specified line shall be incremented by the following value:

84927 $((\text{number of ``+'' characters}) - 1) \times \text{count} + 1$ 84928 If the calculation would result in a number greater than the last line in the edit buffer, it
84929 shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count*
84930 lines or the last line in the edit buffer has been written.84931 =, . If more than a single '.' or '=' is specified, it shall be an error. The following steps shall
84932 be taken:84933 1. If *count* is zero, nothing shall be written.84934 2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count*
84935 or '!' was specified, *N* shall be:84936 $(\text{count} - 1) / 2$ 84937 Otherwise, *N* shall be:84938 $(\text{count} - 3) / 2$ 84939 If *N* is a number less than 3, no lines shall be written.84940 3. If '=' was specified as the type character, write a line consisting of the smaller of the
84941 number of columns in the display divided by two, or 40 '-' characters.

- 84942 4. Write the current line.
- 84943 5. Repeat step 3.
- 84944 6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall
- 84945 be defined as in step 2. If *N* is a number less than 3, no lines shall be written. If *count*
- 84946 is less than 3, no lines shall be written.

84947 \wedge The specified line shall be decremented by the following value:

84948 $((\text{number of ``^`` characters}) + 1) \times \text{count} - 1$

84949 If the calculation would result in a number less than 1, it shall be an error. Write lines from

84950 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit

84951 buffer has been written.

84952 *Current line*: Set to the last line written, unless the type is =, in which case, set to the specified

84953 line.

84954 *Current column*: Set to non-<blank>.

84955 Escape

84956 *Synopsis*: ! *command*

84957 [*addr*] ! *command*

84958 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as

84959 described in [Command Line Parsing in ex](#) (on page 2580). If the expansion causes the text of the

84960 line to change, it shall be redisplayed, preceded by a single '!' character.

84961 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two

84962 arguments to the program; the first shall be **-c**, and the second shall be the expanded arguments

84963 to the ! command as a single argument.

84964 If no lines are specified, the standard input, standard output, and standard error of the program

84965 shall be set to the standard input, standard output, and standard error of the *ex* program when it

84966 was invoked. In addition, a warning message shall be written if the edit buffer has been

84967 modified since the last complete write, and the **warn** edit option is set.

84968 If lines are specified, they shall be passed to the program as standard input, and the standard

84969 output and standard error of the program shall replace those lines in the edit buffer. Each line in

84970 the program output (as delimited by <newline>s or the end of the output if it is not immediately

84971 preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of

84972 <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s. The

84973 specified lines shall be copied into the unnamed buffer before they are replaced, and the

84974 unnamed buffer shall become a line-mode buffer.

84975 If in *ex* mode, a single '!' character shall be written when the program completes.

84976 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this

84977 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this

84978 command shall be affected by the **autoprint** edit option.

84979 *Current line*:

- 84980 1. If no lines are specified, unchanged.
- 84981 2. Otherwise, set to the last line read in, if any lines are read in.
- 84982 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
- 84983 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.

84984 5. Otherwise, set to zero.

84985 *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.

84986 **Shift Left**

84987 *Synopsis*: `[2addr] <[< ...][count][flags]`

84988 Shift the specified lines to the start of the line; the number of column positions to be shifted shall
84989 be the number of command characters times the value of the **shiftwidth** edit option. Only
84990 leading <blank>s shall be deleted or changed into other <blank>s in shifting; other characters
84991 shall not be affected.

84992 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
84993 buffer.

84994 This command shall be affected by the **autoprint** edit option.

84995 *Current line*: Set to the last line in the lines specified.

84996 *Current column*: Set to non-<blank>.

84997 **Shift Right**

84998 *Synopsis*: `[2addr] >[> ...][count][flags]`

84999 Shift the specified lines away from the start of the line; the number of column positions to be
85000 shifted shall be the number of command characters times the value of the **shiftwidth** edit
85001 option. The shift shall be accomplished by adding <blank>s as a prefix to the line or changing
85002 leading <blank>s into other <blank>s. Empty lines shall not be changed.

85003 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
85004 buffer.

85005 This command shall be affected by the **autoprint** edit option.

85006 *Current line*: Set to the last line in the lines specified.

85007 *Current column*: Set to non-<blank>.

85008 **<control>-D**

85009 *Synopsis*: `<control>-D`

85010 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the
85011 number of lines after the current line in the edit buffer. If the current line is the last line of the
85012 edit buffer it shall be an error.

85013 *Current line*: Set to the last line written.

85014 *Current column*: Set to non-<blank>.

85015 **Write Line Number**

85016 *Synopsis*: `[1addr] = [flags]`

85017 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of
85018 the specified line.

85019 *Current line*: Unchanged.

85020 *Current column*: Unchanged.

85021

Execute

85022

Synopsis: [2addr] @ buffer

85023

[2addr] * buffer

85024

If no buffer is specified or is specified as '@' or '*', the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

85025

85026

For each line specified by the addresses, set the current line ('.') to the specified line, and execute the contents of the named *buffer* (as they were at the time the @ command was executed) as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.

85027

85028

85029

85030

If an error occurs during this process, or a line specified by the addresses does not exist when the current line would be set to it, or more than a single line was specified by the addresses, and the contents of the edit buffer are replaced (for example, by the *ex* :edit command) an error message shall be written, and no more commands resulting from the execution of this command shall be processed.

85031

85032

85033

85034

85035

Current line: As specified for the individual *ex* commands.

85036

Current column: As specified for the individual *ex* commands.

85037

Regular Expressions in ex

85038

The *ex* utility shall support regular expressions that are a superset of the basic regular expressions described in XBD [Section 9.3](#) (on page 169). A null regular expression ("//") shall be equivalent to the last regular expression encountered.

85039

85040

85041

Regular expressions can be used in addresses to specify lines and, in some commands (for example, the **substitute** command), to specify portions of a line to be substituted.

85042

85043

The following constructs can be used to enhance the basic regular expressions:

85044

\< Match the beginning of a *word*. (See the definition of *word* at the beginning of [Command Descriptions in ex](#) (on page 2586).)

85045

85046

\> Match the end of a *word*.

85047

~ Match the replacement part of the last **substitute** command. The tilde ('~') character can be escaped in a regular expression to become a normal character with no special meaning. The backslash shall be discarded.

85048

85049

85050

When the editor option **magic** is not set, the only characters with special meanings shall be '^' at the beginning of a pattern, '\$' at the end of a pattern, and '\'. The characters '.', '*', '[', and '~' shall be treated as ordinary characters unless preceded by a '\'; when preceded by a '\' they shall regain their special meaning, or in the case of backslash, be handled as a single backslash. Backslashes used to escape other characters shall be discarded.

85051

85052

85053

85054

85055

Replacement Strings in ex

85056

The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not set) shall be replaced by the replacement part of the previous **substitute** command. The sequence '\n', where *n* is an integer, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters '\n' shall be replaced by the empty string.

85057

85058

85059

85060

85061

85062

The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the replacement string (using the '&' or "\"digit) notation. The string '\l' ('\u') shall cause the character that follows to be converted to lowercase (uppercase). The string '\L' ('\U') shall cause all characters subsequent to it to be converted to lowercase (uppercase) as they are

85063

85064

85065

85066 inserted by the substitution until the string '`\e`' or '`\E`', or the end of the replacement string,
85067 is encountered.

85068 Otherwise, any character following a backslash shall be treated as that literal character, and the
85069 escaping backslash shall be discarded.

85070 An example of case conversion with the `s` command is as follows:

```
85071 :p
85072 The cat sat on the mat.
85073 :s/\<.at\>/\u&/gp
85074 The Cat Sat on the Mat.
85075 :s/S\(.*\)M/S\U\1\eM/p
85076 The Cat SAT ON THE Mat.
```

85077 Edit Options in ex

85078 The `ex` utility has a number of options that modify its behavior. These options have default
85079 settings, which can be changed using the `set` command.

85080 Options are Boolean unless otherwise specified.

85081 **autoindent, ai**

85082 [Default *unset*]

85083 If **autoindent** is set, each line in input mode shall be indented (using first as many `<tab>`s as
85084 possible, as determined by the editor option **tabstop**, and then using `<space>`s) to align with
85085 another line, as follows:

- 85086 1. If in open or visual mode and the text input is part of a line-oriented command (see the
85087 EXTENDED DESCRIPTION in *vi*), align to the first column.
- 85088 2. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
 - 85089 a. If a line was previously inserted as part of this command, it shall be set to the
85090 indentation of the last inserted line by default, or as otherwise specified for the
85091 `<control>-D` character in *Input Mode Commands in vi* (on page 3248).
 - 85092 b. Otherwise, it shall be set to the indentation of the previous current line, if any;
85093 otherwise, to the first column.
- 85094 3. For the `ex a`, `i`, and `c` commands, indentation for each line shall be set as follows:
 - 85095 a. If a line was previously inserted as part of this command, it shall be set to the
85096 indentation of the last inserted line by default, or as otherwise specified for the `eof`
85097 character in *Scroll* (on page 2584).
 - 85098 b. Otherwise, if the command is the `ex a` command, it shall be set to the line
85099 appended after, if any; otherwise to the first column.
 - 85100 c. Otherwise, if the command is the `ex i` command, it shall be set to the line inserted
85101 before, if any; otherwise to the first column.
 - 85102 d. Otherwise, if the command is the `ex c` command, it shall be set to the indentation of
85103 the line replaced.

85104

autoprint, ap

85105

[Default *set*]

85106

If **autoprint** is set, the current line shall be written after each *ex* command that modifies the contents of the current edit buffer, and after each **tag** command for which the tag search pattern was found or tag line number was valid, unless:

85107

85108

85109

1. The command was executed while in open or visual mode.

85110

2. The command was executed as part of a **global** or **v** command or **@** buffer execution.

85111

3. The command was the form of the **read** command that reads a file into the edit buffer.

85112

4. The command was the **append**, **change**, or **insert** command.

85113

5. The command was not terminated by a <newline>.

85114

6. The current line shall be written by a flag specified to the command; for example, **delete #** shall write the current line as specified for the flag modifier to the **delete** command, and not as specified by the **autoprint** edit option.

85115

85116

85117

autowrite, aw

85118

[Default *unset*]

85119

If **autowrite** is set, and the edit buffer has been modified since it was last completely written to any file, the contents of the edit buffer shall be written as if the *ex* **write** command had been specified without arguments, before each command affected by the **autowrite** edit option is executed. Appending the character **'!'** to the command name of any of the *ex* commands except **'!'** shall prevent the write. If the write fails, it shall be an error and the command shall not be executed.

85120

85121

85122

85123

85124

85125

beautify, bf

85126

XSI

[Default *unset*]

85127

If **beautify** is set, all non-printable characters, other than <tab>s, <newline>s, and <form-feed>s, shall be discarded from text read in from files.

85128

85129

directory, dir

85130

[Default *implementation-defined*]

85131

The value of this option specifies the directory in which the editor buffer is to be placed. If this directory is not writable by the user, the editor shall quit.

85132

85133

edcompatible, ed

85134

[Default *unset*]

85135

Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled by repeating the suffixes.

85136

85137 **errorbells, eb**

85138 [Default *unset*]

85139 If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse
85140 video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.

85141 **exrc**

85142 [Default *unset*]

85143 If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in [Initialization in
85144 *ex* and *vi*](#) (on page 2576). If **exrc** is not set, *ex* shall ignore any **.exrc** file in the current directory
85145 during initialization, unless the current directory is that named by the *HOME* environment
85146 variable.

85147 **ignorecase, ic**

85148 [Default *unset*]

85149 If **ignorecase** is set, characters that have uppercase and lowercase representations shall have
85150 those representations considered as equivalent for purposes of regular expression comparison.

85151 The **ignorecase** edit option shall affect all remembered regular expressions; for example,
85152 unsetting the **ignorecase** edit option shall cause a subsequent *vi n* command to search for the
85153 last basic regular expression in a case-sensitive fashion.

85154 **list**

85155 [Default *unset*]

85156 If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for
85157 the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall
85158 be displayed as specified for the *ex print* command with the **l** flag specified. In open or visual
85159 text input mode, when the cursor does not rest on any character in the line, it shall rest on the
85160 ' \$ ' marking the end of the line.

85161 **magic**

85162 [Default *set*]

85163 If **magic** is set, modify the interpretation of characters in regular expressions and substitution
85164 replacement strings (see [Regular Expressions in *ex*](#) (on page 2608) and [Replacement Strings in
85165 *ex*](#), on page 2608).

85166 **mesg**

85167 [Default *set*]

85168 If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the
85169 terminal shall be turned on while in open or visual mode. The shell-level command *mesg n* shall
85170 take precedence over any setting of the *ex mesg* option; that is, if **mesg y** was issued before the
85171 editor started (or in a shell escape), such as:

85172 : !mesg y

85173 the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable
85174 incoming messages if **mesg n** was issued.

85175

number, nu

85176

[Default *unset*]

85177

If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input mode, each line shall be preceded by the line number it will have in the file.

85178

85179

85180

In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in the format specified by the *ex* **print** command with the **#** flag specified. This line number shall not be considered part of the line for the purposes of evaluating the current column; that is, column position 1 shall be the first column position after the format specified by the **print** command.

85181

85182

85183

85184

85185

paragraphs, para

85186

[Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

85187

The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual mode commands. The **paragraphs** edit option can be set to a character string consisting of zero or more character pairs. It shall be an error to set it to an odd number of characters.

85188

85189

85190

prompt

85191

[Default *set*]

85192

If **prompt** is set, *ex* command mode input shall be prompted for with a colon (':'); when unset, no prompt shall be written.

85193

85194

readonly

85195

[Default *see text*]

85196

If the **readonly** edit option is set, read-only mode shall be enabled (see [Write](#), on page 2603). The **readonly** edit option shall be initialized to set if either of the following conditions are true:

85197

85198

- The command-line option `-R` was specified.
- Performing actions equivalent to the `access()` function called with the following arguments indicates that the file lacks write permission:

85199

85200

1. The current pathname is used as the *path* argument.

85201

2. The constant `W_OK` is used as the *amode* argument.

85202

85203

The **readonly** edit option may be initialized to set for other, implementation-defined reasons. The **readonly** edit option shall not be initialized to unset based on any special privileges of the user or process. The **readonly** edit option shall be reinitialized each time that the contents of the edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again be reinitialized each time that the contents of the edit buffer are replaced.

85204

85205

85206

85207

85208

85209

redraw

85210

[Default *unset*]

85211

The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

85212

85213

remap

85214

[Default *set*]

85215

If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation shall continue until a final product is obtained. If unset, only a one-step translation shall be done.

85216

85216

85217

85218

report

85219

[Default 5]

85220

The value of this **report** edit option specifies what number of lines being added, copied, deleted, or modified in the edit buffer will cause an informational message to be written to the user. The following conditions shall cause an informational message. The message shall contain the number of lines added, copied, deleted, or modified, but is otherwise unspecified.

85221

85222

85223

85224

- An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.

85225

85226

85227

- An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.

85228

85229

85230

- An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or deletes a total of at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written. (For example, if 3 lines were added and 8 lines deleted during an *ex* **visual** command, 5 would be the number compared against the **report** edit option after the command completed.)

85231

85232

85233

85234

85235

85236

scroll, scr

85237

[Default (number of lines in the display -1)/2]

85238

The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex* <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the initial number of lines to scroll when no previous <control>-D or <control>-U command has been executed.

85239

85240

85241

85242

sections

85243

[Default in the POSIX locale *NHSHH HUnhsh*]

85244

The **sections** edit option shall define additional section boundaries for the open and visual mode commands. The **sections** edit option can be set to a character string consisting of zero or more character pairs; it shall be an error to set it to an odd number of characters.

85245

85246

85247

shell, sh

85248

[Default from the environment variable *SHELL*]

85249

The value of this option shall be a string. The default shall be taken from the *SHELL* environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility shall be the default.

85250

85251

85252

shiftwidth, sw

85253

[Default 8]

85254

The value of this option shall give the width in columns of an indentation level used during autoindentation and by the shift commands (< and >).

85255

85256

showmatch, sm

85257

[Default *unset*]

85258

The functionality described for the **showmatch** edit option need not be supported on block-mode terminals or terminals with insufficient capabilities.

85259

85260

If **showmatch** is set, in open or visual mode, when a ') ' or ' } ' is typed, if the matching ' (' or ' { ' is currently visible on the display, the matching ' (' or ' { ' shall be flagged moving the cursor to its location for an unspecified amount of time.

85261

85262

85263

showmode

85264

[Default *unset*]

85265

If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be displayed on the last line of the display. Command mode and text input mode shall be differentiated; other unspecified modes and implementation-defined information may be displayed.

85266

85267

85268

85269

slowopen

85270

[Default *unset*]

85271

If **slowopen** is set during open and visual text input modes, the editor shall not update portions of the display other than those display line columns that display the characters entered by the user (see [Input Mode Commands in vi](#), on page 3248).

85272

85273

85274

tabstop, ts

85275

[Default 8]

85276

The value of this edit option shall specify the column boundary used by a <tab> in the display (see [autoprint, ap](#) (on page 2610) and [Input Mode Commands in vi](#), on page 3248).

85277

85278

taglength, tl

85279

[Default zero]

85280

The value of this edit option shall specify the maximum number of characters that are considered significant in the user-specified tag name and in the tag name from the tags file. If the value is zero, all characters in both tag names shall be significant.

85281

85282

85283

tags

85284

[Default *see text*]

85285

The value of this edit option shall be a string of <blank>-delimited pathnames of files used by the **tag** command. The default value is unspecified.

85286

85287

term

85288

[Default from the environment variable *TERM*]

85289

85290

85291

85292

The value of this edit option shall be a string. The default shall be taken from the *TERM* variable in the environment. If the *TERM* environment variable is empty or null, the default is unspecified. The editor shall use the value of this edit option to determine the type of the display device.

85293

85294

The results are unspecified if the user changes the value of the term edit option after editor initialization.

85295

terse

85296

[Default *unset*]

85297

85298

85299

If **terse** is set, error messages may be less verbose. However, except for this caveat, error messages are unspecified. Furthermore, not all error messages need change for different settings of this option.

85300

warn

85301

[Default *set*]

85302

85303

85304

If **warn** is set, and the contents of the edit buffer have been modified since they were last completely written, the editor shall write a warning message before certain ! commands (see [Escape](#), on page 2606).

85305

window

85306

[Default *see text*]

85307

85308

A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in visual mode, to specify the number of lines displayed when the screen is repainted.

85309

85310

85311

If the **-w** command-line option is not specified, the default value shall be set to the value of the *LINES* environment variable. If the *LINES* environment variable is empty or null, the default shall be the number of lines in the display minus 1.

85312

85313

85314

Setting the **window** edit option to zero or to a value greater than the number of lines in the display minus 1 (either explicitly or based on the **-w** option or the *LINES* environment variable) shall cause the **window** edit option to be set to the number of lines in the display minus 1.

85315

The baud rate of the terminal line may change the default in an implementation-defined manner.

85316

wrapmargin, wm

85317

[Default 0]

85318

If the value of this edit option is zero, it shall have no effect.

85319

If not in the POSIX locale, the effect of this edit option is implementation-defined.

85320

Otherwise, it shall specify a number of columns from the ending margin of the terminal.

85321

85322

85323

During open and visual text input modes, for each character for which any part of the character is displayed in a column that is less than **wrapmargin** columns from the ending margin of the display line, the editor shall behave as follows:

85324

85325

85326

85327

1. If the character triggering this event is a <blank>, it, and all immediately preceding <blank>s on the current line entered during the execution of the current text input command, shall be discarded, and the editor shall behave as if the user had entered a single <newline> instead. In addition, if the next user-entered character is a <space>, it

85328 shall be discarded as well.

85329 2. Otherwise, if there are one or more <blank>s on the current line immediately preceding
85330 the last group of inserted non-<blank>s which was entered during the execution of the
85331 current text input command, the <blank>s shall be replaced as if the user had entered a
85332 single <newline> instead.

85333 If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any
85334 <blank>s at or after the cursor in the current line shall be discarded.

85335 The ending margin shall be determined by the system or overridden by the user, as described for
85336 *COLUMNS* in the ENVIRONMENT VARIABLES section and XBD [Chapter 8](#) (on page 159).

85337 **wrapsan, ws**

85338 [Default *set*]

85339 If **wrapsan** is set, searches (the *ex* / or ? addresses, or open and visual mode /, ?, N, and n
85340 commands) shall wrap around the beginning or end of the edit buffer; when unset, searches
85341 shall stop at the beginning or end of the edit buffer.

85342 **writeany, wa**

85343 [Default *unset*]

85344 If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be
85345 inhibited, as described in editor option **autowrite**.

85346 EXIT STATUS

85347 The following exit values shall be returned:

85348 0 Successful completion.

85349 >0 An error occurred.

85350 CONSEQUENCES OF ERRORS

85351 When any error is encountered and the standard input is not a terminal device file, *ex* shall not
85352 write the file or return to command or text input mode, and shall terminate with a non-zero exit
85353 status.

85354 Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP
85355 asynchronous event.

85356 Otherwise, when an error is encountered, the editor shall behave as specified in [Command Line
85357 Parsing in ex](#) (on page 2580).

85358 APPLICATION USAGE

85359 If a SIGSEGV signal is received while *ex* is saving a file, the file might not be successfully saved.

85360 The **next** command can accept more than one file, so usage such as:

85361 `next `ls [abc]*``

85362 is valid; it would not be valid for the **edit** or **read** commands, for example, because they expect
85363 only one file and unspecified results occur.

85364 EXAMPLES

85365 None.

85366 RATIONALE

85367 The *ex/vi* specification is based on the historical practice found in the 4 BSD and System V
85368 implementations of *ex* and *vi*.

85369 A *restricted editor* (both the historical *red* utility and modifications to *ex*) were considered and

85370 rejected for inclusion. Neither option provided the level of security that users might expect.

85371 It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to
 85372 implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor
 85373 addressing; thus, it is not a mandatory requirement that such features should work on all
 85374 terminals. It is the intention, however, that an *ex* implementation should provide the full set of
 85375 capabilities on all terminals capable of supporting them.

85376 Options

85377 The `-c` replacement for `+command` was inspired by the `-e` option of *sed*. Historically, all such
 85378 commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This
 85379 meant, for example, that `"/pattern"` would fail unless the **wrapsan** option was set.
 85380 POSIX.1-200x requires conformance to historical practice. The `+command` option is no longer
 85381 specified by POSIX.1-200x but may be present in some implementations. Historically, some
 85382 implementations restricted the *ex* commands that could be listed as part of the command line
 85383 arguments. For consistency, POSIX.1-200x does not permit these restrictions.

85384 In historical implementations of the editor, the `-R` option (and the **readonly** edit option) only
 85385 prevented overwriting of files; appending to files was still permitted, mapping loosely into the
 85386 *cs*h **noclobber** variable. Some implementations, however, have not followed this semantic, and
 85387 **readonly** does not permit appending either. POSIX.1-200x follows the latter practice, believing
 85388 that it is a more obvious and intuitive meaning of **readonly**.

85389 The `-s` option suppresses all interactive user feedback and is useful for editing scripts in batch
 85390 jobs. The list of specific effects is historical practice. The terminal type "incapable of supporting
 85391 open and visual modes" has historically been named "dumb".

85392 The `-t` option was required because the *ctags* utility appears in POSIX.1-200x and the option is
 85393 available in all historical implementations of *ex*.

85394 Historically, the *ex* and *vi* utilities accepted a `-x` option, which did encryption based on the
 85395 algorithm found in the historical *crypt* utility. The `-x` option for encryption, and the associated
 85396 *crypt* utility, were omitted because the algorithm used was not specifiable and the export control
 85397 laws of some nations make it difficult to export cryptographic technology. In addition, it did not
 85398 historically provide the level of security that users might expect.

85399 Standard Input

85400 An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file
 85401 character, `<control>-D`, is historically an *ex* command.

85402 There was no maximum line length in historical implementations of *ex*. Specifically, as it was
 85403 parsed in chunks, the addresses had a different maximum length than the filenames. Further, the
 85404 maximum line buffer size was declared as `BUFSIZ`, which was different lengths on different
 85405 systems. This version selected the value of `{LINE_MAX}` to impose a reasonable restriction on
 85406 portable usage of *ex* and to aid test suite writers in their development of realistic tests that
 85407 exercise this limit.

85408 Input Files

85409 It was an explicit decision by the standard developers that a `<newline>` be added to any file
 85410 lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make
 85411 text files lacking a trailing `<newline>` more portable. It is recognized that this will require a user-
 85412 specified option or extension for implementations that permit *ex* and *vi* to edit files of type other
 85413 than text if such files are not otherwise identified by the system. It was agreed that the ability to
 85414 edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an
 85415 *ex* or *vi* implementation be required to handle files other than text files.

85416 The paragraph in the INPUT FILES section, "By default, ...", is intended to close a long-
 85417 standing security problem in *ex* and *vi*; that of the "modeline" or "modelines" edit option. This
 85418 feature allows any line in the first or last five lines of the file containing the strings "ex:" or
 85419 "vi:" (and, apparently, "ei:" or "vx:") to be a line containing editor commands, and *ex*
 85420 interprets all the text up to the next ':' or <newline> as a command. Consider the
 85421 consequences, for example, of an unsuspecting user using *ex* or *vi* as the editor when replying to
 85422 a mail message in which a line such as:

```
85423 ex:! rm -rf :
```

85424 appeared in the signature lines. The standard developers believed strongly that an editor should
 85425 not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from
 85426 their implementations of *ex* and *vi*.

85427 Asynchronous Events

85428 The intention of the phrase "complete write" is that the entire edit buffer be written to stable
 85429 storage. The note regarding temporary files is intended for implementations that use temporary
 85430 files to back edit buffers unnamed by the user.

85431 Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual
 85432 mode; that is, it exited visual mode and entered *ex* mode. POSIX.1-200x permits, but does not
 85433 require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input
 85434 mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the
 85435 terminal on this event, and some did not. POSIX.1-200x requires that SIGINT behave identically
 85436 to <ESC>, and that the terminal not be alerted.

85437 Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as
 85438 completed lines were retained, but any partial line discarded, and the editor returned to
 85439 command mode. POSIX.1-200x is silent on this issue; implementations are encouraged to follow
 85440 historical practice, where possible.

85441 Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore
 85442 impossible to suspend the editor in visual text input mode. There are two major reasons for this.
 85443 The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where
 85444 the shell *execs* an application that then *execs vi* usually caused confusion for the terminal state if
 85445 SIGTSTP was delivered to the process group in the default manner. The second was that most
 85446 implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the
 85447 wrong time will cause them to crash. POSIX.1-200x is silent on this issue; implementations are
 85448 encouraged to treat suspension as an asynchronous event if possible.

85449 Historically, modifications to the edit buffer made before SIGINT interrupted an operation were
 85450 retained; that is, anywhere from zero to all of the lines to be modified might have been modified
 85451 by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT.
 85452 POSIX.1-200x permits this behavior, noting that the **undo** command is required to be able to
 85453 undo these partially completed commands.

85454 The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is
 85455 unspecified because some implementations attempt to save the edit buffer in a useful state when
 85456 other signals are received.

85457

Standard Error85458
85459
85460
85461

For *ex/vi*, diagnostic messages are those messages reported as a result of a failed attempt to invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

85462

Initialization in ex and vi85463
85464

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

85465
85466
85467
85468

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.

85469
85470

2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.

85471

3. Otherwise, the alternate pathname will be set to the filename argument.

85472
85473
85474
85475
85476
85477
85478
85479
85480

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. POSIX.1-200x requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

85481
85482

Historically, *ex* and *vi* read the **.exrc** file in the *\$HOME* directory twice, if the editor was executed in the *\$HOME* directory. POSIX.1-200x prohibits this behavior.

85483
85484
85485
85486
85487

Historically, the 4 BSD *ex* and *vi* read the *\$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc** file. POSIX.1-200x does not specify the **sourceany** option, and historical implementations are encouraged to delete it.

85488
85489
85490

The **.exrc** files must be owned by the real ID of the user, and not writable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the **.exrc** files in their home directories.

85491
85492
85493
85494
85495
85496
85497
85498

System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off, so by default, local **.exrc** files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that POSIX.1-200x can do. The implementation-defined exception is intended to permit groups to have local **.exrc** files that are shared by users, by creating pseudo-users to own the shared files.

85499
85500
85501
85502
85503
85504

POSIX.1-200x does not mention system-wide *ex* and *vi* start-up files. While they exist in several implementations of *ex* and *vi*, they are not present in any implementations considered historical practice by POSIX.1-200x. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if they are not writable by any user other than their owner. System-wide start-up files should be read before the *EXINIT* variable, *\$HOME/.exrc*, or local **.exrc** files are evaluated.

85505 Historically, any *ex* command could be entered in the *EXINIT* variable or the *.exrc* file, although
 85506 ones requiring that the edit buffer already contain lines of text generally caused historical
 85507 implementations of the editor to drop **core**. POSIX.1-200x requires that any *ex* command be
 85508 permitted in the *EXINIT* variable and *.exrc* files, for simplicity of specification and consistency,
 85509 although many of them will obviously fail under many circumstances.

85510 The initialization of the contents of the edit buffer uses the phrase “the effect shall be” with
 85511 regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded
 85512 during the initialization phase not be lost; that is, loading the edit buffer should fail if the *.exrc*
 85513 file read in the contents of a file and did not subsequently write the edit buffer. An additional
 85514 intent of this phrase is to specify that the initial current line and column is set as specified for the
 85515 individual *ex* commands.

85516 Historically, the *-t* option behaved as if the tag search were a *+command*; that is, it was executed
 85517 from the last line of the file specified by the tag. This resulted in the search failing if the pattern
 85518 was a forward search pattern and the **wraps**can edit option was not set. POSIX.1-200x does not
 85519 permit this behavior, requiring that the search for the tag pattern be performed on the entire file,
 85520 and, if not found, that the current line be set to a more reasonable location in the file.

85521 Historically, the empty edit buffer presented for editing when a file was not specified by the user
 85522 was unnamed. This is permitted by POSIX.1-200x; however, implementations are encouraged to
 85523 provide users a temporary filename for this buffer because it permits them the use of *ex*
 85524 commands that use the current pathname during temporary edit sessions.

85525 Historically, the file specified using the *-t* option was not part of the current argument list. This
 85526 practice is permitted by POSIX.1-200x; however, implementations are encouraged to include its
 85527 name in the current argument list for consistency.

85528 Historically, the *-c* command was generally not executed until a file that already exists was
 85529 edited. POSIX.1-200x requires conformance to this historical practice. Commands that could
 85530 cause the *-c* command to be executed include the *ex* commands **edit**, **next**, **recover**, **rewind**, and
 85531 **tag**, and the *vi* commands *<control>-^* and *<control>-]*. Historically, reading a file into an edit
 85532 buffer did not cause the *-c* command to be executed (even though it might set the current
 85533 pathname) with the exception that it did cause the *-c* command to be executed if: the editor was
 85534 in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read
 85535 commands had yet been attempted. For consistency and simplicity of specification,
 85536 POSIX.1-200x does not permit this behavior.

85537 Historically, the *-r* option was the same as a normal edit session if there was no recovery
 85538 information available for the file. This allowed users to enter:

```
85539 vi -r *.c
```

85540 and recover whatever files were recoverable. In some implementations, recovery was attempted
 85541 only on the first file named, and the file was not entered into the argument list; in others,
 85542 recovery was attempted for each file named. In addition, some historical implementations
 85543 ignored *-r* if *-t* was specified or did not support command line *file* arguments with the *-t* option.
 85544 For consistency and simplicity of specification, POSIX.1-200x disallows these special cases, and
 85545 requires that recovery be attempted the first time each file is edited.

85546 Historically, *vi* initialized the ‘ and ’ marks, but *ex* did not. This meant that if the first command
 85547 in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi +10 file*), *vi*
 85548 was entered without the marks being initialized. Because the standard developers believed the marks
 85549 to be generally useful, and for consistency and simplicity of specification, POSIX.1-200x requires
 85550 that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is
 85551 not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however,
 85552 it has always been possible to set (and use) marks in empty edit buffers in open and visual mode
 85553 edit sessions.

85554

Addressing

85555

Historically, *ex* and *vi* accepted the additional addressing forms '`\/'` and '`\?'`'. They were equivalent to '`/'`' and '`??'`', respectively. They are not required by POSIX.1-200x, mostly because nobody can remember whether they ever did anything different historically.

85556

85557

85558

Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the % address in empty files for others. For consistency, POSIX.1-200x requires support for the former in the few commands where it makes sense, and disallows it otherwise. In addition, because POSIX.1-200x requires that % be logically equivalent to "`1, $`", it is also supported where it makes sense and disallowed otherwise.

85559

85560

85561

85562

85563

Historically, the % address could not be followed by further addresses. For consistency and simplicity of specification, POSIX.1-200x requires that additional addresses be supported.

85564

85565

All of the following are valid *addresses*:

85566

`+++` Three lines after the current line.

85567

`/re/-` One line before the next occurrence of *re*.

85568

`-2` Two lines before the current line.

85569

`3 ---- 2` Line one (note intermediate negative address).

85570

`1 2 3` Line six.

85571

Any number of addresses can be provided to commands taking addresses; for example, "`1, 2, 3, 4, 5p`" prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next two lines. Note that the address **3;** must be evaluated before being discarded because the search origin for the **/foo/** command depends on this.

85572

85573

85574

85575

85576

85577

85578

Historically, values could be added to addresses by including them after one or more <blank>s; for example, **3 - 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as **/foo/+5**. However, only absolute values could be added; for example, **5 /foo/** was an error. POSIX.1-200x requires conformance to historical practice. Address offsets are separately specified from addresses because they could historically be provided to visual mode search commands.

85579

85580

85581

85582

85583

Historically, any missing addresses defaulted to the current line. This was true for leading and trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For consistency, POSIX.1-200x requires it for leading semicolon addresses as well.

85584

85585

85586

Historically, *ex* and *vi* accepted the '`^`' character as both an address and as a flag offset for commands. In both cases it was identical to the '`-`' character. POSIX.1-200x does not require or prohibit this behavior.

85587

85588

85589

Historically, the enhancements to basic regular expressions could be used in addressing; for example, '`~`', '`\<`', and '`\>`'. POSIX.1-200x requires conformance to historical practice; that is, that regular expression usage be consistent, and that regular expression enhancements be supported wherever regular expressions are used.

85590

85591

85592

85593

Command Line Parsing in ex

85594

85595

85596

85597

85598

85599

85600

Historical *ex* command parsing was even more complex than that described here. POSIX.1-200x requires the subset of the command parsing that the standard developers believed was documented and that users could reasonably be expected to use in a portable fashion, and that was historically consistent between implementations. (The discarded functionality is obscure, at best.) Historical implementations will require changes in order to comply with POSIX.1-200x; however, users are not expected to notice any of these changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

85601

85602

1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by `<newline>s` (they can contain vertical-line characters that are usually shell pipes).

85603

85604

2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands, optionally containing vertical-line characters, as their first arguments.

85605

85606

3. The **s** command takes a regular expression as its first argument, and uses the delimiting characters to delimit the command.

85607

85608

85609

85610

Historically, vertical-line characters in the *+command* argument of the **ex**, **edit**, **next**, **vi**, and **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did not delimit the command at all. For example, the following commands are all valid:

85611

85612

85613

85614

85615

```
:edit +25 | s/abc/ABC/ file.c
:s/ | /PIPE/
:read !spell % | columnate
:global/pattern/p | l
:s/a/b/ | s/c/d | set
```

85616

85617

85618

85619

Historically, empty or `<blank>` filled lines in **.exrc** files and **sourced** files (as well as *EXINIT* variables and *ex* command scripts) were treated as default commands; that is, **print** commands. POSIX.1-200x specifically requires that they be ignored when encountered in **.exrc** and **sourced** files to eliminate a common source of new user error.

85620

85621

85622

85623

85624

85625

85626

Historically, *ex* commands with multiple adjacent (or `<blank>`-separated) vertical lines were handled oddly when executed from *ex* mode. For example, the command `||| <carriage-return>`, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command `|` would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. POSIX.1-200x requires the *vi* behavior; that is, a single default command and line number increment for each command separator, and trailing `<newline>s` after vertical-line separators are discarded.

85627

85628

85629

Historically, *ex* permitted a single extra colon as a leading command character; for example, **:g/pattern/:p** was a valid command. POSIX.1-200x generalizes this to require that any number of leading colon characters be stripped.

85630

85631

85632

Historically, any prefix of the **delete** command could be followed without intervening `<blank>s` by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*. POSIX.1-200x requires conformance to historical practice.

85633

85634

Historically, the **k** command could be followed by the mark name without intervening `<blank>s`. POSIX.1-200x requires conformance to historical practice.

85635

85636

85637

85638

85639

85640

Historically, the **s** command could be immediately followed by flag and option characters; for example, **s/e/E/l s |sgc3p** was a valid command. However, flag characters could not stand alone; for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would succeed. (Obviously, the `'#'` flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; for example, the command **s/e/E/pg** would fail, while the

85641 command **s/e/E/gp** would succeed. POSIX.1-200x requires conformance to historical practice.

85642 Historically, the first command name that had a prefix matching the input from the user was the
85643 executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command.
85644 Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**.
85645 POSIX.1-200x requires conformance to historical practice. The restriction on command search
85646 order for implementations with extensions is to avoid the addition of commands such that the
85647 historical prefixes would fail to work portably.

85648 Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands,
85649 separated by vertical-line characters, that entered or exited visual mode or the editor. Because
85650 implementations of *vi* exist that do not exhibit this failure mode, POSIX.1-200x does not permit
85651 it.

85652 The requirement that alphabetic command names consist of all following alphabetic characters
85653 up to the next non-alphabetic character means that alphabetic command names must be
85654 separated from their arguments by one or more non-alphabetic characters, normally a <blank>
85655 or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

85656 Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*,
85657 <newline>, <carriage-return>) erased any prompting character and displayed the next lines
85658 without scrolling the terminal; that is, immediately below any previously displayed lines. This
85659 provided a cleaner presentation of the lines in the file for the user. POSIX.1-200x does not require
85660 this behavior because it may be impossible in some situations; however, implementations are
85661 strongly encouraged to provide this semantic if possible.

85662 Historically, it was possible to change files in the middle of a command, and have the rest of the
85663 command executed in the new file; for example:

```
85664 :edit +25 file.c | s/abc/ABC/ | 1
```

85665 was a valid command, and the substitution was attempted in the newly edited file.
85666 POSIX.1-200x requires conformance to historical practice. The following commands are
85667 examples that exercise the *ex* parser:

```
85668 echo 'foo | bar' > file1; echo 'foo/bar' > file2;
```

```
85669 vi
```

```
85670 :edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\/SLASH/ | wq
```

85671 Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or *****
85672 commands changing edit buffers during execution of their associated commands. Because this
85673 would almost invariably result in catastrophic failure of the editor, and implementations exist
85674 that do exhibit these problems, POSIX.1-200x requires that changing the edit buffer during a
85675 **global** or **v** command, or during a **@** or ***** command for which there will be more than a single
85676 execution, be an error. Implementations supporting multiple edit buffers simultaneously are
85677 strongly encouraged to apply the same semantics to switching between buffers as well.

85678 The *ex* command quoting required by POSIX.1-200x is a superset of the quoting in historical
85679 implementations of the editor. For example, it was not historically possible to escape a <blank>
85680 in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had been
85681 entered for the edit command, and there was no method of escaping a <blank> in the first
85682 argument of an **edit**, **ex**, **next**, or **visual** command at all. POSIX.1-200x extends historical
85683 practice, requiring that quoting behavior be made consistent across all *ex* commands, except for
85684 the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V
85685 instead of backslashes for quoting. For those four commands, POSIX.1-200x requires
85686 conformance to historical practice.

85687 Backslash quoting in *ex* is non-intuitive. Backslash escapes are ignored unless they escape a
85688 special character; for example, when performing *file* argument expansion, the string "\\%" is
85689 equivalent to '\%', not "\<current pathname>". This can be confusing for users because

85690 backslash is usually one of the characters that causes shell expansion to be performed, and
 85691 therefore shell quoting rules must be taken into consideration. Generally, quoting characters are
 85692 only considered if they escape a special character, and a quoting character must be provided for
 85693 each layer of parsing for which the character is special. As another example, only a single
 85694 backslash is necessary for the '\1' sequence in substitute replacement patterns, because the
 85695 character '1' is not special to any parsing layer above it.

85696 <control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands
 85697 where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character
 85698 may be escaped by a <control>-V whether it would have a special meaning or not. POSIX.1-200x
 85699 requires conformance to historical practice.

85700 Historical implementations of the editor did not require delimiters within character classes to be
 85701 escaped; for example, the command **s/[/]/** on the string "xxx/yyy" would delete the '/' from
 85702 the string. POSIX.1-200x disallows this historical practice for consistency and because it places a
 85703 large burden on implementations by requiring that knowledge of regular expressions be built
 85704 into the editor parser.

85705 Historically, quoting <newline>s in *ex* commands was handled inconsistently. In most cases, the
 85706 <newline> always terminated the command, regardless of any preceding escape character,
 85707 because backslash characters did not escape <newline>s for most *ex* commands. However, some
 85708 *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted <newline>s to be escaped
 85709 (although in the case of **map** and **abbreviation**, <control>-V characters escaped them instead of
 85710 backslashes). This was true in not only the command line, but also **.exrc** and **sourced** files. For
 85711 example, the command:

```
85712 map = foo<control-V><newline>bar
```

85713 would succeed, although it was sometimes difficult to get the <control>-V and the inserted
 85714 <newline> passed to the *ex* parser. For consistency and simplicity of specification, POSIX.1-200x
 85715 requires that it be possible to escape <newline>s in *ex* commands at all times, using backslashes
 85716 for most *ex* commands, and using <control>-V characters for the **map** and **abbreviation**
 85717 commands. For example, the command **print<newline>list** is required to be parsed as the single
 85718 command **print<newline>list**. While this differs from historical practice, POSIX.1-200x
 85719 developers believed it unlikely that any script or user depended on the historical behavior.

85720 Historically, an error in a command specified using the **-c** option did not cause the rest of the **-c**
 85721 commands to be discarded. POSIX.1-200x disallows this for consistency with mapped keys, the
 85722 **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

85723 Input Editing in *ex*

85724 One of the common uses of the historical *ex* editor is over slow network connections. Editors that
 85725 run in canonical mode can require far less traffic to and from, and far less processing on, the host
 85726 machine, as well as more easily supporting block-mode terminals. For these reasons,
 85727 POSIX.1-200x requires that *ex* be implemented using canonical mode input processing, as was
 85728 done historically.

85729 POSIX.1-200x does not require the historical 4 BSD input editing characters "word erase" or
 85730 "literal next". For this reason, it is unspecified how they are handled by *ex*, although they must
 85731 have the required effect. Implementations that resolve them after the line has been ended using a
 85732 <newline> or <control>-M character, and implementations that rely on the underlying system
 85733 terminal support for this processing, are both conforming. Implementations are strongly urged
 85734 to use the underlying system functionality, if at all possible, for compatibility with other system
 85735 text input interfaces.

85736 Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor
 85737 moved to display the new end of the **autoindent** characters, but did not move the cursor to a
 85738 new line, nor did it erase the <control>-D character from the line. POSIX.1-200x does not specify

85739 that the cursor remain on the same line or that the rest of the line is erased; however,
 85740 implementations are strongly encouraged to provide the best possible user interface; that is, the
 85741 cursor should remain on the same line, and any <control>-D character on the line should be
 85742 erased.

85743 POSIX.1-200x does not require the historical 4 BSD input editing character “reprint”,
 85744 traditionally <control>-R, which redisplayed the current input from the user. For this reason,
 85745 and because the functionality cannot be implemented after the line has been terminated by the
 85746 user, POSIX.1-200x makes no requirements about this functionality. Implementations are
 85747 strongly urged to make this historical functionality available, if possible.

85748 Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*.
 85749 POSIX.1-200x requires conformance to historical practice to avoid breaking historical *ex* scripts
 85750 and *.exrc* files.

85751 **eof**

85752 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left
 85753 unspecified so that implementations can conform in the presence of systems that do not support
 85754 this functionality. Implementations are encouraged to modify the line and redisplay it
 85755 immediately, if possible.

85756 The specification of the handling of the *eof* character differs from historical practice only in that
 85757 *eof* characters are not discarded if they follow normal characters in the text input. Historically,
 85758 they were always discarded.

85759 **Command Descriptions in ex**

85760 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and
 85761 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit
 85762 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or
 85763 command execution in an empty file, make sense only for commands that add new text to the
 85764 edit buffer or write commands (because users may wish to write empty files). POSIX.1-200x
 85765 requires this behavior for such commands and disallows it otherwise, for consistency and
 85766 simplicity of specification.

85767 A count to an *ex* command has been historically corrected to be no greater than the last line in a
 85768 file; for example, in a five-line file, the command **1,6print** would fail, but the command
 85769 **1print300** would succeed. POSIX.1-200x requires conformance to historical practice.

85770 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as
 85771 described by POSIX.1-200x, but there were some special cases. For instance, the **list**, **number**,
 85772 and **print** commands ignored trailing address offsets; for example, **3p +++#** would display line
 85773 3, and 3 would be the current line after the execution of the command. The **open** and **visual**
 85774 commands ignored both the trailing offsets and the trailing flags. Also, flags specified to the
 85775 **open** and **visual** commands interacted badly with the **list** edit option, and setting and then
 85776 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the
 85777 specified format. For consistency and simplicity of specification, POSIX.1-200x does not permit
 85778 any of these exceptions to the general rule.

85779 POSIX.1-200x uses the word *copy* in several places when discussing buffers. This is not intended
 85780 to imply implementation.

85781 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would
 85782 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a
 85783 *count*. POSIX.1-200x requires conformance to historical practice by default, but does not
 85784 preclude extensions.

85785 Historically, the contents of the unnamed buffer were frequently discarded after commands that

85786 did not explicitly affect it; for example, when using the **edit** command to switch files. For
85787 consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

85788 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting
85789 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user
85790 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric
85791 buffers would not have changed. POSIX.1-200x requires conformance to historical practice.
85792 Numeric buffers are described in the *ex* utility in order to confine the description of buffers to a
85793 single location in POSIX.1-200x.

85794 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as
85795 does the method for doing shell expansion. Implementations wishing to provide users with the
85796 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit
85797 option.

85798 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to
85799 do so; for example, **!date > /dev/null** does not require a screen refresh because the output of
85800 the UNIX *date* command requires only a single line of the screen. POSIX.1-200x requires that the
85801 screen be refreshed if it has been overwritten, but makes no requirements as to how an
85802 implementation should make that determination. Implementations may prompt and refresh the
85803 screen regardless.

85804 Abbreviate

85805 Historical practice was that characters that were entered as part of an abbreviation replacement
85806 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,
85807 and so on; that is, they were logically pushed onto the terminal input queue, and were not a
85808 simple replacement. POSIX.1-200x requires conformance to historical practice. Historical
85809 practice was that whenever a non-word character (that had not been escaped by a <control>-V)
85810 was entered after a word character, *vi* would check for abbreviations. The check was based on
85811 the type of the character entered before the word character of the word/non-word pair that
85812 triggered the check. The word character of the word/non-word pair that triggered the check and
85813 all characters entered before the trigger pair that were of that type were included in the check,
85814 with the exception of <blank>s, which always delimited the abbreviation.

85815 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can
85816 be no transitions from word to non-word characters (or *vice versa*) other than between the last
85817 and next-to-last characters in the *lhs*, and there can be no <blank>s in the *lhs*. In addition,
85818 because of the historical quoting rules, it was impossible to enter a literal <control>-V in the *lhs*.
85819 POSIX.1-200x requires conformance to historical practice. Historical implementations did not
85820 inform users when abbreviations that could never be used were entered; implementations are
85821 strongly encouraged to do so.

85822 For example, the following abbreviations will work:

```
85823 :ab (p REPLACE
85824 :ab p REPLACE
85825 :ab ((p REPLACE
```

85826 The following abbreviations will not work:

```
85827 :ab ( REPLACE
85828 :ab (pp REPLACE
```

85829 Historical practice is that words on the *vi* colon command line were subject to abbreviation
85830 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**
85831 command. Because there are implementations that do not do abbreviation expansion for the first
85832 argument to those commands, this is permitted, but not required, by POSIX.1-200x. However,
85833 the following sequence:

85834 :ab foo bar

85835 :ab foo baz

85836 resulted in the addition of an abbreviation of "baz" for the string "bar" in historical *ex/vi*, and
85837 the sequence:

85838 :ab foo1 bar

85839 :ab foo2 bar

85840 :unabbreviate foo2

85841 deleted the abbreviation "foo1", not "foo2". These behaviors are not permitted by
85842 POSIX.1-200x because they clearly violate the expectations of the user.

85843 It was historical practice that <control>-V, not backslash, characters be interpreted as escaping
85844 subsequent characters in the **abbreviate** command. POSIX.1-200x requires conformance to
85845 historical practice; however, it should be noted that an abbreviation containing a <blank> will
85846 never work.

85847 **Append**

85848 Historically, any text following a vertical-line command separator after an **append**, **change**, or
85849 **insert** command became part of the insert text. For example, in the command:

85850 :g/pattern/append|stuff1

85851 a line containing the text "stuff1" would be appended to each line matching pattern. It was
85852 also historically valid to enter:

85853 :append|stuff1

85854 stuff2

85855 .

85856 and the text on the *ex* command line would be appended along with the text inserted after it.
85857 There was an historical bug, however, that the user had to enter two terminating lines (the ' . '
85858 lines) to terminate text input mode in this case. POSIX.1-200x requires conformance to historical
85859 practice, but disallows the historical need for multiple terminating lines.

85860 **Change**

85861 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
85862 the change command when no text is input, is as described in POSIX.1-200x. However, one
85863 System V implementation is known to have been modified such that the cursor is positioned on
85864 the first address specified, and not on the line before the first address. POSIX.1-200x disallows
85865 this modification for consistency.

85866 Historically, the **change** command did not support buffer arguments, although some
85867 implementations allow the specification of an optional buffer. This behavior is neither required
85868 nor disallowed by POSIX.1-200x.

85869 **Change Directory**

85870 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
85871 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have
85872 ' . ' or " . ." as their first component. Elements in the **cdpath** edit option are colon-separated.
85873 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
85874 variable. This feature was not included in POSIX.1-200x because it does not exist in any of the
85875 implementations considered historical practice.

85876

Copy

85877

85878

85879

Historical implementations of *ex* permitted copies to lines inside of the specified range; for example, **:2,5copy3** was a valid command. POSIX.1-200x requires conformance to historical practice.

85880

Delete

85881

85882

POSIX.1-200x requires support for the historical parsing of a **delete** command followed by flags, without any intervening <blank>s. For example:

85883

1dp Deletes the first line and prints the line that was second.

85884

1delep As for **1dp**.

85885

1d Deletes the first line, saving it in buffer *p*.

85886

85887

1d p1l (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was second.

85888

Edit

85889

85890

85891

85892

Historically, any *ex* command could be entered as a *+command* argument to the **edit** command, although some (for example, **insert** and **append**) were known to confuse historical implementations. For consistency and simplicity of specification, POSIX.1-200x requires that any command be supported as an argument to the **edit** command.

85893

85894

85895

Historically, the command argument was executed with the current line set to the last line of the file, regardless of whether the **edit** command was executed from visual mode or not. POSIX.1-200x requires conformance to historical practice.

85896

85897

85898

Historically, the *+command* specified to the **edit** and **next** commands was delimited by the first <blank>, and there was no way to quote them. For consistency, POSIX.1-200x requires that the usual *ex* backslash quoting be provided.

85899

85900

85901

Historically, specifying the *+command* argument to the edit command required a filename to be specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of specification, POSIX.1-200x does not permit this usage to fail for that reason.

85902

85903

85904

Historically, only the cursor position of the last file edited was remembered by the editor. POSIX.1-200x requires that this be supported; however, implementations are permitted to remember and restore the cursor position for any file previously edited.

85905

File

85906

85907

85908

85909

85910

Historical versions of the *ex* editor **file** command displayed a current line and number of lines in the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a current line and number of lines in the edit buffer of 1 in the same situation. POSIX.1-200x does not permit this discrepancy, instead requiring that a message be displayed indicating that the file is empty.

85911

Global

85912

85913

The two-pass operation of the **global** and **v** commands is not intended to imply implementation, only the required result of the operation.

85914

85915

85916

The current line and column are set as specified for the individual *ex* commands. This requirement is cumulative; that is, the current line and column must track across all the commands executed by the **global** or **v** commands.

85917

Insert

85918

See the RATIONALE for the **append** command.

85919

85920

85921

Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer was empty. POSIX.1-200x requires that this command behave consistently with the **append** command.

85922

Join

85923

85924

85925

The action of the **join** command in relation to the special characters is only defined for the POSIX locale because the correct amount of white space after a period varies; in Japanese none is required, in French only a single space, and so on.

85926

List

85927

85928

85929

The historical output of the **list** command was potentially ambiguous. The standard developers believed correcting this to be more important than adhering to historical practice, and POSIX.1-200x requires unambiguous output.

85930

Map

85931

85932

85933

85934

85935

85936

85937

85938

85939

Historically, command mode maps only applied to command names; for example, if the character 'x' was mapped to 'y', the command **fx** searched for the 'x' character, not the 'y' character. POSIX.1-200x requires this behavior. Historically, entering <control>-V as the first character of a *vi* command was an error. Several implementations have extended the semantics of *vi* such that <control>-V means that the subsequent command character is not mapped. This is permitted, but not required, by POSIX.1-200x. Regardless, using <control>-V to escape the second or later character in a sequence of characters that might match a **map** command, or any character in text input mode, is historical practice, and stops the entered keys from matching a map. POSIX.1-200x requires conformance to historical practice.

85940

85941

Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored the map. POSIX.1-200x requires that the mapped digits not be ignored.

85942

85943

85944

The historical implementation of the **map** command did not permit **map** commands that were more than a single character in length if the first character was printable. This behavior is permitted, but not required, by POSIX.1-200x.

85945

85946

Historically, mapped characters were remapped unless the **remap** edit option was not set, or the prefix of the mapped characters matched the mapping characters; for example, in the **map**:

85947

```
:map ab abcd
```

85948

85949

85950

the characters "ab" were used as is and were not remapped, but the characters "cd" were mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms. POSIX.1-200x requires conformance to historical practice, and that such loops be interruptible.

85951

85952

85953

85954

Text input maps had the same problems with expanding the *lhs* for the *ex* **map!** and **unmap!** command as did the *ex* **abbreviate** and **unabbreviate** commands. See the RATIONALE for the *ex* **abbreviate** command. POSIX.1-200x requires similar modification of some historical practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate** commands.

85955

85956

Historically, **maps** that were subsets of other **maps** behaved differently depending on the order in which they were defined. For example:

85957

85958

```
:map! ab      short
```

```
:map! abc     long
```

85959

85960

would always translate the characters "ab" to "short", regardless of how fast the characters "abc" were entered. If the entry order was reversed:

```
85961      :map! abc      long
85962      :map! ab       short
```

85963 the characters "ab" would cause the editor to pause, waiting for the completing 'c' character,
85964 and the characters might never be mapped to "short". For consistency and simplicity of
85965 specification, POSIX.1-200x requires that the shortest match be used at all times.

85966 The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified
85967 because the timing capabilities of systems are often inexact and variable, and it may depend on
85968 other factors such as the speed of the connection. The time should be long enough for the user to
85969 be able to complete the sequence, but not long enough for the user to have to wait. Some
85970 implementations of *vi* have added a **keytime** option, which permits users to set the number of
85971 0,1 seconds the editor waits for the completing characters. Because mapped terminal function
85972 and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input
85973 mode, **maps** starting with <ESC> characters are generally exempted from this timeout period,
85974 or, at least timed out differently.

85975 **Mark**

85976 Historically, users were able to set the "previous context" marks explicitly. In addition, the *ex*
85977 commands " and " and the *vi* commands ", ", ", and " all referred to the same mark. In addition,
85978 the previous context marks were not set if the command, with which the address setting the
85979 mark was associated, failed. POSIX.1-200x requires conformance to historical practice.
85980 Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the
85981 change was undone. POSIX.1-200x requires conformance to historical practice.

85982 The description of the special events that set the ' and ' marks matches historical practice. For
85983 example, historically the command */a,/b/* did not set the ' and ' marks, but the command
85984 */a,/b/delete* did.

85985 **Next**

85986 Historically, any *ex* command could be entered as a *+command* argument to the **next** command,
85987 although some (for example, **insert** and **append**) were known to confuse historical
85988 implementations. POSIX.1-200x requires that any command be permitted and that it behave as
85989 specified. The **next** command can accept more than one file, so usage such as:

```
85990 next `ls [abc] `
```

85991 is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect
85992 only one filename.

85993 Historically, the **next** command behaved differently from the **:rewind** command in that it
85994 ignored the force flag if the **autowrite** flag was set. For consistency, POSIX.1-200x does not
85995 permit this behavior.

85996 Historically, the **next** command positioned the cursor as if the file had never been edited before,
85997 regardless. POSIX.1-200x does not permit this behavior, for consistency with the **edit** command.

85998 Implementations wanting to provide a counterpart to the **next** command that edited the
85999 previous file have used the command **prev[ious]**, which takes no *file* argument. POSIX.1-200x
86000 does not require this command.

86001

Open

86002

86003

86004

86005

Historically, the **open** command would fail if the **open** edit option was not set. POSIX.1-200x does not mention the **open** edit option and does not require this behavior. Some historical implementations do not permit entering open mode from open or visual mode, only from *ex* mode. For consistency, POSIX.1-200x does not permit this behavior.

86006

86007

86008

Historically, entering open mode from the command line (that is, *vi* **+open**) resulted in anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command <control>-G did not work. For consistency, POSIX.1-200x does not permit this behavior.

86009

86010

86011

Historically, the **open** command only permitted ' / ' characters to be used as the search pattern delimiter. For consistency, POSIX.1-200x requires that the search delimiters used by the **s**, **global**, and **v** commands be accepted as well.

86012

Preserve

86013

86014

86015

The **preserve** command does not historically cause the file to be considered unmodified for the purposes of future commands that may exit the editor. POSIX.1-200x requires conformance to historical practice.

86016

86017

86018

Historical documentation stated that mail was not sent to the user when preserve was executed; however, historical implementations did send mail in this case. POSIX.1-200x requires conformance to the historical implementations.

86019

Print

86020

86021

86022

The writing of NUL by the **print** command is not specified as a special case because the standard developers did not want to require *ex* to support NUL characters. Historically, characters were displayed using the ARPA standard mappings, which are as follows:

86023

1. Printable characters are left alone.

86024

86025

2. Control characters less than \177 are represented as '^' followed by the character offset from the '@' character in the ASCII map; for example, \007 is represented as '^G'.

86026

3. \177 is represented as '^?' followed by '? '.

86027

86028

86029

86030

86031

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their eighth bit set as the two characters "M-" followed by the seven-bit display as described above.) The latter probably has the best claim to historical practice because it was used for the **-v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

86032

No specific display format is required by POSIX.1-200x.

86033

86034

86035

86036

86037

Explicit dependence on the ASCII character set has been avoided where possible, hence the use of the phrase an "implementation-defined multi-character sequence" for the display of non-printable characters in preference to the historical usage of, for instance, "^I" for the <tab>. Implementations are encouraged to conform to historical practice in the absence of any strong reason to diverge.

86038

86039

86040

86041

Historically, all *ex* commands beginning with the letter 'p' could be entered using capitalized versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command names. POSIX.1-200x permits, but does not require, this historical practice because capital forms of the commands are used by some implementations for other purposes.

86042

Put86043
86044
86045
86046
86047
86048
86049
86050
86051
86052

Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open or visual mode **P** command, if the buffer was named and was cut in character mode, and the same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer was the source of the text, the entire line from which the text was taken was usually **put**, and the buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior. In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in errors as well, such as appending text that was unrelated to the (supposed) contents of the buffer. For consistency and simplicity of specification, POSIX.1-200x does not permit these behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of the buffers are not altered by changing the mode of the editor.

86053

Read86054
86055
86056
86057
86058

Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior. Historically, a **read** in open or visual mode from a program left the cursor at the last line read in, not the first. For consistency, POSIX.1-200x does not permit this behavior.

86059
86060

Historical implementations of *ex* were unable to undo **read** commands that read from the output of a program. For consistency, POSIX.1-200x does not permit this behavior.

86061
86062
86063
86064
86065

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified “characters”, not “bytes”. POSIX.1-200x requires that the number of bytes be displayed, not the number of characters, because it may be difficult in multi-byte implementations to determine the number of characters read. Implementations are encouraged to clarify the message displayed to the user.

86066
86067
86068
86069

Historically, reads were not permitted on files other than type regular, except that FIFO files could be read (probably only because they did not exist when *ex* and *vi* were originally written). Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way to force the read. POSIX.1-200x permits, but does not require, this behavior.

86070

Recover86071
86072
86073
86074

Some historical implementations of the editor permitted users to recover the edit buffer contents from a previous edit session, and then exit without saving those contents (or explicitly discarding them). The intent of POSIX.1-200x in requiring that the edit buffer be treated as already modified is to prevent this user error.

86075

Rewind86076
86077
86078

Historical implementations supported the **rewind** command when the user was editing the first file in the list; that is, the file that the **rewind** command would edit. POSIX.1-200x requires conformance to historical practice.

86079

Substitute86080
86081
86082
86083
86084

Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the last regular expression used in any command as the pattern, the same as the **~** command. The **r** option is not required by POSIX.1-200x. Historically, the **c** and **g** options were toggled; for example, the command **:s/abc/def/** was the same as **s/abc/def/ccccggg**. For simplicity of specification, POSIX.1-200x does not permit this behavior.

86085
86086

The tilde command is often used to replace the last search RE. For example, in the sequence:

```
s/red/blue/
```


86087 /green
86088 ~

86089 the ~ command is equivalent to:

86090 s/green/blue/

86091 Historically, *ex* accepted all of the following forms:

86092 s/abc/def/

86093 s/abc/def

86094 s/abc/

86095 s/abc

86096 POSIX.1-200x requires conformance to this historical practice.

86097 The *s* command presumes that the '^' character only occupies a single column in the display.
86098 Much of the *ex* and *vi* specification presumes that the <space> only occupies a single column in
86099 the display. There are no known character sets for which this is not true.

86100 Historically, the final column position for the substitute commands was based on previous
86101 column movements; a search for a pattern followed by a substitution would leave the column
86102 position unchanged, while a *0* command followed by a substitution would change the column
86103 position to the first non-<blank>. For consistency and simplicity of specification, POSIX.1-200x
86104 requires that the final column position always be set to the first non-<blank>.

86105 Set

86106 Historical implementations redisplayed all of the options for each occurrence of the **all** keyword.
86107 POSIX.1-200x permits, but does not require, this behavior.

86108 Tag

86109 No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry.
86110 Historical practice has been to look for the path found in the **tags** file, based on the current
86111 directory. A useful extension found in some implementations is to look based on the directory
86112 containing the tags file that held the entry, as well. No requirement is made as to which reference
86113 for the tag in the tags file is used. This is deliberate, in order to permit extensions such as
86114 multiple entries in a tags file for a tag.

86115 Because users often specify many different tags files, some of which need not be relevant or exist
86116 at any particular time, POSIX.1-200x requires that error messages about problem tags files be
86117 displayed only if the requested tag is not found, and then, only once for each time that the **tag**
86118 edit option is changed.

86119 The requirement that the current edit buffer be unmodified is only necessary if the file indicated
86120 by the tag entry is not the same as the current file (as defined by the current pathname).
86121 Historically, the file would be reloaded if the filename had changed, as well as if the filename
86122 was different from the current pathname. For consistency and simplicity of specification,
86123 POSIX.1-200x does not permit this behavior, requiring that the name be the only factor in the
86124 decision.

86125 Historically, *vi* only searched for tags in the current file from the current cursor to the end of the
86126 file, and therefore, if the **wrapsan** option was not set, tags occurring before the current cursor
86127 were not found. POSIX.1-200x considers this a bug, and implementations are required to search
86128 for the first occurrence in the file, regardless.

86129

Undo

86130

The **undo** description deliberately uses the word “modified”. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

86131

86132

86133

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. POSIX.1-200x requires a simplified behavior for consistency and simplicity of specification.

86134

86135

86136

86137

86138

Version

86139

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

86140

86141

86142

Write

86143

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified “characters”, not “bytes”. POSIX.1-200x requires that the number of bytes be displayed, not the number of characters because it may be difficult in multi-byte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

86144

86145

86146

86147

86148

Implementation-defined tests are permitted so that implementations can make additional checks; for example, for locks or file modification times.

86149

86150

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in POSIX.1-200x to permit implementations to let the **write** succeed, so that the append semantics are similar to those of the historical *csht*.

86151

86152

86153

Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around dealing with “empty” files was to always have a line in the edit buffer, no matter what, it wrote them as files of a single, empty line. POSIX.1-200x does not permit this behavior.

86154

86155

86156

Historically, *ex* restored standard output and standard error to their values as of when *ex* was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals. POSIX.1-200x does not permit this, requiring that the program output be captured and displayed as if by the *ex* **print** command.

86157

86158

86159

86160

86161

Adjust Window

86162

Historically, the line count was set to the value of the **scroll** option if the type character was end-of-file. This feature was broken on most historical implementations long ago, however, and is not documented anywhere. For this reason, POSIX.1-200x is resolutely silent.

86163

86164

86165

Historically, the **z** command was <blank>-sensitive and **z +** and **z -** did different things than **z+** and **z-** because the type could not be distinguished from a flag. (The commands **z .** and **z =** were historically invalid.) POSIX.1-200x requires conformance to this historical practice.

86166

86167

86168

Historically, the **z** command was further <blank>-sensitive in that the *count* could not be <blank>-delimited; for example, the commands **z= 5** and **z- 5** were also invalid. Because the *count* is not ambiguous with respect to either the type character or the flags, this is not permitted by POSIX.1-200x.

86169

86170

86171

86172

Escape86173
86174
86175
86176

Historically, *ex* filter commands only read the standard output of the commands, letting standard error appear on the terminal as usual. The *vi* utility, however, read both standard output and standard error. POSIX.1-200x requires the latter behavior for both *ex* and *vi*, for consistency.

86177

Shift Left and Shift Right86178
86179
86180

Historically, it was possible to add shift characters to increase the effect of the command; for example, <<< outdented (or >>> indented) the lines 3 levels of indentation instead of the default 1. POSIX.1-200x requires conformance to historical practice.

86181

<control>-D86182
86183
86184
86185

Historically, the <control>-D command erased the prompt, providing the user with an unbroken presentation of lines from the edit buffer. This is not required by POSIX.1-200x; implementations are encouraged to provide it if possible. Historically, the <control>-D command took, and then ignored, a *count*. POSIX.1-200x does not permit this behavior.

86186

Write Line Number86187
86188
86189

Historically, the *ex* = command, when executed in *ex* mode in an empty edit buffer, reported 0, and from open or visual mode, reported 1. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

86190

Execute86191
86192
86193

Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**, **insert**, and **change**) in executed buffers. POSIX.1-200x does not permit this exclusion for consistency.

86194
86195
86196

Historically, the logical contents of the buffer being executed did not change if the buffer itself were modified by the commands being executed; that is, buffer execution did not support self-modifying code. POSIX.1-200x requires conformance to historical practice.

86197
86198
86199

Historically, the @ command took a range of lines, and the @ buffer was executed once per line, with the current line (' . ') set to each specified line. POSIX.1-200x requires conformance to historical practice.

86200
86201
86202
86203
86204

Some historical implementations did not notice if errors occurred during buffer execution. This, coupled with the ability to specify a range of lines for the *ex* @ command, makes it trivial to cause them to drop **core**. POSIX.1-200x requires that implementations stop buffer execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are replaced (for example, the buffer executes the *ex* :**edit** command).

86205

Regular Expressions in ex86206
86207
86208
86209
86210

Historical practice is that the characters in the replacement part of the last **s** command—that is, those matched by entering a '~' in the regular expression—were not further expanded by the regular expression engine. So, if the characters contained the string "a.," they would match 'a' followed by ".," and not 'a' followed by any character. POSIX.1-200x requires conformance to historical practice.

86211
86212
86213
86214
86215
86216
86217
86218
86219
86220
86221
86222
86223
86224
86225
86226
86227
86228
86229
86230
86231
86232
86233
86234
86235
86236
86237
86238
86239
86240
86241
86242
86243
86244
86245
86246
86247
86248
86249
86250
86251
86252
86253
86254
86255
86256
86257
86258

Edit Options in ex

The following paragraphs describe the historical behavior of some edit options that were not, for whatever reason, included in POSIX.1-200x. Implementations are strongly encouraged to only use these names if the functionality described here is fully supported.

- extended** The **extended** edit option has been used in some implementations of *vi* to provide extended regular expressions instead of basic regular expressions. This option was omitted from POSIX.1-200x because it is not widespread historical practice.
- flash** The **flash** edit option historically caused the screen to flash instead of beeping on error. This option was omitted from POSIX.1-200x because it is not found in some historical implementations.
- hardtabs** The **hardtabs** edit option historically defined the number of columns between hardware tab settings. This option was omitted from POSIX.1-200x because it was believed to no longer be generally useful.
- modeline** The **modeline** (sometimes named **modelines**) edit option historically caused *ex* or *vi* to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.
- open** The **open** edit option historically disallowed the *ex* **open** and **visual** commands. This edit option was omitted because these commands are required by POSIX.1-200x.
- optimize** The **optimize** edit option historically expedited text throughput by setting the terminal to not do automatic <carriage-return>s when printing more than one logical line of output. This option was omitted from POSIX.1-200x because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.
- ruler** The **ruler** edit option has been used in some implementations of *vi* to present a current row/column ruler for the user. This option was omitted from POSIX.1-200x because it is not widespread historical practice.
- sourceany** The **sourceany** edit option historically caused *ex* or *vi* to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.
- timeout** The **timeout** edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from POSIX.1-200x because its behavior is now standard, it is not widely useful, and it was rarely documented.
- verbose** The **verbose** edit option has been used in some implementations of *vi* to cause *vi* to output error messages for common errors; for example, attempting to move the cursor past the beginning or end of the line instead of only alerting the screen. (The historical *vi* only alerted the terminal and presented no message for such errors. The historical editor option **terse** did not select when to present error messages, it only made existing error messages more or less verbose.) This option was omitted from POSIX.1-200x because it is not widespread historical practice; however, implementors are encouraged to use it if they wish to provide error messages for naive users.
- wraplen** The **wraplen** edit option has been used in some implementations of *vi* to specify an automatic margin measured from the left margin instead of from the right margin. This is useful when multiple screen sizes are being used to edit a single file. This

86259 option was omitted from POSIX.1-200x because it is not widespread historical
86260 practice; however, implementors are encouraged to use it if they add this
86261 functionality.

86262 **autoindent, ai**

86263 Historically, the command **0a** did not do any autoindentation, regardless of the current
86264 indentation of line 1. POSIX.1-200x requires that any indentation present in line 1 be used.

86265 **autoprint, ap**

86266 Historically, the **autoprint** edit option was not completely consistent or based solely on
86267 modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,
86268 but not from a filter), the **append, change, insert, global,** and **v** commands, all of which were not
86269 affected by **autoprint**, and the **tag** command, which was affected by **autoprint**. POSIX.1-200x
86270 requires conformance to historical practice.

86271 Historically, the **autoprint** option only applied to the last of multiple commands entered using
86272 vertical-bar delimiters; for example, **delete <newline>** was affected by **autoprint**, but
86273 **delete | version <newline>** was not. POSIX.1-200x requires conformance to historical practice.

86274 **autowrite, aw**

86275 Appending the '!' character to the *ex* **next** command to avoid performing an automatic write
86276 was not supported in historical implementations. POSIX.1-200x requires that the behavior match
86277 the other *ex* commands for consistency.

86278 **ignorecase, ic**

86279 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to
86280 counterintuitive situations when uppercase characters were used in range expressions.
86281 Historically, the process was as follows:

- 86282 1. Take a line of text from the edit buffer.
- 86283 2. Convert uppercase to lowercase in text line.
- 86284 3. Convert uppercase to lowercase in regular expressions, except in character class
86285 specifications.
- 86286 4. Match regular expressions against text.

86287 This would mean that, with **ignorecase** in effect, the text:

86288 The cat sat on the mat

86289 would be matched by

86290 /[^]the/

86291 but not by:

86292 /[^][A-Z]he/

86293 For consistency with other commands implementing regular expressions, POSIX.1-200x does not
86294 permit this behavior.

86295

paragraphs, para

86296

86297

86298

86299

86300

86301

The ISO POSIX-2:1993 standard made the default **paragraphs** and **sections** edit options implementation-defined, arguing they were historically oriented to the UNIX system *troff* text formatter, and a “portable user” could use the {, }, [[,]], (, and) commands in open or visual mode and have the cursor stop in unexpected places. POSIX.1-200x specifies their values in the POSIX locale because the unusual grouping (they only work when grouped into two characters at a time) means that they cannot be used for general-purpose movement, regardless.

86302

readonly

86303

86304

86305

86306

Implementations are encouraged to provide the best possible information to the user as to the read-only status of the file, with the exception that they should not consider the current special privileges of the process. This provides users with a safety net because they must force the overwrite of read-only files, even when running with additional privileges.

86307

86308

86309

86310

86311

The **readonly** edit option specification largely conforms to historical practice. The only difference is that historical implementations did not notice that the user had set the **readonly** edit option in cases where the file was already marked read-only for some reason, and would therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were replaced. This behavior is disallowed by POSIX.1-200x.

86312

report

86313

86314

86315

The requirement that lines copied to a buffer interact differently than deleted lines is historical practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be written, but 4 lines must be copied before a report is written.

86316

86317

86318

86319

86320

86321

86322

86323

86324

The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based on the total number of lines added or deleted during the command execution, and that commands executed by the **global** and **v** commands not present reports, is historical practice. POSIX.1-200x extends historical practice by requiring that buffer execution be treated similarly. The reasons for this are two-fold. Historically, only the report by the last command executed from the buffer would be seen by the user, as each new report would overwrite the last. In addition, the standard developers believed that buffer execution had more in common with **global** and **v** commands than it did with other *ex* commands, and should behave similarly, for consistency and simplicity of specification.

86325

showmatch, sm

86326

86327

86328

86329

86330

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals the cursor pauses on the matching character.

86331

showmode

86332

86333

86334

86335

86336

86337

86338

86339

86340

The **showmode** option has been used in some historical implementations of *ex* and *vi* to display the current editing mode when in open or visual mode. The editing modes have generally included “command” and “input”, and sometimes other modes such as “replace” and “change”. The string was usually displayed on the bottom line of the screen at the far right-hand corner. In addition, a preceding ‘*’ character often denoted whether the contents of the edit buffer had been modified. The latter display has sometimes been part of the **showmode** option, and sometimes based on another option. This option was not available in the 4 BSD historical implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is required by POSIX.1-200x.

86341 The **smd** shorthand for the **showmode** option was not present in all historical implementations
86342 of the editor. POSIX.1-200x requires it, for consistency.

86343 Not all historical implementations of the editor displayed a mode string for command mode,
86344 differentiating command mode from text input mode by the absence of a mode string.
86345 POSIX.1-200x permits this behavior for consistency with historical practice, but implementations
86346 are encouraged to provide a display string for both modes.

86347 **slowopen**

86348 Historically the **slowopen** option was automatically set if the terminal baud rate was less than
86349 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen**
86350 option had two effects. First, when inserting characters in the middle of a line, characters after
86351 the cursor would not be pushed ahead, but would appear to be overwritten. Second, when
86352 creating a new line of text, lines after the current line would not be scrolled down, but would
86353 appear to be overwritten. In both cases, ending text input mode would cause the screen to be
86354 refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently
86355 intelligent caused the editor to ignore the **slowopen** option. POSIX.1-200x permits most
86356 historical behavior, extending historical practice to require **slowopen** behaviors if the edit option
86357 is set by the user.

86358 **tags**

86359 The default path for tags files is left unspecified as implementations may have their own **tags**
86360 implementations that do not correspond to the historical ones. The default **tags** option value
86361 should probably at least include the file **./tags**.

86362 **term**

86363 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial
86364 terminal information was loaded. This is permitted by POSIX.1-200x; however, implementations
86365 are encouraged to permit the user to modify their terminal type at any time.

86366 **terse**

86367 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,
86368 for some error messages. This is permitted, but not required, by POSIX.1-200x. Historically, most
86369 common visual mode errors (for example, trying to move the cursor past the end of a line) did
86370 not result in an error message, but simply alerted the terminal. Implementations wishing to
86371 provide messages for novice users are urged to do so based on the **edit** option **verbose**, and not
86372 **terse**.

86373 **window**

86374 In historical implementations, the default for the **window** edit option was based on the baud
86375 rate as follows:

- 86376 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for
86377 example, the line:
86378

```
set w300=12
```


86379 would set the window option to 12 if the baud rate was less than 1 200.
- 86380 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.
- 86381 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

86382 The **w300**, **w1200**, and **w9600** options do not appear in POSIX.1-200x because of their
86383 dependence on specific baud rates.

86384 In historical implementations, the size of the window displayed by various commands was
 86385 related to, but not necessarily the same as, the **window** edit option. For example, the size of the
 86386 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**
 86387 edit option. However, changing the value of the **window** edit option did change the number of
 86388 lines that were displayed when the screen was repainted. POSIX.1-200x does not permit this
 86389 behavior in the interests of consistency and simplicity of specification, and requires that all
 86390 commands that change the number of lines that are displayed do it by setting the value of the
 86391 **window** edit option.

86392 **wrapmargin, wm**

86393 Historically, the **wrapmargin** option did not affect maps inserting characters that also had
 86394 associated *counts*; for example **:map K 5aABC DEF**. Unfortunately, there are widely used
 86395 maps that depend on this behavior. For consistency and simplicity of specification,
 86396 POSIX.1-200x does not permit this behavior.

86397 Historically, **wrapmargin** was calculated using the column display width of all characters on the
 86398 screen. For example, an implementation using "**^I**" to represent <tab>s when the **list** edit
 86399 option was set, where '**^**' and '**I**' each took up a single column on the screen, would calculate
 86400 the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option similarly
 86401 changed the effective length of the line as well. POSIX.1-200x requires conformance to historical
 86402 practice.

86403 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 86404 but this has been modified in this version.

86405 **FUTURE DIRECTIONS**

86406 None.

86407 **SEE ALSO**

86408 [Section 2.9.1.1](#) (on page 2264), *ctags*, *ed*, *sed*, *sh*, *stty*, *vi*

86409 XBD [Table 5-1](#) (on page 108), [Chapter 8](#) (on page 159), [Section 9.3](#) (on page 169), [Section 12.2](#) (on
 86410 page 201)

86411 XSH [access\(\)](#)

86412 **CHANGE HISTORY**

86413 First released in Issue 2.

86414 **Issue 5**

86415 The FUTURE DIRECTIONS section is added.

86416 **Issue 6**

86417 This utility is marked as part of the User Portability Utilities option.

86418 The obsolescent SYNOPSIS is removed, removing the *+command* and *-* options.

86419 The following new requirements on POSIX implementations derive from alignment with the
 86420 Single UNIX Specification:

- 86421 • In the **map** command description, the sequence *#digit* is added.
- 86422 • The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

86423 The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This
 86424 includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52,
 86425 #55, #56, #57, #61, #62, #63, #64, #65, and #78.

86426 The **-l** option is removed.

86427 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/23 is applied, correcting a URL.

- 86428 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/8 is applied, making an editorial
86429 correction in the EXTENDED DESCRIPTION.
- 86430 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/9 is applied, removing text describing
86431 behavior on systems with bytes consisting of more than eight bits.
- 86432 **Issue 7**
- 86433 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if an operand is
86434 '–'.
- 86435 Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.
- 86436 Austin Group Interpretation 1003.1-2001 #121 is applied, clarifying the *ex write* command. +
- 86437 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



86438 **NAME**86439 `expand` — convert tabs to spaces86440 **SYNOPSIS**86441 `expand [-t tablist] [file...]`86442 **DESCRIPTION**

86443 The *expand* utility shall write files or the standard input to the standard output with `<tab>`s
 86444 replaced with one or more `<space>`s needed to pad to the next tab stop. Any `<backspace>`s shall
 86445 be copied to the output and cause the column position count for tab stop calculations to be
 86446 decremented; the column position count shall not be decremented below zero.

86447 **OPTIONS**86448 The *expand* utility shall conform to XBD [Section 12.2](#) (on page 201). |

86449 The following option shall be supported:

86450 `-t tablist` Specify the tab stops. The application shall ensure that the argument *tablist* consists
 86451 of either a single positive decimal integer or a list of tabstops. If a single number is
 86452 given, tabs shall be set that number of column positions apart instead of the
 86453 default 8.

86454 If a list of tabstops is given, the application shall ensure that it consists of a list of
 86455 two or more positive decimal integers, separated by `<blank>`s or commas, in
 86456 ascending order. The tabs shall be set at those specific column positions. Each tab
 86457 stop *N* shall be an integer value greater than zero, and the list is in strictly
 86458 ascending order. This is taken to mean that, from the start of a line of output,
 86459 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th
 86460 column position on that line.

86461 In the event of *expand* having to process a `<tab>` at a position beyond the last of
 86462 those specified in a multiple tab-stop list, the `<tab>` shall be replaced by a single
 86463 `<space>` in the output.

86464 **OPERANDS**

86465 The following operand shall be supported:

86466 `file` The pathname of a text file to be used as input.86467 **STDIN**

86468 See the INPUT FILES section.

86469 **INPUT FILES**

86470 Input files shall be text files.

86471 **ENVIRONMENT VARIABLES**86472 The following environment variables shall affect the execution of *expand*:

86473 `LANG` Provide a default value for the internationalization variables that are unset or null. |
 86474 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 86475 variables used to determine the values of locale categories.)

86476 `LC_ALL` If set to a non-empty string value, override the values of all the other
 86477 internationalization variables.

86478 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
 86479 characters (for example, single-byte as opposed to multi-byte characters in
 86480 arguments and input files), the processing of `<tab>`s and `<space>`s, and for the
 86481 determination of the width in column positions each character would occupy on

86482 an output device.

86483 *LC_MESSAGES*

86484 Determine the locale that should be used to affect the format and contents of
86485 diagnostic messages written to standard error.

86486 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86487 ASYNCHRONOUS EVENTS

86488 Default.

86489 STDOUT

86490 The standard output shall be equivalent to the input files with <tab>s converted into the
86491 appropriate number of <space>s.

86492 STDERR

86493 The standard error shall be used only for diagnostic messages.

86494 OUTPUT FILES

86495 None.

86496 EXTENDED DESCRIPTION

86497 None.

86498 EXIT STATUS

86499 The following exit values shall be returned:

86500 0 Successful completion

86501 >0 An error occurred.

86502 CONSEQUENCES OF ERRORS

86503 The *expand* utility shall terminate with an error message and non-zero exit status upon
86504 encountering difficulties accessing one of the *file* operands.

86505 APPLICATION USAGE

86506 None.

86507 EXAMPLES

86508 None.

86509 RATIONALE

86510 The *expand* utility is useful for preprocessing text files (before sorting, looking at specific
86511 columns, and so on) that contain <tab>s.

86512 See XBD [Section 3.102](#) (on page 47). |

86513 The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8
86514 mandates that *expand* shall accept the integers (within the single argument) separated using
86515 either commas or <blank>s.

86516 Earlier versions of this standard allowed the following form in the SYNOPSIS:

86517 `expand [-tabstop][-tab1,tab2,...,tabn][file ...]`

86518 This form is no longer specified by POSIX.1-200x but may be present in some implementations. |

86519 FUTURE DIRECTIONS

86520 None.

86521 SEE ALSO

86522 *tabs*, *unexpand*

86523 XBD [Section 3.102](#) (on page 47), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

86524

CHANGE HISTORY

86525

First released in Issue 4.

86526

Issue 6

86527

This utility is marked as part of the User Portability Utilities option.

86528

The APPLICATION USAGE section is added.

86529

The obsolescent SYNOPSIS is removed.

86530

The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b draft standard.

86531

86532

The normative text is reworded to avoid use of the term “must” for application requirements.

86533

Issue 7

86534

Austin Group Interpretation 1003.1-2001 #027 is applied.

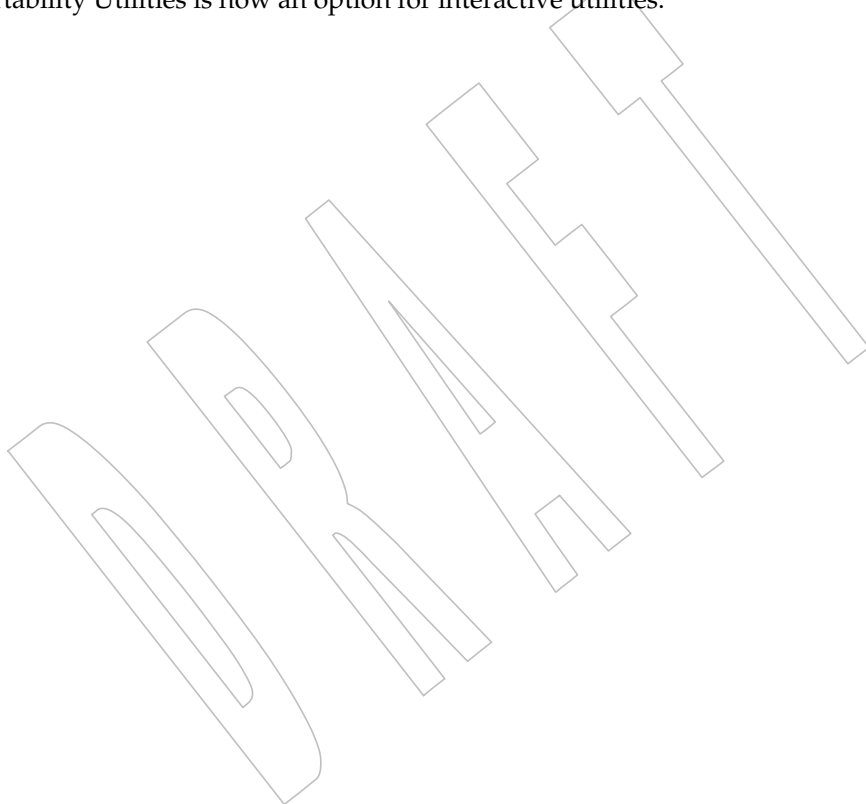
86535

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS. +

86536

The *expand* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities. -

86537



86538 **NAME**
 86539 `expr` — evaluate arguments as an expression

86540 **SYNOPSIS**
 86541 `expr operand...`

86542 **DESCRIPTION**
 86543 The `expr` utility shall evaluate an expression and write the result to standard output.

86544 **OPTIONS**
 86545 None.

86546 **OPERANDS**
 86547 The single expression evaluated by `expr` shall be formed from the `operand` operands, as described
 86548 in the EXTENDED DESCRIPTION section. The application shall ensure that each of the
 86549 expression operator symbols:

86550 () | & = > >= < <= != + - * / % :

86551 and the symbols `integer` and `string` in the table are provided as separate arguments to `expr`.

86552 **STDIN**
 86553 Not used.

86554 **INPUT FILES**
 86555 None.

86556 **ENVIRONMENT VARIABLES**
 86557 The following environment variables shall affect the execution of `expr`:

86558 `LANG` Provide a default value for the internationalization variables that are unset or null.
 86559 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 86560 variables used to determine the values of locale categories.)

86561 `LC_ALL` If set to a non-empty string value, override the values of all the other
 86562 internationalization variables.

86563 `LC_COLLATE`
 86564 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 86565 character collating elements within regular expressions and by the string
 86566 comparison operators.

86567 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as
 86568 characters (for example, single-byte as opposed to multi-byte characters in
 86569 arguments) and the behavior of character classes within regular expressions.

86570 `LC_MESSAGES`
 86571 Determine the locale that should be used to affect the format and contents of
 86572 diagnostic messages written to standard error.

86573 XSI `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

86574 **ASYNCHRONOUS EVENTS**
 86575 Default.

86576 **STDOUT**
 86577 The `expr` utility shall evaluate the expression and write the result, followed by a <newline>, to
 86578 standard output.

86579 **STDERR**
 86580 The standard error shall be used only for diagnostic messages.

86581 **OUTPUT FILES**
 86582 None.

86583 EXTENDED DESCRIPTION

86584 The formation of the expression to be evaluated is shown in the following table. The symbols
 86585 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
 86586 expression operator symbols (all separate arguments) by recursive application of the constructs
 86587 described in the table. The expressions are listed in order of increasing precedence, with equal-
 86588 precedence operators grouped between horizontal lines. All of the operators shall be left-
 86589 associative.

Expression	Description
<i>expr1</i> <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> if it is not null; otherwise, zero.
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i> <i>expr1</i> % <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result. Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> : <i>expr2</i>	Matching expression; see below.
(<i>expr</i>)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>integer</i>	An argument consisting only of an (optional) unary minus followed by digits.
<i>string</i>	A string argument; see below.

86621

Matching Expression86622
86623
86624
86625
86626
86627
86628
86629
86630
86631

The `' : '` matching operator shall compare the string resulting from the evaluation of *expr1* with the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax shall be that defined in XBD [Section 9.3](#) (on page 169), except that all patterns are anchored to the beginning of the string (that is, only sequences starting at the first character of a string are matched by the regular expression) and, therefore, it is unspecified whether `' ^ '` is a special character in that context. Usually, the matching operator shall return a string representing the number of characters matched (`' 0 '` on failure). Alternatively, if the pattern contains at least one regular expression subexpression `"[\(\.\.\.\)"]`, the string matched by the back-reference expression `"\1"` shall be returned. If the back-reference expression `"\1"` does not match, then the null string shall be returned.

86632

String Operand86633
86634

A string argument is an argument that cannot be identified as an *integer* argument or as one of the expression operator symbols shown in the OPERANDS section.

86635

The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

86636

EXIT STATUS

86637

The following exit values shall be returned:

86638

0 The *expression* evaluates to neither null nor zero.

86639

1 The *expression* evaluates to null or zero.

86640

2 Invalid *expression*.

86641

>2 An error occurred.

86642

CONSEQUENCES OF ERRORS

86643

Default.

86644

APPLICATION USAGE

86645

After argument processing by the shell, *expr* is not required to be able to tell the difference between an operator and an operand except by the value. If `"$a"` is `' = '`, the command:

86646

```
expr $a = '='
```

86648

looks like:

86649

```
expr = = =
```

86650

as the arguments are passed to *expr* (and they all may be taken as the `' = '` operator). The following works reliably:

86651

```
expr X$a = X=
```

86652

86653

Also note that this volume of POSIX.1-200x permits implementations to extend utilities. The *expr* utility permits the integer arguments to be preceded with a unary minus. This means that an integer argument could look like an option. Therefore, the conforming application must employ the `"--"` construct of Guideline 10 of XBD [Section 12.2](#) (on page 201) to protect its operands if there is any chance the first operand might be a negative integer (or any string with a leading minus).

86654

86655

86656

86657

86658

86659

EXAMPLES

86660

The *expr* utility has a rather difficult syntax:

86661

- Many of the operators are also shell control operators or reserved words, so they have to be escaped on the command line.

86662

- Each part of the expression is composed of separate arguments, so liberal usage of <blank>s is required. For example:

Invalid	Valid
<code>expr 1+2</code>	<code>expr 1 + 2</code>
<code>expr "1 + 2"</code>	<code>expr 1 + 2</code>
<code>expr 1 + (2 * 3)</code>	<code>expr 1 + \(2 * 3 \)</code>

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see [Section 2.5](#) (on page 2249) and [Section 2.6.4](#) (on page 2257).

The following command:

```
a=$(expr $a + 1)
```

adds 1 to the variable *a*.

The following command, for "*\$a*" equal to either */usr/abc/file* or just *file*:

```
expr $a : '.*\/(.*)' \| $a
```

returns the last segment of a pathname (that is, *file*). Applications should avoid the character */* used alone as an argument; *expr* may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*\/(.*)'
```

is a better representation of the previous example. The addition of the *///* characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the *IFS* variable and should be quoted to avoid having "*\$a*" expand into multiple arguments.

The following command:

```
expr "$VAR" : '.*'
```

returns the number of characters in *VAR*.

RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading circumflex in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo      expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 2.5](#) (on page 2249), [Section 2.6.4](#) (on page 2257)

[XBD Chapter 8](#) (on page 159), [Section 9.3](#) (on page 169), [Section 12.2](#) (on page 201)

+

86704

CHANGE HISTORY

86705

First released in Issue 2.

86706

Issue 5

86707

The FUTURE DIRECTIONS section is added.

86708

Issue 6

86709

The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE PASC Interpretation 1003.2 #104.

86710

86711

The normative text is reworded to avoid use of the term “must” for application requirements.

86712

Issue 7

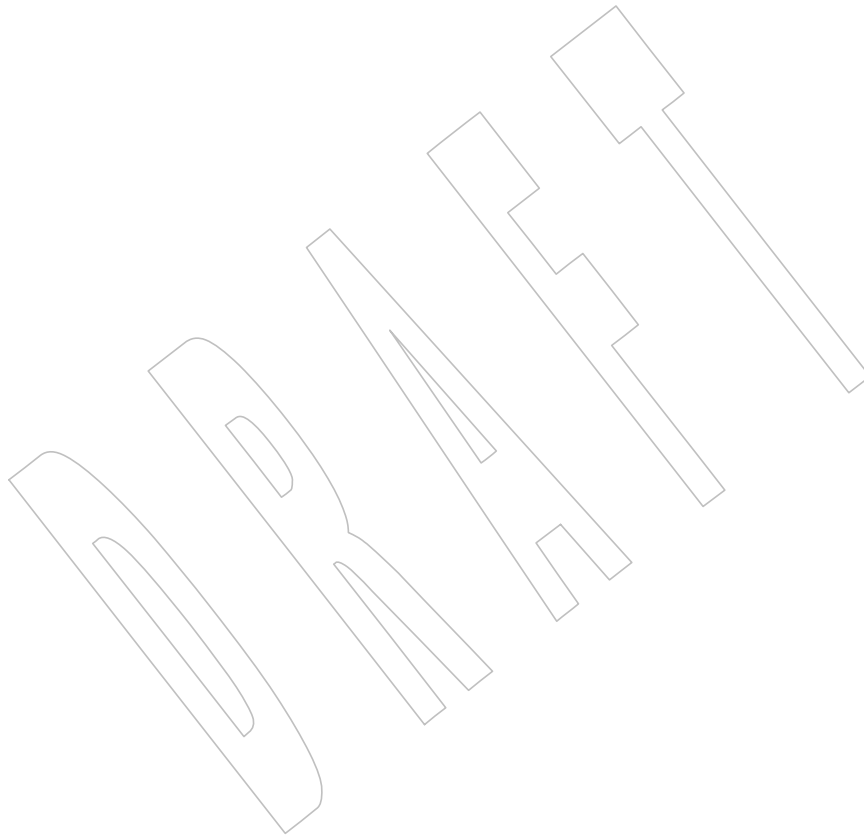
86713

Austin Group Interpretation 1003.1-2001 #036 is applied, clarifying the behavior for BREs.

86714

The SYNOPSIS and OPERANDS sections are revised to explicitly state that the name of each of the operands is *operand*.

86715



86716 **NAME**
86717 false — return false value

86718 **SYNOPSIS**
86719 false

86720 **DESCRIPTION**
86721 The *false* utility shall return with a non-zero exit code.

86722 **OPTIONS**
86723 None.

86724 **OPERANDS**
86725 None.

86726 **STDIN**
86727 Not used.

86728 **INPUT FILES**
86729 None.

86730 **ENVIRONMENT VARIABLES**
86731 None.

86732 **ASYNCHRONOUS EVENTS**
86733 Default.

86734 **STDOUT**
86735 Not used.

86736 **STDERR**
86737 Not used.

86738 **OUTPUT FILES**
86739 None.

86740 **EXTENDED DESCRIPTION**
86741 None.

86742 **EXIT STATUS**
86743 The *false* utility shall always exit with a value other than zero.

86744 **CONSEQUENCES OF ERRORS**
86745 Default.

86746 **APPLICATION USAGE**
86747 None.

86748 **EXAMPLES**
86749 None.

86750 **RATIONALE**
86751 None.

86752 **FUTURE DIRECTIONS**
86753 None.

86754

SEE ALSO

86755

true

86756

CHANGE HISTORY

86757

First released in Issue 2.

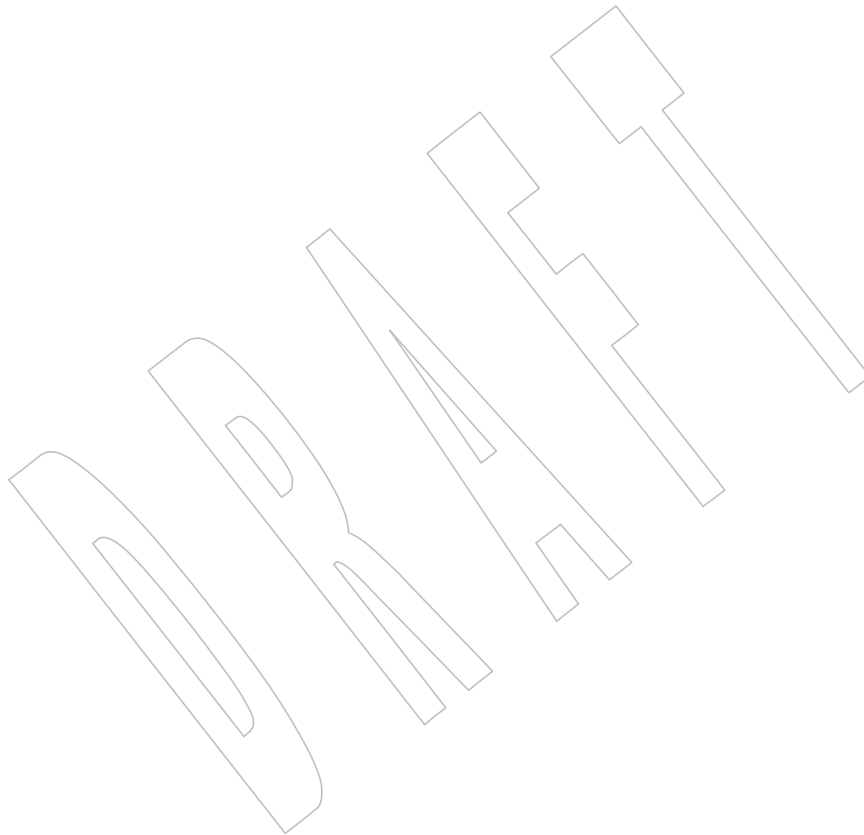
86758

Issue 6

86759

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/24 is applied, changing the STDERR section from “None.” to “Not used.” for alignment with [Section 1.4](#) (on page 2235).

86760



86761 **NAME**
86762 `fc` — process the command history list

86763 **SYNOPSIS**

86764 UP `fc [-r] [-e editor] [first [last]]`
86765 `fc -l [-nr] [first [last]]`
86766 `fc -s [old=new] [first]`

86767 **DESCRIPTION**

86768 The `fc` utility shall list, or shall edit and re-execute, commands previously entered to an
86769 interactive `sh`.

86770 The command history list shall reference commands by number. The first number in the list is
86771 selected arbitrarily. The relationship of a number to its command shall not change except when
86772 the user logs in and no other process is accessing the list, at which time the system may reset the
86773 numbering to start the oldest retained command at another number (usually 1). When the
86774 number reaches an implementation-defined upper limit, which shall be no smaller than the
86775 value in `HISTSIZE` or 32767 (whichever is greater), the shell may wrap the numbers, starting the
86776 next command with a lower number (usually 1). However, despite this optional wrapping of
86777 numbers, `fc` shall maintain the time-ordering sequence of the commands. For example, if four
86778 commands in sequence are given the numbers 32766, 32767, 1 (wrapped), and 2 as they are
86779 executed, command 32767 is considered the command previous to 1, even though its number is
86780 higher.

86781 When commands are edited (when the `-l` option is not specified), the resulting lines shall be
86782 entered at the end of the history list and then re-executed by `sh`. The `fc` command that caused the
86783 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this
86784 shall suppress the entry into the history list and the command re-execution. Any command line
86785 variable assignments or redirection operators used with `fc` shall affect both the `fc` command itself
86786 as well as the command that results; for example:

86787 `fc -s -- -l 2>/dev/null`

86788 reinvokes the previous command, suppressing standard error for both `fc` and the previous
86789 command.

86790 **OPTIONS**

86791 The `fc` utility shall conform to XBD [Section 12.2](#) (on page 201).

86792 The following options shall be supported:

86793 `-e editor` Use the editor named by `editor` to edit the commands. The `editor` string is a utility
86794 name, subject to search via the `PATH` variable (see XBD [Chapter 8](#), on page 159).
86795 The value in the `FCEDIT` variable shall be used as a default when `-e` is not
86796 specified. If `FCEDIT` is null or unset, `ed` shall be used as the editor.

86797 `-l` (The letter ell.) List the commands rather than invoking an editor on them. The
86798 commands shall be written in the sequence indicated by the `first` and `last` operands,
86799 as affected by `-r`, with each command preceded by the command number.

86800 `-n` Suppress command numbers when listing with `-l`.

86801 `-r` Reverse the order of the commands listed (with `-l`) or edited (with neither `-l` nor
86802 `-s`).

86803 -s Re-execute the command without invoking an editor.

86804 OPERANDS

86805 The following operands shall be supported:

86806 *first, last* Select the commands to list or edit. The number of previous commands that can be
86807 accessed shall be determined by the value of the *HISTSIZE* variable. The value of
86808 *first* or *last* or both shall be one of the following:

86809 [+]*number* A positive number representing a command number; command
86810 numbers can be displayed with the -l option.

86811 -*number* A negative decimal number representing the command that was
86812 executed *number* of commands previously. For example, -1 is the
86813 immediately previous command.

86814 *string* A string indicating the most recently entered command that begins
86815 with that string. If the *old=new* operand is not also specified with -s,
86816 the string form of the *first* operand cannot contain an embedded
86817 equal sign.

86818 When the synopsis form with -s is used:

86819 • If *first* is omitted, the previous command shall be used.

86820 For the synopsis forms without -s:

86821 • If *last* is omitted, *last* shall default to the previous command when -l is
86822 specified; otherwise, it shall default to *first*.

86823 • If *first* and *last* are both omitted, the previous 16 commands shall be listed or
86824 the previous single command shall be edited (based on the -l option).

86825 • If *first* and *last* are both present, all of the commands from *first* to *last* shall be
86826 edited (without -l) or listed (with -l). Editing multiple commands shall be
86827 accomplished by presenting to the editor all of the commands at one time,
86828 each command starting on a new line. If *first* represents a newer command
86829 than *last*, the commands shall be listed or edited in reverse sequence,
86830 equivalent to using -r. For example, the following commands on the first
86831 line are equivalent to the corresponding commands on the second:

86832 fc -r 10 20 fc 30 40
86833 fc 20 10 fc -r 40 30

86834 • When a range of commands is used, it shall not be an error to specify *first* or
86835 *last* values that are not in the history list; *fc* shall substitute the value
86836 representing the oldest or newest command in the list, as appropriate. For
86837 example, if there are only ten commands in the history list, numbered 1 to 10:

86838 fc -l
86839 fc 1 99

86840 shall list and edit, respectively, all ten commands.

86841 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
86842 string *new*.

86843 STDIN

86844 Not used.

86845 **INPUT FILES**

86846 None.

86847 **ENVIRONMENT VARIABLES**86848 The following environment variables shall affect the execution of *fc*:

86849 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for
 86850 the *-e editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
 86851 used as the editor.

86852 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
 86853 not set, the shell may attempt to access or create a file *.sh_history* in the directory
 86854 referred to by the *HOME* environment variable. If the shell cannot obtain both read
 86855 and write access to, or create, the history file, it shall use an unspecified
 86856 mechanism that allows the history to operate properly. (References to history "file"
 86857 in this section shall be understood to mean this unspecified mechanism in such
 86858 cases.) An implementation may choose to access this variable only when
 86859 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
 86860 to retrieve entries from, or add entries to, the file, as the result of commands issued
 86861 by the user, the file named by the *ENV* variable, or implementation-defined system
 86862 start-up files. In some historical shells, the history file is initialized just after the
 86863 *ENV* file has been processed. Therefore, it is implementation-defined whether
 86864 changes made to *HISTFILE* after the history file has been initialized are effective.
 86865 Implementations may choose to disable the history list mechanism for users with
 86866 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
 86867 which this occurs are implementation-defined. If more than one instance of the
 86868 shell is using the same history file, it is unspecified how updates to the history file
 86869 from those shells interact. As entries are deleted from the history file, they shall be
 86870 deleted oldest first. It is unspecified when history file entries are physically
 86871 removed from the history file.

86872 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
 86873 commands that are accessible. If this variable is unset, an unspecified default
 86874 greater than or equal to 128 shall be used. The maximum number of commands in
 86875 the history list is unspecified, but shall be at least 128. An implementation may
 86876 choose to access this variable only when initializing the history file, as described
 86877 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
 86878 after the history file has been initialized are effective.

86879 *LANG* Provide a default value for the internationalization variables that are unset or null.
 86880 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 86881 variables used to determine the values of locale categories.)

86882 *LC_ALL* If set to a non-empty string value, override the values of all the other
 86883 internationalization variables.

86884 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 86885 characters (for example, single-byte as opposed to multi-byte characters in
 86886 arguments and input files).

86887 *LC_MESSAGES*

86888 Determine the locale that should be used to affect the format and contents of
 86889 diagnostic messages written to standard error.

86890 *XSHELL* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

86891 **ASYNCHRONOUS EVENTS**

86892 Default.

86893 **STDOUT**86894 When the `-l` option is used to list commands, the format of each command in the list shall be as
86895 follows:86896 "`%d\t%s\n`", *<line number>*, *<command>*86897 If both the `-l` and `-n` options are specified, the format of each command shall be:86898 "`\t%s\n`", *<command>*86899 If the *<command>* consists of more than one line, the lines after the first shall be displayed as:86900 "`\t%s\n`", *<continued-command>*86901 **STDERR**

86902 The standard error shall be used only for diagnostic messages.

86903 **OUTPUT FILES**

86904 None.

86905 **EXTENDED DESCRIPTION**

86906 None.

86907 **EXIT STATUS**

86908 The following exit values shall be returned:

86909 0 Successful completion of the listing.

86910 >0 An error occurred.

86911 Otherwise, the exit status shall be that of the commands executed by *fc*.86912 **CONSEQUENCES OF ERRORS**

86913 Default.

86914 **APPLICATION USAGE**86915 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
86916 descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the
86917 *FCEDIT* editor, the command:86918 `fc -s | more`

86919 does not work correctly on many systems.

86920 Users on windowing systems may want to have separate history files for each window by
86921 setting *HISTFILE* as follows:86922 `HISTFILE=$HOME/.sh_hist$$`86923 **EXAMPLES**

86924 None.

86925 **RATIONALE**86926 This utility is based on the *fc* built-in of the KornShell.86927 An early proposal specified the `-e` option as [`-e editor [old= new]`], which is not historical
86928 practice. Historical practice in *fc* of either [`-e editor`] or [`-e - [old= new]`] is acceptable, but not
86929 both together. To clarify this, a new option `-s` was introduced replacing the [`-e -`]. This resolves
86930 the conflict and makes *fc* conform to the Utility Syntax Guidelines.86931 *HISTFILE* Some implementations of the KornShell check for the superuser and do not create
86932 a history file unless *HISTFILE* is set. This is done primarily to avoid creating
86933 unlinked files in the root file system when logging in during single-user mode.

86934 *HISTFILE* must be set for the superuser to have history.

86935 *HISTSIZE* Needed to limit the size of history files. It is the intent of the standard developers
86936 that when two shells share the same history file, commands that are entered in one
86937 shell shall be accessible by the other shell. Because of the difficulties of
86938 synchronization over a network, the exact nature of the interaction is unspecified.

86939 The initialization process for the history file can be dependent on the system start-up files, in
86940 that they may contain commands that effectively preempt the settings the user has for *HISTFILE*
86941 and *HISTSIZE*. For example, function definition commands are recorded in the history file. If
86942 the system administrator includes function definitions in some system start-up file called before
86943 the *ENV* file, the history file is initialized before the user can influence its characteristics. In some
86944 historical shells, the history file is initialized just after the *ENV* file has been processed. Because
86945 of these situations, the text requires the initialization process to be implementation-defined.

86946 Consideration was given to omitting the *fc* utility in favor of the command line editing feature in
86947 *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

86948 EDITOR=vi fc

86949 However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously
86950 (such as *fc* 10 20) and to use editors other than those supported by *sh* for command line editing.

86951 In the KornShell, the alias *r* ("re-do") is preset to *fc -e -* (equivalent to the POSIX *fc -s*). This is
86952 probably an easier command name to remember than *fc* ("fix command"), but it does not meet
86953 the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this
86954 description closely matches historical KornShell practice already, such a renaming was seen as
86955 gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*,
86956 *grep*, or *yacc* are standardized by POSIX.

86957 Command numbers have no ordering effects; they are like serial numbers. The *-r* option and
86958 *-number* operand address the sequence of command execution, regardless of serial numbers. So,
86959 for example, if the command number wrapped back to 1 at some arbitrary point, there would be
86960 no ambiguity associated with traversing the wrap point. For example, if the command history
86961 were:

```
86962 32766: echo 1
86963 32767: echo 2
86964 1: echo 3
```

86965 the number *-2* refers to command 32767 because it is the second previous command, regardless
86966 of serial number.

86967 FUTURE DIRECTIONS

86968 None.

86969 SEE ALSO

86970 *sh*

86971 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

86972 CHANGE HISTORY

86973 First released in Issue 4.

86974 Issue 5

86975 The FUTURE DIRECTIONS section is added.

86976 Issue 6

86977 This utility is marked as part of the User Portability Utilities option.

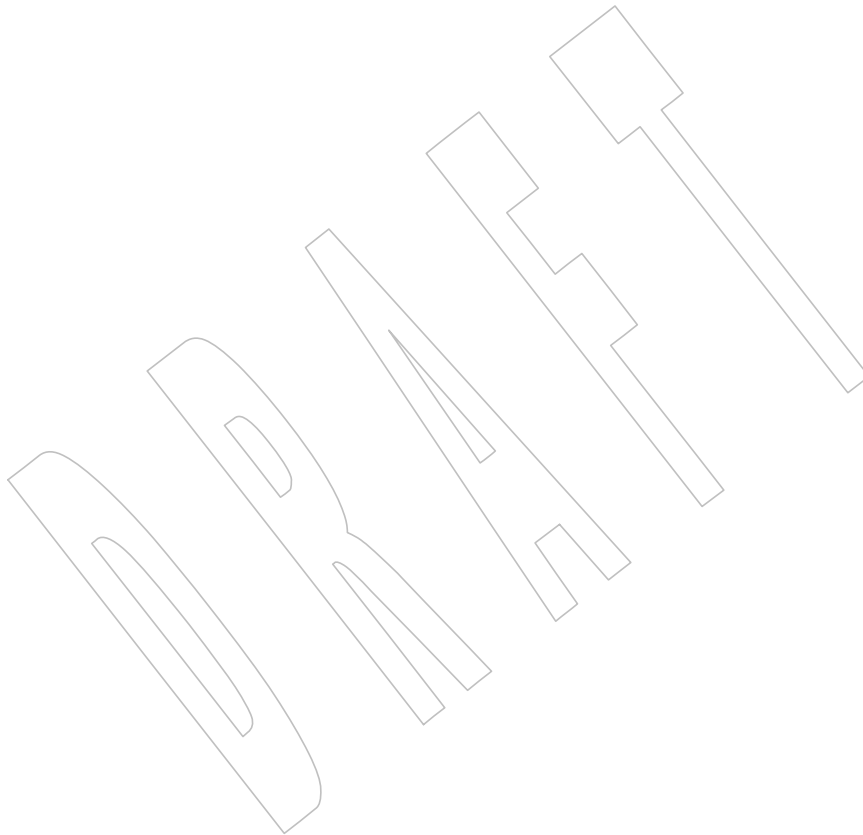
86978 In the ENVIRONMENT VARIABLES section, the text "user's home directory" is updated to
86979 "directory referred to by the *HOME* environment variable".

86980

Issue 7

86981

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



86982 **NAME**

86983 fg — run jobs in the foreground

86984 **SYNOPSIS**86985 UP fg [*job_id*]86986 **DESCRIPTION**86987 If job control is enabled (see the description of *set -m*), the *fg* utility shall move a background job
86988 from the current environment (see [Section 2.12](#), on page 2277) into the foreground.86989 Using *fg* to place a job into the foreground shall remove its process ID from the list of those
86990 “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page 2266).86991 **OPTIONS**

86992 None.

86993 **OPERANDS**

86994 The following operand shall be supported:

86995 *job_id* Specify the job to be run as a foreground job. If no *job_id* operand is given, the
86996 *job_id* for the job that was most recently suspended, placed in the background, or
86997 run as a background job shall be used. The format of *job_id* is described in XBD
86998 [Section 3.202](#) (on page 61).86999 **STDIN**

87000 Not used.

87001 **INPUT FILES**

87002 None.

87003 **ENVIRONMENT VARIABLES**87004 The following environment variables shall affect the execution of *fg*:87005 *LANG* Provide a default value for the internationalization variables that are unset or null.
87006 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
87007 variables used to determine the values of locale categories.)87008 *LC_ALL* If set to a non-empty string value, override the values of all the other
87009 internationalization variables.87010 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
87011 characters (for example, single-byte as opposed to multi-byte characters in
87012 arguments).87013 *LC_MESSAGES*
87014 Determine the locale that should be used to affect the format and contents of
87015 diagnostic messages written to standard error.87016 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.87017 **ASYNCHRONOUS EVENTS**

87018 Default.

87019 **STDOUT**87020 The *fg* utility shall write the command line of the job to standard output in the following format:

87021 "%s\n", <command>

87022 **STDERR**

87023 The standard error shall be used only for diagnostic messages.

87024 **OUTPUT FILES**

87025 None.

87026 **EXTENDED DESCRIPTION**

87027 None.

87028 **EXIT STATUS**

87029 The following exit values shall be returned:

87030 0 Successful completion.

87031 >0 An error occurred.

87032 **CONSEQUENCES OF ERRORS**87033 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the
87034 foreground.87035 **APPLICATION USAGE**87036 The *fg* utility does not work as expected when it is operating in its own utility execution
87037 environment because that environment has no applicable jobs to manipulate. See the
87038 APPLICATION USAGE section for *bg*. For this reason, *fg* is generally implemented as a shell
87039 regular built-in.87040 **EXAMPLES**

87041 None.

87042 **RATIONALE**87043 The extensions to the shell specified in this volume of POSIX.1-200x have mostly been based on
87044 features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs* are also
87045 based on the KornShell. The standard developers examined the characteristics of the C shell
87046 versions of these utilities and found that differences exist. Despite widespread use of the C shell,
87047 the KornShell versions were selected for this volume of POSIX.1-200x to maintain a degree of
87048 uniformity with the rest of the KornShell features selected (such as the very popular command
87049 line editing features).87050 **FUTURE DIRECTIONS**

87051 None.

87052 **SEE ALSO**87053 [Section 2.9.3.1](#) (on page 2266), [Section 2.12](#) (on page 2277), *bg*, *kill*, *jobs*, *wait*87054 XBD [Section 3.202](#) (on page 61), [Chapter 8](#) (on page 159) +87055 **CHANGE HISTORY**

87056 First released in Issue 4.

87057 **Issue 6**

87058 This utility is marked as part of the User Portability Utilities option.

87059 The APPLICATION USAGE section is added.

87060 The JC marking is removed from the SYNOPSIS since job control is mandatory in this version. |

87061 **NAME**

87062 file — determine file type

87063 **SYNOPSIS**

87064 file [-dh] [-M file] [-m file] file...

87065 file -i [-h] file...

87066 **DESCRIPTION**87067 The *file* utility shall perform a series of tests in sequence on each specified *file* in an attempt to
87068 classify it:

- 87069 1. If *file* does not exist, cannot be read, or its file status could not be determined, the output
87070 shall indicate that the file was processed, but that its type could not be determined.
- 87071 2. If the file is not a regular file, its file type shall be identified. The file types directory,
87072 FIFO, socket, block special, and character special shall be identified as such. Other
87073 implementation-defined file types may also be identified. If *file* is a symbolic link, by
87074 default the link shall be resolved and *file* shall test the type of file referenced by the
87075 symbolic link. (See the **-h** and **-i** options below.)
- 87076 3. If the length of *file* is zero, it shall be identified as an empty file.
- 87077 4. The *file* utility shall examine an initial segment of *file* and shall make a guess at
87078 identifying its contents based on position-sensitive tests. (The answer is not guaranteed to
87079 be correct; see the **-d**, **-M**, and **-m** options below.)
- 87080 5. The *file* utility shall examine *file* and make a guess at identifying its contents based on
87081 context-sensitive default system tests. (The answer is not guaranteed to be correct.)
- 87082 6. The file shall be identified as a data file.

87083 If *file* does not exist, cannot be read, or its file status could not be determined, the output shall
87084 indicate that the file was processed, but that its type could not be determined.87085 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file
87086 referenced by the symbolic link.87087 **OPTIONS**87088 The *file* utility shall conform to XBD [Section 12.2](#) (on page 201), except that the order of the **-m**,
87089 **-d**, and **-M** options shall be significant.

87090 The following options shall be supported by the implementation:

- 87091 **-d** Apply any position-sensitive default system tests and context-sensitive default
87092 system tests to the file. This is the default if no **-M** or **-m** option is specified.
- 87093 **-h** When a symbolic link is encountered, identify the file as a symbolic link. If **-h** is
87094 not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall
87095 identify the file as a symbolic link, as if **-h** had been specified.
- 87096 **-i** If a file is a regular file, do not attempt to classify the type of the file further, but
87097 identify the file as specified in the **STDOUT** section.
- 87098 **-M file** Specify the name of a file containing position-sensitive tests that shall be applied to
87099 a file in order to classify it (see the **EXTENDED DESCRIPTION**). No position-
87100 sensitive default system tests nor context-sensitive default system tests shall be
87101 applied unless the **-d** option is also specified.

87102 **-m file** Specify the name of a file containing position-sensitive tests that shall be applied to
87103 a file in order to classify it (see the EXTENDED DESCRIPTION).

87104 If the **-m** option is specified without specifying the **-d** option or the **-M** option, position-
87105 sensitive default system tests shall be applied after the position-sensitive tests specified by the
87106 **-m** option. If the **-M** option is specified with the **-d** option, the **-m** option, or both, or the **-m**
87107 option is specified with the **-d** option, the concatenation of the position-sensitive tests specified
87108 by these options shall be applied in the order specified by the appearance of these options. If a
87109 **-M** or **-m file** option-argument is **-**, the results are unspecified.

87110 OPERANDS

87111 The following operand shall be supported:

87112 *file* A pathname of a file to be tested.

87113 STDIN

87114 Not used.

87115 INPUT FILES

87116 The *file* can be any file type.

87117 ENVIRONMENT VARIABLES

87118 The following environment variables shall affect the execution of *file*:

87119 **LANG** Provide a default value for the internationalization variables that are unset or null. |
87120 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
87121 variables used to determine the values of locale categories.)

87122 **LC_ALL** If set to a non-empty string value, override the values of all the other
87123 internationalization variables.

87124 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
87125 characters (for example, single-byte as opposed to multi-byte characters in
87126 arguments and input files).

87127 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
87128 diagnostic messages written to standard error and informative messages written to
87129 standard output.
87130

87131 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

87132 ASYNCHRONOUS EVENTS

87133 Default.

87134 STDOUT

87135 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

87136 "**%s: %s\n**", *<file>*, *<type>*

87137 The values for *<type>* are unspecified, except that in the POSIX locale, if *file* is identified as one
87138 of the types listed in the following table, *<type>* shall contain (but is not limited to) the
87139 corresponding string, unless the file is identified by a position-sensitive test specified by a **-M** or
87140 **-m** option. Each space shown in the strings shall be exactly one *<space>*.

87141

Table 4-8 File Utility Output Strings

87142

	If <i>file</i> is:	<type> shall contain the string:	Notes
87143	Nonexistent	cannot open	
87144	Block special	block special	1
87145	Character special	character special	1
87146	Directory	directory	1
87147	FIFO	fifo	1
87148	Socket	socket	1
87149	Symbolic link	symbolic link to	1
87150	Regular file	regular file	1,2
87151	Empty regular file	empty	3
87152	Regular file that cannot be read	cannot open	3
87153	Executable binary	executable	3,4,6
87154	<i>ar</i> archive library (see <i>ar</i>)	archive	3,4,6
87155	Extended <i>cpio</i> format (see <i>pax</i>)	<i>cpio</i> archive	3,4,6
87156	Extended <i>tar</i> format (see ustar in <i>pax</i>)	<i>tar</i> archive	3,4,6
87157	Shell script	commands text	3,5,6
87158	C-language source	c program text	3,5,6
87159	FORTRAN source	fortran program text	3,5,6
87160	Regular file whose type cannot be determined	data	3

87161

Notes:

87162

1. This is a file type test.

87163

2. This test is applied only if the **-i** option is specified.

87164

3. This test is applied only if the **-i** option is not specified.

87165

4. This is a position-sensitive default system test.

87166

5. This is a context-sensitive default system test.

87167

6. Position-sensitive default system tests and context-sensitive default system tests are not applied if the **-M** option is specified unless the **-d** option is also specified.

87168

87169

In the POSIX locale, if *file* is identified as a symbolic link (see the **-h** option), the following alternative output format shall be used:

87170

```
"%s: %s %s\n", <file>, <type>, <contents of link>"
```

87171

87172

If the file named by the *file* operand does not exist, cannot be read, or the type of the file named by the *file* operand cannot be determined, this shall not be considered an error that affects the exit status.

87173

87174

87175

STDERR

87176

The standard error shall be used only for diagnostic messages.

87177

OUTPUT FILES

87178

None.

87179

EXTENDED DESCRIPTION

87180

A file specified as an option-argument to the **-m** or **-M** options shall contain one position-sensitive test per line, which shall be applied to the file. If the test succeeds, the message field of the line shall be printed and no further tests shall be applied, with the exception that tests on immediately following lines beginning with a single **'>'** character shall be applied.

87181

87182

87183

87184

Each line shall be composed of the following four <tab>-separated fields. (Implementations may

87185		allow any combination of one or more white space characters other than <newline> to act as
87186		field separators.)
87187	<i>offset</i>	An unsigned number (optionally preceded by a single ' > ' character) specifying
87188		the <i>offset</i> , in bytes, of the value in the file that is to be compared against the <i>value</i>
87189		field of the line. If the file is shorter than the specified offset, the test shall fail.
87190		If the <i>offset</i> begins with the character ' > ', the test contained in the line shall not be
87191		applied to the file unless the test on the last line for which the <i>offset</i> did not begin
87192		with a ' > ' was successful. By default, the <i>offset</i> shall be interpreted as an unsigned
87193		decimal number. With a leading 0x or 0X, the <i>offset</i> shall be interpreted as a
87194		hexadecimal number; otherwise, with a leading 0, the <i>offset</i> shall be interpreted as
87195		an octal number.
87196	<i>type</i>	The type of the value in the file to be tested. The type shall consist of the type
87197		specification characters d, s, and u, specifying signed decimal, string, and
87198		unsigned decimal, respectively.
87199		The <i>type</i> string shall be interpreted as the bytes from the file starting at the
87200		specified <i>offset</i> and including the same number of bytes specified by the <i>value</i> field.
87201		If insufficient bytes remain in the file past the <i>offset</i> to match the <i>value</i> field, the test
87202		shall fail.
87203		The type specification characters d and u can be followed by an optional unsigned
87204		decimal integer that specifies the number of bytes represented by the type. The
87205		type specification characters d and u can be followed by an optional C, S, I, or L,
87206		indicating that the value is of type char , short , int , or long , respectively.
87207		The default number of bytes represented by the type specifiers d, f, and u shall
87208		correspond to their respective C-language types as follows. If the system claims
87209		conformance to the C-Language Development Utilities option, those specifiers
87210		shall correspond to the default sizes used in the <i>c99</i> utility. Otherwise, the default
87211		sizes shall be implementation-defined.
87212		For the type specifier characters d and u, the default number of bytes shall
87213		correspond to the size of a basic integer type of the implementation. For these
87214		specifier characters, the implementation shall support values of the optional
87215		number of bytes to be converted corresponding to the number of bytes in the C-
87216		language types char , short , int , or long . These numbers can also be specified by an
87217		application as the characters C, S, I, and L, respectively. The byte order used when
87218		interpreting numeric values is implementation-defined, but shall correspond to the
87219		order in which a constant of the corresponding type is stored in memory on the
87220		system.
87221		All type specifiers, except for s, can be followed by a mask specifier of the form
87222		&number. The mask value shall be AND'ed with the value of the input file before
87223		the comparison with the <i>value</i> field of the line is made. By default, the mask shall
87224		be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask
87225		shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading
87226		0, the mask shall be interpreted as an unsigned octal number.
87227		The strings byte , short , long , and string shall also be supported as type fields,
87228		being interpreted as dC, dS, dL, and s, respectively.
87229	<i>value</i>	The <i>value</i> to be compared with the value from the file.
87230		If the specifier from the type field is s or string , then interpret the value as a string.
87231		Otherwise, interpret it as a number. If the value is a string, then the test shall
87232		succeed only when a string value exactly matches the bytes from the file.

87233

If the *value* is a string, it can contain the following sequences:

87234

\character The backslash-escape sequences as specified in XBD Table 5-1 (on page 108) ('\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'). In addition, the escape sequence '\ ' (the <backslash> character followed by a <space> character) shall be recognized to represent a <space> character. The results of using any other character, other than an octal digit, following the backslash are unspecified.

87241

\octal Octal sequences that can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three octal-digit characters (01234567).

87242

87243

87244

87245

87246

87247

87248

By default, any value that is not a string shall be interpreted as a signed decimal number. Any such value, with a leading 0x or 0X, shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading zero, the value shall be interpreted as an unsigned octal number.

87249

87250

87251

If the value is not a string, it can be preceded by a character indicating the comparison to be performed. Permissible characters and the comparisons they specify are as follows:

87252

= The test shall succeed if the value from the file equals the *value* field.

87253

< The test shall succeed if the value from the file is less than the *value* field.

87254

> The test shall succeed if the value from the file is greater than the *value* field.

87255

87256

& The test shall succeed if all of the set bits in the *value* field are set in the value from the file.

87257

87258

^ The test shall succeed if at least one of the set bits in the *value* field is not set in the value from the file.

87259

87260

x The test shall succeed if the file is large enough to contain a value of the type specified starting at the offset specified.

87261

message

The *message* to be printed if the test succeeds. The *message* shall be interpreted using the notation for the *printf* formatting specification; see *printf*. If the *value* field was a string, then the value from the file shall be the argument for the *printf* formatting specification; otherwise, the value from the file shall be the argument.

87262

87263

87264

EXIT STATUS

87265

The following exit values shall be returned:

87266

0 Successful completion.

87267

>0 An error occurred.

87268

CONSEQUENCES OF ERRORS

87269

Default.

87270

87271
87272
87273
87274

87275
87276
87277
87278
87279

87280
87281

87282
87283

87284
87285
87286

87287
87288
87289
87290
87291
87292
87293

87294
87295
87296
87297
87298

87299
87300
87301
87302

87303
87304
87305
87306
87307
87308
87309
87310
87311

87312
87313

87314

87315
87316

87317
87318

APPLICATION USAGE

The *file* utility can only be required to guess at many of the file types because only exhaustive testing can determine some types with certainty. For example, binary data on some implementations might match the initial segment of an executable or a *tar* archive.

Note that the table indicates that the output contains the stated string. Systems may add text before or after the string. For executables, as an example, the machine architecture and various facts about how the file was link-edited may be included. Note also that on systems that recognize shell script files starting with "#!" as executable files, these may be identified as executable binary files rather than as shell scripts.

EXAMPLES

Determine whether an argument is a binary executable file:

```
file -- "$1" | grep -q ':. *executable' &&
  printf "%s is executable.\n" "$1"
```

RATIONALE

The `-f` option was omitted because the same effect can (and should) be obtained using the *xargs* utility.

Historical versions of the *file* utility attempt to identify the following types of files: symbolic link, directory, character special, block special, socket, *tar* archive, *cpio* archive, SCCS archive, archive library, empty, *compress* output, *pack* output, binary data, C source, FORTRAN source, assembler source, *nroff*/*troff*/*eqn*/*tbl* source *troff* output, shell script, C shell script, English text, ASCII text, various executables, APL workspace, compiled terminfo entries, and CURSES screen images. Only those types that are reasonably well specified in POSIX or are directly related to POSIX utilities are listed in the table.

Historical systems have used a "magic file" named `/etc/magic` to help identify file types. Because it is generally useful for users and scripts to be able to identify special file types, the `-m` flag and a portable format for user-created magic files has been specified. No requirement is made that an implementation of *file* use this method of identifying files, only that users be permitted to add their own classifying tests.

In addition, three options have been added to historical practice. The `-d` flag has been added to permit users to cause their tests to follow any default system tests. The `-i` flag has been added to permit users to test portably for regular files in shell scripts. The `-M` flag has been added to permit users to ignore any default system tests.

The POSIX.1-200x description of default system tests and the interaction between the `-d`, `-M`, and `-m` options did not clearly indicate that there were two types of "default system tests". The "position-sensitive tests" determine file types by looking for certain string or binary values at specific offsets in the file being examined. These position-sensitive tests were implemented in historical systems using the magic file described above. Some of these tests are now built into the *file* utility itself on some implementations so the output can provide more detail than can be provided by magic files. For example, a magic file can easily identify a **core** file on most implementations, but cannot name the program file that dropped the core. A magic file could produce output such as:

```
/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1
```

but by building the test into the *file* utility, you could get output such as:

```
/home/dwc/core: ELF 32-bit MSB core file SPARC Version 1, from 'testprog'
```

These extended built-in tests are still to be treated as position-sensitive default system tests even if they are not listed in `/etc/magic` or any other magic file.

The context-sensitive default system tests were always built into the *file* utility. These tests looked for language constructs in text files trying to identify shell scripts, C, FORTRAN, and

87319 other computer language source files, and even plain text files. With the addition of the `-m` and
 87320 `-M` options the distinction between position-sensitive and context-sensitive default system tests
 87321 became important because the order of testing is important. The context-sensitive system default
 87322 tests should never be applied before any position-sensitive tests even if the `-d` option is specified
 87323 before a `-m` option or `-M` option due to the high probability that the context-sensitive system
 87324 default tests will incorrectly identify arbitrary text files as text files before position-sensitive tests
 87325 specified by the `-m` or `-M` option would be applied to give a more accurate identification.

87326 Leaving the meaning of `-M` – and `-m` – unspecified allows an existing prototype of these
 87327 options to continue to work in a backwards-compatible manner. (In that implementation, `-M` –
 87328 was roughly equivalent to `-d` in POSIX.1-200x.)

87329 The historical `-c` option was omitted as not particularly useful to users or portable shell scripts.
 87330 In addition, a reasonable implementation of the *file* utility would report any errors found each
 87331 time the magic file is read.

87332 The historical format of the magic file was the same as that specified by the Rationale in the
 87333 ISO POSIX-2:1993 standard for the *offset*, *value*, and *message* fields; however, it used less precise
 87334 type fields than the format specified by the current normative text. The new type field values are
 87335 a superset of the historical ones.

87336 The following is an example magic file:

```
87337 0 short      070707      cpio archive
87338 0 short      0143561    Byte-swapped cpio archive
87339 0 string     070707      ASCII cpio archive
87340 0 long       0177555    Very old archive
87341 0 short      0177545    Old archive
87342 0 short      017437     Old packed data
87343 0 string     \037\036   Packed data
87344 0 string     \377\037   Compacted data
87345 0 string     \037\235   Compressed data
87346 >2 byte&0x80 >0      Block compressed
87347 >2 byte&0x1f x        %d bits
87348 0 string     \032\001   Compiled Terminfo Entry
87349 0 short      0433       Curses screen image
87350 0 short      0434       Curses screen image
87351 0 string     <ar>       System V Release 1 archive
87352 0 string     !<arch>\n__SYMDEF Archive random library
87353 0 string     !<arch>    Archive
87354 0 string     ARF_BEGARF PHIGS clear text archive
87355 0 long       0x137A2950 Scalable OpenFont binary
87356 0 long       0x137A2951 Encrypted scalable OpenFont binary
```

87357 The use of a basic integer data type is intended to allow the implementation to choose a word
 87358 size commonly used by applications on that architecture.

87359 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 87360 but this has been modified in this version.

87361 FUTURE DIRECTIONS

87362 None.

87363 SEE ALSO

87364 *ar*, *ls*, *pax*

87365 XBD Table 5-1 (on page 108), Chapter 8 (on page 159), Section 12.2 (on page 201)

+

87366

CHANGE HISTORY

87367

First released in Issue 4.

87368

Issue 6

87369

This utility is marked as part of the User Portability Utilities option.

87370

Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft standard.

87371

87372

IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

87373

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/25 is applied, making major changes to address ambiguities raised in defect reports.

87374

87375

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/26 is applied, making it clear in the OPTIONS section that the **-m**, **-d**, and **-M** options do not comply with Guideline 11 of the Utility Syntax Guidelines.

87376

87377

87378

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/10 is applied, clarifying the specification characters.

87379

87380

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/11 is applied, allowing application developers to create portable magic files that can match characters in strings, and allowing common extensions found in existing implementations.

87381

87382

87383

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/12 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

87384

87385

Issue 7

87386

SD5-XCU-ERN-4 is applied, adding further entries in the Notes column in [Table 4-8](#).

87387

The *file* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

87388

87389

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87390

The EXAMPLES section is revised to correct an error with the pathname "\$1".

+

87391 **NAME**
87392 find — find files

87393 **SYNOPSIS**
87394 find [-H|-L] *path...* [*operand_expression...*]

87395 **DESCRIPTION**
87396 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,
87397 evaluating a Boolean expression composed of the primaries described in the OPERANDS section
87398 for each file encountered.

87399 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail
87400 due to path length limitations (unless a *path* operand specified by the application exceeds
87401 {PATH_MAX} requirements).

87402 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an
87403 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a
87404 diagnostic message to standard error and shall either recover its position in the hierarchy or
87405 terminate.

87406 **OPTIONS**
87407 The *find* utility shall conform to XBD [Section 12.2](#) (on page 201).

87408 The following options shall be supported by the implementation:

87409 **-H** Cause the file information and file type evaluated for each symbolic link
87410 encountered on the command line to be those of the file referenced by the link, and
87411 not the link itself. If the referenced file does not exist, the file information and type
87412 shall be for the link itself. File information for all symbolic links not on the
87413 command line shall be that of the link itself.

87414 **-L** Cause the file information and file type evaluated for each symbolic link to be
87415 those of the file referenced by the link, and not the link itself. If the referenced file
87416 does not exist, the file information and type shall be for the link itself.

87417 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
87418 an error. The last option specified shall determine the behavior of the utility.

87419 **OPERANDS**
87420 The following operands shall be supported:

87421 The *path* operand is a pathname of a starting point in the directory hierarchy.

87422 The first operand that starts with a '-', or is a '!' or a '(', and all subsequent arguments shall
87423 be interpreted as an *expression* made up of the following primaries and operators. In the
87424 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal
87425 integer optionally preceded by a plus ('+') or minus ('-') sign, as follows:

87426 **+n** More than *n*.

87427 **n** Exactly *n*.

87428 **-n** Less than *n*.

87429 The following primaries shall be supported:

87430 **-name** *pattern*

87431 The primary shall evaluate as true if the basename of the current pathname
87432 matches *pattern* using the pattern matching notation described in [Section 2.13](#) (on
87433 page 2278). The additional rules in [Section 2.13.3](#) (on page 2279) do not apply as +

- 87434 this is a matching operation, not an expansion.
- 87435 **-path** *pattern*
- 87436 The primary shall evaluate as true if the current pathname matches *pattern* using +
 87437 the pattern matching notation described in Section 2.13 (on page 2278). The +
 87438 additional rules in Section 2.13.3 (on page 2279) do not apply as this is a matching +
 87439 operation, not an expansion.
- 87440 **-nouser** The primary shall evaluate as true if the file belongs to a user ID for which the
 87441 *getpwuid()* function defined in the System Interfaces volume of POSIX.1-200x (or
 87442 equivalent) returns NULL.
- 87443 **-nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the
 87444 *getgrgid()* function defined in the System Interfaces volume of POSIX.1-200x (or
 87445 equivalent) returns NULL.
- 87446 **-xdev** The primary shall always evaluate as true; it shall cause *find* not to continue
 87447 descending past directories that have a different device ID (*st_dev*, see the *stat()*
 87448 function defined in the System Interfaces volume of POSIX.1-200x). If any **-xdev**
 87449 primary is specified, it shall apply to the entire expression even if the **-xdev**
 87450 primary would not normally be evaluated.
- 87451 **-prune** The primary shall always evaluate as true; it shall cause *find* not to descend the
 87452 current pathname if it is a directory. If the **-depth** primary is specified, the **-prune**
 87453 primary shall have no effect.
- 87454 **-perm** [-]*mode*
- 87455 The *mode* argument is used to represent file mode bits. It shall be identical in
 87456 format to the *symbolic_mode* operand described in *chmod*, and shall be interpreted
 87457 as follows. To start, a template shall be assumed with all file mode bits cleared. An
 87458 *op* symbol of '+' shall set the appropriate mode bits in the template; '-' shall
 87459 clear the appropriate bits; '=' shall set the appropriate mode bits, without regard
 87460 to the contents of the file mode creation mask of the process. The *op* symbol of '-'
 87461 cannot be the first character of *mode*; this avoids ambiguity with the optional
 87462 leading hyphen. Since the initial mode is all bits off, there are not any symbolic
 87463 modes that need to use '-' as the first character.
- 87464 If the hyphen is omitted, the primary shall evaluate as true when the file
 87465 permission bits exactly match the value of the resulting template.
- 87466 Otherwise, if *mode* is prefixed by a hyphen, the primary shall evaluate as true if at
 87467 least all the bits in the resulting template are set in the file permission bits.
- 87468 **-perm** [-]*onum*
- 87469 If the hyphen is omitted, the primary shall evaluate as true when the file mode bits
 87470 exactly match the value of the octal number *onum* (see the description of the octal
 87471 *mode* in *chmod*). Otherwise, if *onum* is prefixed by a hyphen, the primary shall
 87472 evaluate as true if at least all of the bits specified in *onum* are set. In both cases, the
 87473 behavior is unspecified when *onum* exceeds 07777.
- 87474 **-type** *c*
- 87475 The primary shall evaluate as true if the type of the file is *c*, where *c* is 'b', 'c',
 87476 'd', 'l', 'p', 'f', or 's' for block special file, character special file, directory,
 symbolic link, FIFO, regular file, or socket, respectively.
- 87477 **-links** *n*
- 87478 The primary shall evaluate as true if the file has *n* links.
- 87479 **-user** *uname*
- 87480 The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is
 a decimal integer and the *getpwnam()* (or equivalent) function does not return a
 valid user name, *uname* shall be interpreted as a user ID.

- 87481 **-group** *gname*
- 87482 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
- 87483 is a decimal integer and the *getgrnam()* (or equivalent) function does not return a
- 87484 valid group name, *gname* shall be interpreted as a group ID.
- 87485 **-size** *n*[*c*]
- 87486 The primary shall evaluate as true if the file size in bytes, divided by 512 and
- 87487 rounded up to the next integer, is *n*. If *n* is followed by the character '*c*', the size
- shall be in bytes.
- 87488 **-atime** *n*
- 87489 The primary shall evaluate as true if the file access time subtracted from the
- initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 87490 **-ctime** *n*
- 87491 The primary shall evaluate as true if the time of last change of file status
- 87492 information subtracted from the initialization time, divided by 86 400 (with any
- remainder discarded), is *n*.
- 87493 **-mtime** *n*
- 87494 The primary shall evaluate as true if the file modification time subtracted from the
- initialization time, divided by 86 400 (with any remainder discarded), is *n*.
- 87495 **-exec** *utility_name* [*argument ...*];
- 87496 **-exec** *utility_name* [*argument ...*] {} +
- 87497 The end of the primary expression shall be punctuated by a semicolon or by a plus
- 87498 sign. Only a plus sign that follows an argument containing the two characters
- 87499 "{}" shall punctuate the end of the primary expression. Other uses of the plus
- 87500 sign shall not be treated as special.
- 87501 If the primary expression is punctuated by a semicolon, the utility *utility_name*
- 87502 shall be invoked once for each pathname and the primary shall evaluate as true if
- 87503 the utility returns a zero value as exit status. A *utility_name* or *argument* containing
- 87504 only the two characters "{}" shall be replaced by the current pathname.
- 87505 If the primary expression is punctuated by a plus sign, the primary shall always
- 87506 evaluate as true, and the pathnames for which the primary is evaluated shall be
- 87507 aggregated into sets. The utility *utility_name* shall be invoked once for each set of
- 87508 aggregated pathnames. Each invocation shall begin after the last pathname in the
- 87509 set is aggregated, and shall be completed before the *find* utility exits and before the
- 87510 first pathname in the next set (if any) is aggregated for this primary, but it is
- 87511 otherwise unspecified whether the invocation occurs before, during, or after the
- 87512 evaluations of other primaries. If any invocation returns a non-zero value as exit
- 87513 status, the *find* utility shall return a non-zero exit status. An argument containing
- 87514 only the two characters "{}" shall be replaced by the set of aggregated
- 87515 pathnames, with each pathname passed as a separate argument to the invoked
- 87516 utility in the same order that it was aggregated. The size of any set of two or more
- 87517 pathnames shall be limited such that execution of the utility does not cause the
- 87518 system's {ARG_MAX} limit to be exceeded. If more than one argument containing
- 87519 only the two characters "{}" is present, the behavior is unspecified.
- 87520 If a *utility_name* or *argument* string contains the two characters "{}", but not just
- 87521 the two characters "{}", it is implementation-defined whether *find* replaces those
- 87522 two characters or uses the string without change. The current directory for the
- 87523 invocation of *utility_name* shall be the same as the current directory when the *find*
- 87524 utility was started. If the *utility_name* names any of the special built-in utilities (see
- 87525 Section 2.14, on page 2280), the results are undefined.
- 87526 **-ok** *utility_name* [*argument ...*];
- 87527 The **-ok** primary shall be equivalent to **-exec**, except that the use of a plus sign to
- 87528 punctuate the end of the primary expression need not be supported, and *find* shall
- 87529 request affirmation of the invocation of *utility_name* using the current file as an
- 87530 argument by writing to standard error as described in the STDERR section. If the

- 87531 response on standard input is affirmative, the utility shall be invoked. Otherwise,
87532 the command shall not be invoked and the value of the **-ok** operand shall be false.
- 87533 **-print** The primary shall always evaluate as true; it shall cause the current pathname to
87534 be written to standard output.
- 87535 **-newer file** The primary shall evaluate as true if the modification time of the current file is
87536 more recent than the modification time of the file named by the pathname *file*.
- 87537 **-depth** The primary shall always evaluate as true; it shall cause descent of the directory
87538 hierarchy to be done so that all entries in a directory are acted on before the
87539 directory itself. If a **-depth** primary is not specified, all entries in a directory shall
87540 be acted on after the directory itself. If any **-depth** primary is specified, it shall
87541 apply to the entire expression even if the **-depth** primary would not normally be
87542 evaluated.

87543 The primaries can be combined using the following operators (in order of decreasing
87544 precedence):

87545 (*expression*) True if *expression* is true.

87546 **!expression** Negation of a primary; the unary NOT operator.

87547 *expression* [**-a**] *expression*

87548 Conjunction of primaries; the AND operator is implied by the juxtaposition of two
87549 primaries or made explicit by the optional **-a** operator. The second expression shall
87550 not be evaluated if the first expression is false.

87551 *expression* **-o** *expression*

87552 Alternation of primaries; the OR operator. The second expression shall not be
87553 evaluated if the first expression is true.

87554 If no *expression* is present, **-print** shall be used as the expression. Otherwise, if the given
87555 expression does not contain any of the primaries **-exec**, **-ok**, or **-print**, the given expression
87556 shall be effectively replaced by:

87557 (*given_expression*) **-print**

87558 The **-user**, **-group**, and **-newer** primaries each shall evaluate their respective arguments only
87559 once.

87560 STDIN

87561 If the **-ok** primary is used, the response shall be read from the standard input. An entire line
87562 shall be read as the response. Otherwise, the standard input shall not be used.

87563 INPUT FILES

87564 None.

87565 ENVIRONMENT VARIABLES

87566 The following environment variables shall affect the execution of *find*:

87567 **LANG** Provide a default value for the internationalization variables that are unset or null. |
87568 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
87569 variables used to determine the values of locale categories.)

87570 **LC_ALL** If set to a non-empty string value, override the values of all the other
87571 internationalization variables.

87572 **LC_COLLATE**

87573 Determine the locale for the behavior of ranges, equivalence classes, and multi-
87574 character collating elements used in the pattern matching notation for the **-n**
87575 option and in the extended regular expression defined for the **yesexpr** locale
87576 keyword in the **LC_MESSAGES** category.

87577 *LC_CTYPE* This variable determines the locale for the interpretation of sequences of bytes of
 87578 text data as characters (for example, single-byte as opposed to multi-byte
 87579 characters in arguments), the behavior of character classes within the pattern
 87580 matching notation used for the **-n** option, and the behavior of character classes
 87581 within regular expressions used in the extended regular expression defined for the
 87582 **yesexpr** locale keyword in the *LC_MESSAGES* category.

87583 *LC_MESSAGES*
 87584 Determine the locale for the processing of affirmative responses that should be
 87585 used to affect the format and contents of diagnostic messages written to standard
 87586 error.

87587 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

87588 *PATH* Determine the location of the *utility_name* for the **-exec** and **-ok** primaries, as
 87589 described in XBD [Chapter 8](#) (on page 159).

87590 ASYNCHRONOUS EVENTS

87591 Default.

87592 STDOUT

87593 The **-print** primary shall cause the current pathnames to be written to standard output. The
 87594 format shall be:

87595 "%s\n", <path>

87596 STDERR

87597 The **-ok** primary shall write a prompt to standard error containing at least the *utility_name* to be
 87598 invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall
 87599 be ' ? '. The exact format used is unspecified.

87600 Otherwise, the standard error shall be used only for diagnostic messages.

87601 OUTPUT FILES

87602 None.

87603 EXTENDED DESCRIPTION

87604 None.

87605 EXIT STATUS

87606 The following exit values shall be returned:

87607 0 All *path* operands were traversed successfully.

87608 >0 An error occurred.

87609 CONSEQUENCES OF ERRORS

87610 Default.

87611 APPLICATION USAGE

87612 When used in operands, pattern matching notation, semicolons, opening parentheses, and
 87613 closing parentheses are special to the shell and must be quoted (see [Section 2.2](#), on page 2246).

87614 The bit that is traditionally used for sticky (historically 01000) is specified in the **-perm** primary
 87615 using the octal number argument form. Since this bit is not defined by this volume of
 87616 POSIX.1-200x, applications must not assume that it actually refers to the traditional sticky bit.

87617 EXAMPLES

87618 1. The following commands are equivalent:

87619 find .

87620 find . -print

- 87621 They both write out the entire directory hierarchy from the current directory.
- 87622 2. The following command:
- 87623 `find / \(-name tmp -o -name '*.xx' \) -atime +7 -exec rm {} \;`
- 87624 removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or
- 87625 more 24-hour periods.
- 87626 3. The following command:
- 87627 `find . -perm -o+w,+s`
- 87628 prints (**-print** is assumed) the names of all files in or below the current directory, with all
- 87629 of the file permission bits **S_ISUID**, **S_ISGID**, and **S_IWOTH** set.
- 87630 4. The following command:
- 87631 `find . -name SCCS -prune -o -print`
- 87632 recursively prints pathnames of all files in the current directory and below, but skips
- 87633 directories named **SCCS** and files in them.
- 87634 5. The following command:
- 87635 `find . -print -name SCCS -prune`
- 87636 behaves as in the previous example, but prints the names of the **SCCS** directories.
- 87637 6. The following command is roughly equivalent to the **-nt** extension to *test*:
- 87638 `if [-n "$(find file1 -prune -newer file2)"]; then`
- 87639 `printf %s\\n "file1 is newer than file2"`
- 87640 `fi`
- 87641 7. The descriptions of **-atime**, **-ctime**, and **-mtime** use the terminology *n* “86 400 second
- 87642 periods (days)”. For example, a file accessed at 23:59 is selected by:
- 87643 `find . -atime -1 -print`
- 87644 at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight
- 87645 boundary between days has no effect on the 24-hour calculation.
- 87646 8. The following command:
- 87647 `find . ! -name . -prune -name '*.old' -exec \`
- 87648 `sh -c 'mv "$@" ../old/' sh {} +`
- 87649 performs the same task as:
- 87650 `mv /*.old ../old`
- 87651 while avoiding an “Argument list too long” error if there are a large number of files
- 87652 ending with **.old** (and avoiding “No such file or directory” errors if no files match **/*.old**
- 87653 or **/*.old**).
- 87654 The alternative:
- 87655 `find . ! -name . -prune -name '*.old' -exec mv {} ../old/ \;`
- 87656 is less efficient if there are many files to move because it executes one *mv* command per
- 87657 file.
- 87658 9. On systems configured to mount removable media on directories under **/media**, the +
- 87659 following command searches the file hierarchy for files larger than 100 000 KB without +
- 87660 searching any mounted removable media: +
- 87661 `find / -path /media -prune -o -size +200000 -print` +

RATIONALE

87662

87663

87664

The **-a** operator was retained as an optional operator for compatibility with historical shell scripts, even though it is redundant with expression concatenation.

87665

87666

87667

87668

87669

The descriptions of the **'-'** modifier on the *mode* and *onum* arguments to the **-perm** primary agree with historical practice on BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching bits set.

87670

87671

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because:

87672

87673

- Implementations may desire more descriptive prompts than those used on historical implementations.

87674

87675

- Since the historical prompt strings do not terminate with <newline>s, there is no portable way for another program to interact with the prompts of this utility via pipes.

87676

87677

Therefore, an application using this prompting option relies on the system to provide the most suitable dialog directly with the user, based on the general guidelines specified.

87678

87679

The **-name** *file* operand was changed to use the shell pattern matching notation so that *find* is consistent with other utilities using pattern matching.

87680

87681

87682

87683

The **-size** operand refers to the size of a file, rather than the number of blocks it may occupy in the file system. The intent is that the *st_size* field defined in the System Interfaces volume of POSIX.1-200x should be used, not the *st_blocks* found in historical implementations. There are at least two reasons for this:

87684

87685

87686

1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands specified by this volume of POSIX.1-200x. (BSD uses *st_blocks* only when processing the **-ls** primary.)

87687

87688

87689

87690

2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility for the output from the **-l** option. (In both System V and BSD, *ls* uses *st_size* for the **-l** option size field and uses *st_blocks* for the *ls -s* calculations. This volume of POSIX.1-200x does not specify *ls -s*.)

87691

87692

87693

87694

87695

87696

The descriptions of **-atime**, **-ctime**, and **-mtime** were changed from the SVID description of *n* "days" to *n* being the result of the integer division of the time difference in seconds by 86 400. The description is also different in terms of the exact timeframe for the *n* case (*versus* the *+n* or *-n*), but it matches all known historical implementations. It refers to one 86 400 second period in the past, not any time from the beginning of that period to the current time. For example, **-atime 2** is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

87697

87698

87699

87700

87701

87702

Historical implementations do not modify "`{ }`" when it appears as a substring of an **-exec** or **-ok** *utility_name* or argument string. There have been numerous user requests for this extension, so this volume of POSIX.1-200x allows the desired behavior. At least one recent implementation does support this feature, but encountered several problems in managing memory allocation and dealing with multiple occurrences of "`{ }`" in a string while it was being developed, so it is not yet required behavior.

87703

87704

87705

87706

87707

Assuming the presence of **-print** was added to correct a historical pitfall that plagues novice users, it is entirely upwards-compatible from the historical System V *find* utility. In its simplest form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers agreed that adding **-print** as a default expression was the correct decision and have added the fast *find* functionality within a new utility called *locate*.

87708

87709

Historically, the **-L** option was implemented using the primary **-follow**. The **-H** and **-L** options were added for two reasons. First, they offer a finer granularity of control and consistency with

87710 other programs that walk file hierarchies. Second, the **-follow** primary always evaluated to true.
 87711 As they were historically really global variables that took effect before the traversal began, some
 87712 valid expressions had unexpected results. An example is the expression **-print -o -follow**.
 87713 Because **-print** always evaluates to true, the standard order of evaluation implies that **-follow**
 87714 would never be evaluated. This was never the case. Historical practice for the **-follow** primary,
 87715 however, is not consistent. Some implementations always follow symbolic links on the
 87716 command line whether **-follow** is specified or not. Others follow symbolic links on the
 87717 command line only if **-follow** is specified. Both behaviors are provided by the **-H** and **-L**
 87718 options, but scripts using the current **-follow** primary would be broken if the **-follow** option is
 87719 specified to work either way.

87720 Since the **-L** option resolves all symbolic links and the **-type l** primary is true for symbolic links
 87721 that still exist after symbolic links have been resolved, the command:

```
87722 find -L . -type l
```

87723 prints a list of symbolic links reachable from the current directory that do not resolve to
 87724 accessible files.

87725 A feature of SVR4's *find* utility was the **-exec** primary's **+** terminator. This allowed filenames
 87726 containing special characters (especially <newline>s) to be grouped together without the
 87727 problems that occur if such filenames are piped to *xargs*. Other implementations have added
 87728 other ways to get around this problem, notably a **-print0** primary that wrote filenames with a
 87729 null byte terminator. This was considered here, but not adopted. Using a null terminator meant
 87730 that any utility that was going to process *find*'s **-print0** output had to add a new option to parse
 87731 the null terminators it would now be reading.

87732 The "**-exec ... { } +**" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210.
 87733 It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For
 87734 example, the following command prints all files with a '-' after their name if they are regular
 87735 files, and a '+' otherwise:

```
87736 find / -type f -exec echo { } - ';' -o -exec echo { } + ';' |
```

87737 The change invalidates usage like this. Even though the previous standard stated that this usage
 87738 would work, in practice many did not support it and the standard developers felt it better to
 87739 now state that this was not allowable.

87740 FUTURE DIRECTIONS

87741 None.

87742 SEE ALSO

87743 [Section 2.2](#) (on page 2246), [Section 2.13](#) (on page 2278), [Section 2.14](#) (on page 2280), *chmod*, *pax*,
 87744 *sh*, *test*

87745 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

87746 XSH *fstatat()*, *getgrgid()*, *getpwuid()*

87747 CHANGE HISTORY

87748 First released in Issue 2.

87749 Issue 5

87750 The FUTURE DIRECTIONS section is added.

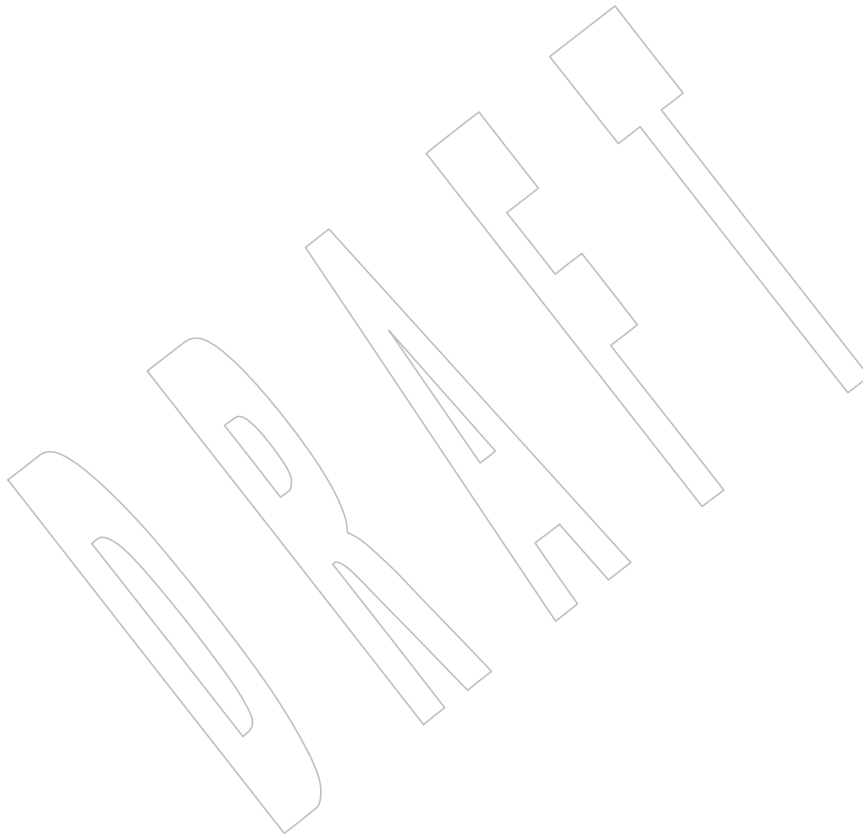
87751 Issue 6

87752 The following new requirements on POSIX implementations derive from alignment with the
 87753 Single UNIX Specification:

- 87754 • The **-perm [-]onum** primary is supported.

87755 The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of

87756	symbolic links and changes to the description of the atime , ctime , and mtime operands.	
87757	IEEE PASC Interpretation 1003.2 #210 is applied, extending the -exec operand.	
87758	IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/13 is applied, updating the RATIONALE	
87759	section to be consistent with the normative text.	
87760	Issue 7	
87761	SD5-XCU-ERN-48 is applied, clarifying the -L option in the case that the referenced file does not	
87762	exist.	
87763	SD5-XCU-ERN-89 is applied, updating the OPERANDS section.	
87764	SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.	
87765	SD5-XCU-ERN-117 is applied, clarifying the -perm operand.	
87766	SD5-XCU-ERN-122 is applied, adding a new EXAMPLE.	
87767	The description of the -name primary is revised and a new example added.	+



87768 **NAME**
 87769 fold — filter for folding lines

87770 **SYNOPSIS**
 87771 fold [-bs] [-w *width*] [*file...*]

87772 **DESCRIPTION**
 87773 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a
 87774 maximum of *width* column positions (or bytes, if the **-b** option is specified). Lines shall be
 87775 broken by the insertion of a <newline> such that each output line (referred to later in this section
 87776 as a *segment*) is the maximum width possible that does not exceed the specified number of
 87777 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior
 87778 is undefined if *width* is less than the number of columns any single character in the input would
 87779 occupy.

87780 If the <carriage-return>s, <backspace>s, or <tab>s are encountered in the input, and the **-b**
 87781 option is not specified, they shall be treated specially:

87782 <backspace> The current count of line width shall be decremented by one, although the count
 87783 never shall become negative. The *fold* utility shall not insert a <newline>
 87784 immediately before or after any <backspace>.

87785 <carriage-return>
 87786 The current count of line width shall be set to zero. The *fold* utility shall not insert a
 87787 <newline> immediately before or after any <carriage-return>.

87788 <tab> Each <tab> encountered shall advance the column position pointer to the next tab
 87789 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

87790 **OPTIONS**
 87791 The *fold* utility shall conform to XBD [Section 12.2](#) (on page 201).

87792 The following options shall be supported:

87793 **-b** Count *width* in bytes rather than column positions.

87794 **-s** If a segment of a line contains a <blank> within the first *width* column positions (or
 87795 bytes), break the line after the last such <blank> meeting the width constraints. If
 87796 there is no <blank> meeting the requirements, the **-s** option shall have no effect for
 87797 that output segment of the input line.

87798 **-w *width*** Specify the maximum line length, in column positions (or bytes if **-b** is specified).
 87799 The results are unspecified if *width* is not a positive decimal number. The default
 87800 value shall be 80.

87801 **OPERANDS**
 87802 The following operand shall be supported:

87803 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard
 87804 input shall be used.

87805 **STDIN**
 87806 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
 87807 section.

87808 **INPUT FILES**

87809 If the **-b** option is specified, the input files shall be text files except that the lines are not limited
 87810 to {LINE_MAX} bytes in length. If the **-b** option is not specified, the input files shall be text files.

87811 **ENVIRONMENT VARIABLES**

87812 The following environment variables shall affect the execution of *fold*:

87813 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 87814 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 87815 variables used to determine the values of locale categories.)

87816 **LC_ALL** If set to a non-empty string value, override the values of all the other
 87817 internationalization variables.

87818 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 87819 characters (for example, single-byte as opposed to multi-byte characters in
 87820 arguments and input files), and for the determination of the width in column
 87821 positions each character would occupy on a constant-width font output device.

87822 **LC_MESSAGES**
 87823 Determine the locale that should be used to affect the format and contents of
 87824 diagnostic messages written to standard error.

87825 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

87826 **ASYNCHRONOUS EVENTS**

87827 Default.

87828 **STDOUT**

87829 The standard output shall be a file containing a sequence of characters whose order shall be
 87830 preserved from the input files, possibly with inserted <newline>s.

87831 **STDERR**

87832 The standard error shall be used only for diagnostic messages.

87833 **OUTPUT FILES**

87834 None.

87835 **EXTENDED DESCRIPTION**

87836 None.

87837 **EXIT STATUS**

87838 The following exit values shall be returned:

87839 0 All input files were processed successfully.

87840 >0 An error occurred.

87841 **CONSEQUENCES OF ERRORS**

87842 Default.

87843 **APPLICATION USAGE**

87844 The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths.
 87845 The *cut* utility should be used when the number of lines (or records) needs to remain constant.
 87846 The *fold* utility should be used when the contents of long lines need to be kept contiguous.

87847 The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines
 87848 wider than the printer is able to print (usually 80 or 132 column positions).

87849 **EXAMPLES**

87850 An example invocation that submits a file of possibly long lines to the printer (under the
87851 assumption that the user knows the line width of the printer to be assigned by *lp*):

87852 `fold -w 132 bigfile | lp`

87853 **RATIONALE**

87854 Although terminal input in canonical processing mode requires the erase character (frequently
87855 set to <backspace>) to erase the previous character (not byte or column position), terminal
87856 output is not buffered and is extremely difficult, if not impossible, to parse correctly; the
87857 interpretation depends entirely on the physical device that actually displays/prints/stores the
87858 output. In all known internationalized implementations, the utilities producing output for
87859 mixed column-width output assume that a <backspace> backs up one column position and
87860 outputs enough <backspace>s to return to the start of the character when <backspace> is used to
87861 provide local line motions to support underlining and boldening operations. Since *fold*
87862 without the **-b** option is dealing with these same constraints, <backspace> is always treated as
87863 backing up one column position rather than backing up one character.

87864 Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column
87865 position when written out. This is no longer always true. Since the most common usage of *fold* is
87866 believed to be folding long lines for output to limited-length output devices, this capability was
87867 preserved as the default case. The **-b** option was added so that applications could *fold* files with
87868 arbitrary length lines into text files that could then be processed by the standard utilities. Note
87869 that although the width for the **-b** option is in bytes, a line is never split in the middle of a
87870 character. (It is unspecified what happens if a width is specified that is too small to hold a single
87871 character found in the input followed by a <newline>.)

87872 The tab stops are hardcoded to be every eighth column to meet historical practice. No new
87873 method of specifying other tab stops was invented.

87874 **FUTURE DIRECTIONS**

87875 None.

87876 **SEE ALSO**

87877 *cut*

87878 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

87879 **CHANGE HISTORY**

87880 First released in Issue 4.

87881 **Issue 6**

87882 The normative text is reworded to avoid use of the term “must” for application requirements.

87883 **Issue 7**

87884 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

87885

NAME

87886

fort77 — FORTRAN compiler (**FORTRAN**)

87887

SYNOPSIS

87888

FD `fort77 [-c] [-g] [-L directory]... [-O optlevel] [-o outfile] [-s]`

87889

`[-w] operand...`

87890

DESCRIPTION

87891

The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually consists of a compiler and link editor. The files referenced by *operands* are compiled and linked to produce an executable file. It is unspecified whether the linking occurs entirely within the operation of *fort77*; some implementations may produce objects that are not fully resolved until the file is executed.

87892

87893

87894

87895

87896

87897

If the `-c` option is present, for all pathname operands of the form *file.f*, the files:

87898

```
$(basename pathname.f) .o
```

87899

shall be created or overwritten as the result of successful compilation. If the `-c` option is not specified, it is unspecified whether such `.o` files are created or deleted for the *file.f* operands.

87900

87901

If there are no options that prevent link editing (such as `-c`) and all operands compile and link without error, the resulting executable file shall be written into the file named by the `-o` option (if present) or to the file **a.out**. The executable file shall be created as specified in the System Interfaces volume of POSIX.1-200x, except that the file permissions shall be set to:

87902

87903

87904

87905

```
S_IRWXO | S_IRWXG | S_IRWXU
```

87906

and that the bits specified by the *umask* of the process shall be cleared.

87907

OPTIONS

87908

The *fort77* utility shall conform to XBD [Section 12.2](#) (on page 201), except that:

87909

- The `-l` *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.

87910

87911

- The order of specifying the multiple `-L` options is significant.

87912

- Conforming applications shall specify each option separately; that is, grouping option letters (for example, `-cg`) need not be recognized by all implementations.

87913

87914

The following options shall be supported:

87915

`-c` Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.

87916

87917

`-g` Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

87918

87919

87920

`-s` Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the *exec* family of functions defined in the System Interfaces volume of POSIX.1-200x has been removed (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.

87921

87922

87923

87924

`-o outfile` Use the pathname *outfile*, instead of the default **a.out**, for the executable file produced. If the `-o` option is present with `-c`, the result is unspecified.

87925

- 87926 **-L *directory*** Change the algorithm of searching for the libraries named in **-I** operands to look in
87927 the directory named by the *directory* pathname before looking in the usual places.
87928 Directories named in **-L** options shall be searched in the specified order. At least
87929 ten instances of this option shall be supported in a single *fort77* command
87930 invocation. If a directory specified by a **-L** option contains a file named **libf.a**, the
87931 results are unspecified.
- 87932 **-O *optlevel*** Specify the level of code optimization. If the *optlevel* option-argument is the digit
87933 '0', all special code optimizations shall be disabled. If it is the digit '1', the
87934 nature of the optimization is unspecified. If the **-O** option is omitted, the nature of
87935 the system's default optimization is unspecified. It is unspecified whether code
87936 generated in the presence of the **-O 0** option is the same as that generated when
87937 **-O** is omitted. Other *optlevel* values may be supported.
- 87938 **-w** Suppress warnings.
- 87939 Multiple instances of **-L** options can be specified.

OPERANDS

87941 An *operand* is either in the form of a pathname or the form **-I *library***. At least one operand of the
87942 pathname form shall be specified. The following operands shall be supported:

- 87943 ***file.f*** The pathname of a FORTRAN source file to be compiled and optionally passed to
87944 the link editor. The filename operand shall be of this form if the **-c** option is used.
- 87945 ***file.a*** A library of object files typically produced by *ar*, and passed directly to the link
87946 editor. Implementations may recognize implementation-defined suffixes other
87947 than **.a** as denoting object file libraries.
- 87948 ***file.o*** An object file produced by *fort77 -c* and passed directly to the link editor.
87949 Implementations may recognize implementation-defined suffixes other than **.o** as
87950 denoting object files.

87951 The processing of other files is implementation-defined.

- 87952 **-I *library*** (The letter ell.) Search the library named:
87953 **lib*library*.a**
- 87954 A library is searched when its name is encountered, so the placement of a **-I**
87955 operand is significant. Several standard libraries can be specified in this manner, as
87956 described in the EXTENDED DESCRIPTION section. Implementations may
87957 recognize implementation-defined suffixes other than **.a** as denoting libraries.

STDIN

87958 Not used.

INPUT FILES

87961 The input file shall be one of the following: a text file containing FORTRAN source code; an
87962 object file in the format produced by *fort77 -c*; or a library of object files, in the format produced
87963 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
87964 that produce files in these formats. Additional input files are implementation-defined.

87965 A **<tab>** encountered within the first six characters on a line of source code shall cause the
87966 compiler to interpret the following character as if it were the seventh character on the line (that
87967 is, in column 7).

ENVIRONMENT VARIABLES

87968 The following environment variables shall affect the execution of *fort77*:

87970		<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)
87971			
87972			
87973		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
87974			
87975		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).
87976			
87977			
87978		<i>LC_MESSAGES</i>	
87979			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
87980			
87981	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
87982		<i>TMPDIR</i>	Determine the pathname that should override the default directory for temporary files, if any.
87983			

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages. If more than one *file* operand ending in *.f* (or possibly other unspecified suffixes) is given, for each such file:

"%s:\n", <file>

may be written to allow identification of the diagnostic message with the appropriate input file.

This utility may produce warning messages about certain conditions that do not warrant returning an error (non-zero) exit value.

OUTPUT FILES

Object files, listing files, and executable files shall be produced in unspecified formats.

EXTENDED DESCRIPTION**Standard Libraries**

The *fort77* utility shall recognize the following *-l* operand for the standard library:

-l f This library contains all functions referenced in the ANSI X3.9-1978 standard. This operand shall not be required to be present to cause a search of this library.

In the absence of options that inhibit invocation of the link editor, such as *-c*, the *fort77* utility shall cause the equivalent of a *-l f* operand to be passed to the link editor as the last *-l* operand, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the library *libf.a* exists as a regular file. The implementation may accept as *-l* operands names of objects that do not exist as regular files.

88007

External Symbols

88008

The FORTRAN compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; case folding is permitted. The action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

88009

88010

88011

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols total. A diagnostic message is written to standard output if the implementation-defined limit is exceeded; other actions are unspecified.

88012

88013

88014

EXIT STATUS

88015

The following exit values shall be returned:

88016

0 Successful compilation or link edit.

88017

>0 An error occurred.

88018

CONSEQUENCES OF ERRORS

88019

When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and continue to compile other source code operands. It shall return a non-zero exit status, but it is implementation-defined whether an object module is created. If the link edit is unsuccessful, a diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero status.

88020

88021

88022

88023

APPLICATION USAGE

88024

None.

88025

EXAMPLES

88026

The following usage example compiles **xyz.f** and creates the executable file **foo**:

88027

```
fort77 -o foo xyz.f
```

88028

The following example compiles **xyz.f** and creates the object file **xyz.o**:

88029

```
fort77 -c xyz.f
```

88030

The following example compiles **xyz.f** and creates the executable file **a.out**:

88031

```
fort77 xyz.f
```

88032

The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

88033

```
fort77 xyz.f b.o
```

88034

RATIONALE

88035

The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The name *f77* was not chosen to avoid problems with historical implementations. The ANSI X3.9-1978 standard was selected as a normative reference because the ISO/IEC version of FORTRAN-77 has been superseded by the ISO/IEC 1539:1990 standard (Fortran-90).

88036

88037

88038

88039

The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not included in this volume of POSIX.1-200x—even though they are commonly implemented—since there is no requirement that the FORTRAN compiler use the C preprocessor.

88040

88041

88042

The **-onetrip** option was not included in this volume of POSIX.1-200x, even though many historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism that should not be perpetuated.

88043

88044

88045

Some implementations produce compilation listings. This aspect of FORTRAN has been left unspecified because there was controversy concerning the various methods proposed for implementing it: a **-V** option overlapped with historical vendor practice and a naming convention of creating files with **.I** suffixes collided with historical *lex* file naming practice.

88046

88047

88048

88049

There is no **-I** option in this version of this volume of POSIX.1-200x to specify a directory for file inclusion. An **INCLUDE** directive has been a part of the Fortran-90 discussions, but an interface

88050

88051 supporting that standard is not in the current scope.

88052 It is noted that many FORTRAN compilers produce an object module even when compilation
88053 errors occur; during a subsequent compilation, the compiler may patch the object module rather
88054 than recompiling all the code. Consequently, it is left to the implementor whether or not an
88055 object file is created.

88056 A reference to MIL-STD-1753 was removed from an early proposal in response to a request from
88057 the POSIX FORTRAN-binding standard developers. It was not the intention of the standard
88058 developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not
88059 specify the military standard or any special preprocessing requirements. Furthermore, use of
88060 that document would have been inappropriate for an international standard.

88061 The specification of optimization has been subject to changes through early proposals. At one
88062 time, `-O` and `-N` were Booleans: optimize and do not optimize (with an unspecified default).
88063 Some historical practice led this to be changed to:

88064 `-O 0` No optimization.

88065 `-O 1` Some level of optimization.

88066 `-O n` Other, unspecified levels of optimization.

88067 It is not always clear whether “good code generation” is the same thing as optimization. Simple
88068 optimizations of local actions do not usually affect the semantics of a program. The `-O 0` option
88069 has been included to accommodate the very particular nature of scientific calculations in a
88070 highly optimized environment; compilers make errors. Some degree of optimization is expected,
88071 even if it is not documented here, and the ability to shut it off completely could be important
88072 when porting an application. An implementation may treat `-O 0` as “do less than normal” if it
88073 wishes, but this is only meaningful if any of the operations it performs can affect the semantics
88074 of a program. It is highly dependent on the implementation whether doing less than normal is
88075 logical. It is not the intent of the `-O 0` option to ask for inefficient code generation, but rather to
88076 assure that any semantically visible optimization is suppressed.

88077 The specification of standard library access is consistent with the C compiler specification.
88078 Implementations are not required to have `/usr/lib/libf.a`, as many historical implementations do,
88079 but if not they are required to recognize `f` as a token.

88080 External symbol size limits are in normative text; conforming applications need to know these
88081 limits. However, the minimum maximum symbol length should be taken as a constraint on a
88082 conforming application, not on an implementation, and consequently the action taken for a
88083 symbol exceeding the limit is unspecified. The minimum size for the external symbol table was
88084 added for similar reasons.

88085 The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when
88086 compilation or link-edit errors occur. The behavior of several historical implementations was
88087 examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in
88088 the face of compiler or linker errors. If a linker writes the executable file, then links it on disk
88089 with `lseek()`s and `write()`s, the partially linked executable file can be left on disk and its execute
88090 bits turned off if the link edit fails. However, if the linker links the image in memory before
88091 writing the file to disk, it need not touch the executable file (if it already exists) because the link
88092 edit fails. Since both approaches are historical practice, a conforming application shall rely on
88093 the exit status of `fort77`, rather than on the existence or mode of the executable file.

88094 The `-g` and `-s` options are not specified as mutually-exclusive. Historically these two options
88095 have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate
88096 to leave their interaction unspecified.

88097 The requirement that conforming applications specify compiler options separately is to reserve
88098 the multi-character option name space for vendor-specific compiler options, which are known to

88099 exist in many historical implementations. Implementations are not required to recognize, for
88100 example, `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The SYNOPSIS shows all
88101 of the options separately to highlight this requirement on applications.

88102 Echoing filenames to standard error is considered a diagnostic message because it would
88103 otherwise be difficult to associate an error message with the erring file. They are described with
88104 “may” to allow implementations to use other methods of identifying files and to parallel the
88105 description in *c99*.

88106 FUTURE DIRECTIONS

88107 A compilation system based on the ISO/IEC 1539:1990 standard (Fortran-90) may be considered
88108 for a future version; it may have a different utility name from *fort77*.

88109 SEE ALSO

88110 *ar*, *asa*, *c99*, *umask*

88111 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

88112 XSH *exec*

88113 CHANGE HISTORY

88114 First released in Issue 4.

88115 Issue 6

88116 This utility is marked as part of the FORTRAN Development Utilities option.

88117 The normative text is reworded to avoid use of the term “must” for application requirements.

88118 Issue 7

88119 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88120 **NAME**88121 `fuser` — list process IDs of all processes that have one or more files open88122 **SYNOPSIS**88123 XSI `fuser [-cfu] file...`88124 **DESCRIPTION**88125 The *fuser* utility shall write to standard output the process IDs of processes running on the local
88126 system that have one or more named files open. For block special devices, all processes using
88127 any file on that device are listed.88128 The *fuser* utility shall write to standard error additional information about the named files
88129 indicating how the file is being used.

88130 Any output for processes running on remote systems that have a named file open is unspecified.

88131 A user may need appropriate privilege to invoke the *fuser* utility.88132 **OPTIONS**88133 The *fuser* utility shall conform to XBD [Section 12.2](#) (on page 201).

88134 The following options shall be supported:

88135 `-c` The file is treated as a mount point and the utility shall report on any files open in
88136 the file system.88137 `-f` The report shall be only for the named files.88138 `-u` The user name, in parentheses, associated with each process ID written to standard
88139 output shall be written to standard error.88140 **OPERANDS**

88141 The following operand shall be supported:

88142 *file* A pathname on which the file or file system is to be reported.88143 **STDIN**

88144 Not used.

88145 **INPUT FILES**

88146 The user database.

88147 **ENVIRONMENT VARIABLES**88148 The following environment variables shall affect the execution of *fuser*:88149 *LANG* Provide a default value for the internationalization variables that are unset or null.
88150 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
88151 variables used to determine the values of locale categories.)88152 *LC_ALL* If set to a non-empty string value, override the values of all the other
88153 internationalization variables.88154 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88155 characters (for example, single-byte as opposed to multi-byte characters in
88156 arguments).88157 *LC_MESSAGES*88158 Determine the locale that should be used to affect the format and contents of
88159 diagnostic messages written to standard error.

88160 `NLSPATH` Determine the location of message catalogs for the processing of `LC_MESSAGES`.

88161 ASYNCHRONOUS EVENTS

88162 Default.

88163 STDOUT

88164 The *fuser* utility shall write the process ID for each process using each file given as an operand to
88165 standard output in the following format:

88166 `"%d", <process_id>`

88167 STDERR

88168 The *fuser* utility shall write diagnostic messages to standard error.

88169 The *fuser* utility also shall write the following to standard error:

- 88170 • The pathname of each named file is written followed immediately by a colon.
- 88171 • For each process ID written to standard output, the character 'c' shall be written to
88172 standard error if the process is using the file as its current directory and the character 'r'
88173 shall be written to standard error if the process is using the file as its root directory.
88174 Implementations may write other alphabetic characters to indicate other uses of files.
- 88175 • When the `-u` option is specified, characters indicating the use of the file shall be followed
88176 immediately by the user name, in parentheses, corresponding to the real user ID of the
88177 process. If the user name cannot be resolved from the real user ID of the process, the real
88178 user ID of the process shall be written instead of the user name.

88179 When standard output and standard error are directed to the same file, the output shall be
88180 interleaved so that the filename appears at the start of each line, followed by the process ID and
88181 characters indicating the use of the file. Then, if the `-u` option is specified, the user name or user
88182 ID for each process using that file shall be written.

88183 A `<newline>` shall be written to standard error after the last output described above for each *file*
88184 operand.

88185 OUTPUT FILES

88186 None.

88187 EXTENDED DESCRIPTION

88188 None.

88189 EXIT STATUS

88190 The following exit values shall be returned:

88191 0 Successful completion.

88192 >0 An error occurred.

88193 CONSEQUENCES OF ERRORS

88194 Default.

88195 APPLICATION USAGE

88196 None.

88197 EXAMPLES

88198 The command:

88199 `fuser -fu .`

88200 writes to standard output the process IDs of processes that are using the current directory and
88201 writes to standard error an indication of how those processes are using the directory and the
88202 user names associated with the processes that are using the current directory.

88203 `fuser -c <mount point>`

88204 writes to standard output the process IDs of processes that are using any file in the file system
88205 which is mounted on *<mount point>* and writes to standard error an indication of how those
88206 processes are using the files.

88207 `fuser <mount point>`

88208 writes to standard output the process IDs of processes that are using the file which is named by
88209 *<mount point>* and writes to standard error an indication of how those processes are using the
88210 file.

88211 `fuser <block device>`

88212 writes to standard output the process IDs of processes that are using any file which is on the
88213 device named by *<block device>* and writes to standard error an indication of how those
88214 processes are using the file.

88215 `fuser --f <block device>`

88216 writes to standard output the process IDs of processes that are using the file *<block device>* itself
88217 and writes to standard error an indication of how those processes are using the file.

88218 **RATIONALE**

88219 The definition of the *fuser* utility follows existing practice.

88220 **FUTURE DIRECTIONS**

88221 None.

88222 **SEE ALSO**

88223 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

88224 **CHANGE HISTORY**

88225 First released in Issue 5.

88226 **Issue 7**

88227 SD5-XCU-ERN-90 is applied, updating the EXAMPLES section.

88228 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88229 **NAME**

88230 gencat — generate a formatted message catalog

88231 **SYNOPSIS**88232 gencat *catfile* *msgfile*...88233 **DESCRIPTION**

88234 The *gencat* utility shall merge the message text source file *msgfile* into a formatted message
 88235 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
 88236 messages shall be included in the new *catfile*. If set and message numbers collide, the new
 88237 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

88238 **OPTIONS**

88239 None.

88240 **OPERANDS**

88241 The following operands shall be supported:

88242 *catfile* A pathname of the formatted message catalog. If '-' is specified, standard output
 88243 shall be used. The format of the message catalog produced is unspecified.

88244 *msgfile* A pathname of a message text source file. If '-' is specified for an instance of
 88245 *msgfile*, standard input shall be used. The format of message text source files is
 88246 defined in the EXTENDED DESCRIPTION section.

88247 **STDIN**88248 The standard input shall not be used unless a *msgfile* operand is specified as '-'.88249 **INPUT FILES**

88250 The input files shall be text files.

88251 **ENVIRONMENT VARIABLES**88252 The following environment variables shall affect the execution of *gencat*:

88253 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 88254 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 88255 variables used to determine the values of locale categories.)

88256 *LC_ALL* If set to a non-empty string value, override the values of all the other
 88257 internationalization variables.

88258 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 88259 characters (for example, single-byte as opposed to multi-byte characters in
 88260 arguments and input files).

88261 *LC_MESSAGES*

88262 Determine the locale that should be used to affect the format and contents of
 88263 diagnostic messages written to standard error.

88264 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

88265 **ASYNCHRONOUS EVENTS**

88266 Default.

88267 **STDOUT**88268 The standard output shall not be used unless the *catfile* operand is specified as '-'.

88269 **STDERR**
 88270 The standard error shall be used only for diagnostic messages.

88271 **OUTPUT FILES**
 88272 None.

88273 EXTENDED DESCRIPTION

88274 The content of a message text file shall be in the format defined as follows. Note that the fields of
 88275 a message text source line are separated by a single <blank>. Any other <blank>s are
 88276 considered to be part of the subsequent field.

88277 **\$set** *n comment*
 88278 This line specifies the set identifier of the following messages until the next **\$set** or
 88279 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
 88280 in the range [1, {NL_SETMAX}] (see the <limits.h> header defined in the Base
 88281 Definitions volume of POSIX.1-200x). The application shall ensure that set
 88282 identifiers are presented in ascending order within a single source file, but need
 88283 not be contiguous. Any string following the set identifier shall be treated as a
 88284 comment. If no **\$set** directive is specified in a message text source file, all messages
 88285 shall be located in an implementation-defined default message set NL_SETD (see
 88286 the <nl_types.h> header defined in the Base Definitions volume of POSIX.1-200x).

88287 **\$delset** *n comment*
 88288 This line deletes message set *n* from an existing message catalog. The *n* denotes the
 88289 set number [1, {NL_SETMAX}]. Any string following the set number shall be
 88290 treated as a comment.

88291 **\$** *comment* A line beginning with ' \$ ' followed by a <blank> shall be treated as a comment.

88292 *m message-text*
 88293 The *m* denotes the message identifier, which is defined as a number in the range [1,
 88294 {NL_MSGMAX}] (see the <limits.h> header). The *message-text* shall be stored in the
 88295 message catalog with the set identifier specified by the last **\$set** directive, and with
 88296 message identifier *m*. If the *message-text* is empty, and a <blank> field separator is
 88297 present, an empty string shall be stored in the message catalog. If a message source
 88298 line has a message number, but neither a field separator nor *message-text*, the
 88299 existing message with that number (if any) shall be deleted from the catalog. The
 88300 application shall ensure that message identifiers are in ascending order within a
 88301 single set, but need not be contiguous. The application shall ensure that the length
 88302 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the <limits.h> header).

88303 **\$quote** *n* This line specifies an optional quote character *c*, which can be used to surround
 88304 *message-text* so that trailing spaces or null (empty) messages are visible in a
 88305 message source line. By default, or if an empty **\$quote** directive is supplied, no
 88306 quoting of *message-text* shall be recognized.

88307 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
 88308 character other than those defined above are implementation-defined.

88309 Text strings can contain the special characters and escape sequences defined in the following
 88310 table:

88311
88312
88313
88314
88315
88316
88317
88318
88319

Description	Symbol	Sequence
<newline>	NL(LF)	\n
Horizontal-tab	HT	\t
<vertical-tab>	VT	\v
<backspace>	BS	\b
<carriage-return>	CR	\r
<form-feed>	FF	\f
Backslash	\	\\
Bit pattern	ddd	\ddd

88320
88321
88322

The escape sequence "\ddd" consists of backslash followed by one, two, or three octal digits, which shall be taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash shall be ignored.

88323
88324

Backslash ('\'') followed by a <newline> is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

88325
88326

```
1 This line continues \  
to the next line
```

88327

which shall be equivalent to:

88328

```
1 This line continues to the next line
```

88329

EXIT STATUS

88330

The following exit values shall be returned:

88331

0 Successful completion.

88332

>0 An error occurred.

88333

CONSEQUENCES OF ERRORS

88334

Default.

88335

APPLICATION USAGE

88336

Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

88337

88338

88339

EXAMPLES

88340

None.

88341

RATIONALE

88342

None.

88343

FUTURE DIRECTIONS

88344

None.

88345

SEE ALSO

88346

iconv

88347

XBD Chapter 8 (on page 159), <limits.h>, <nl_types.h>

88348

CHANGE HISTORY

88349

First released in Issue 3.

88350

Issue 6

88351

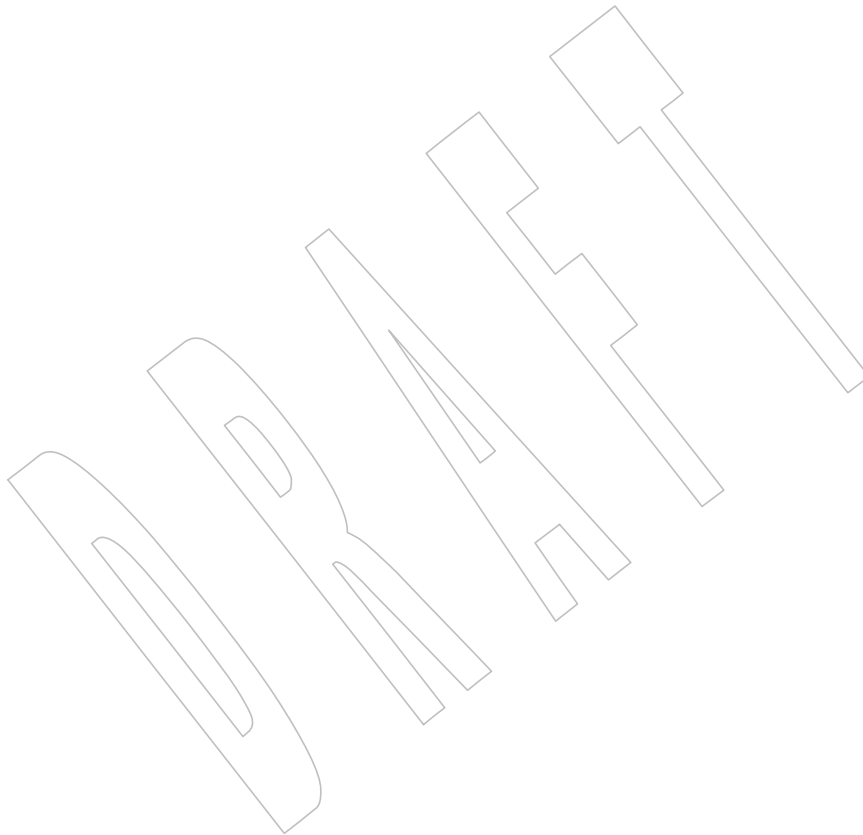
The normative text is reworded to avoid use of the term “must” for application requirements.

88352

Issue 7

88353

The *gencat* utility is moved from the XSI option to the Base.



88354 **NAME**
 88355 `get` — get a version of an SCCS file (**DEVELOPMENT**)

88356 **SYNOPSIS**
 88357 `get [-begkmnlpst] [-c cutoff] [-i list] [-r SID] [-x list] file...`

88358 **DESCRIPTION**
 88359 The `get` utility shall generate a text file from each named SCCS *file* according to the specifications
 88360 given by its options.

88361 The generated text shall normally be written into a file called the **g-file** whose name is derived
 88362 from the SCCS filename by simply removing the leading "s. ".

88363 **OPTIONS**
 88364 The `get` utility shall conform to XBD [Section 12.2](#) (on page 201).

88365 The following options shall be supported:

88366 **-r *SID*** Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file
 88367 to be retrieved. The table shows, for the most useful cases, what version of an
 88368 SCCS file is retrieved (as well as the SID of the version to be eventually created by
 88369 *delta* if the **-e** option is also used), as a function of the SID specified.

88370 **-c *cutoff*** Indicate the *cutoff* date-time, in the form:

88371 `YY[MM[DD[HH[MM[SS]]]]]`

88372 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999
 88373 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

88374 **Note:** It is expected that in a future version of this standard the default century inferred
 88375 from a 2-digit year will change. (This would apply to all commands accepting a
 88376 2-digit year as input.)

88377 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
 88378 date-time shall be included in the generated text file. Units omitted from the date-
 88379 time default to their maximum possible values; for example, **-c 7502** is equivalent
 88380 to **-c 750228235959**.

88381 Any number of non-numeric characters may separate the various 2-digit pieces of
 88382 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:
 88383 **-c "77/2/2 9:22:25"**.

88384 **-e** Indicate that the `get` is for the purpose of editing or making a change (delta) to the
 88385 SCCS file via a subsequent use of *delta*. The **-e** option used in a `get` for a particular
 88386 version (SID) of the SCCS file shall prevent further `get` commands from editing on
 88387 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.
 88388 Concurrent use of `get -e` for different SIDs is always allowed.

88389 If the **g-file** generated by `get` with a **-e** option is accidentally ruined in the process
 88390 of editing, it may be regenerated by re-executing the `get` command with the **-k**
 88391 option in place of the **-e** option.

88392 SCCS file protection specified via the ceiling, floor, and authorized user list stored
 88393 in the SCCS file shall be enforced when the **-e** option is used.

88394 **-b** Use with the **-e** option to indicate that the new delta should have an SID in a new
 88395 branch as shown in the table below. This option shall be ignored if the **b** flag is not
 88396 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that

- 88397 has no successors on the SCCS file tree.)
- 88398 **Note:** A branch delta may always be created from a non-leaf delta.
- 88399 **-i list** Indicate a *list* of deltas to be included (forced to be applied) in the creation of the
88400 generated file. The *list* has the following syntax:
- 88401 `<list> ::= <range> | <list> , <range>`
88402 `<range> ::= SID | SID - SID`
- 88403 SID, the SCCS Identification of a delta, may be in any form shown in the "SID
88404 Specified" column of the table in the EXTENDED DESCRIPTION section, except
88405 that the result of supplying a partial SID is unspecified. A diagnostic message shall
88406 be written if the first SID in the range is not an ancestor of the second SID in the
88407 range.
- 88408 **-x list** Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of
88409 the generated file. See the **-i** option for the *list* format.
- 88410 **-k** Suppress replacement of identification keywords (see below) in the retrieved text
88411 by their value. The **-k** option shall be implied by the **-e** option.
- 88412 **-l** Write a delta summary into an **l-file**.
- 88413 **-L** Write a delta summary to standard output. All informative output that normally is
88414 written to standard output shall be written to standard error instead, unless the **-s**
88415 option is used, in which case it shall be suppressed.
- 88416 **-p** Write the text retrieved from the SCCS file to the standard output. No **g-file** shall
88417 be created. All informative output that normally goes to the standard output shall
88418 go to standard error instead, unless the **-s** option is used, in which case it shall
88419 disappear.
- 88420 **-s** Suppress all informative output normally written to standard output. However,
88421 fatal error messages (which shall always be written to the standard error) shall
88422 remain unaffected.
- 88423 **-m** Precede each text line retrieved from the SCCS file by the SID of the delta that
88424 inserted the text line in the SCCS file. The format shall be:
- 88425 `"%s\t%s", <SID>, <text line>`
- 88426 **-n** Precede each generated text line with the **%M%** identification keyword value (see
88427 below). The format shall be:
- 88428 `"%s\t%s", <%M% value>, <text line>`
- 88429 When both the **-m** and **-n** options are used, the `<text line>` shall be replaced by the
88430 **-m** option-generated format.
- 88431 **-g** Suppress the actual retrieval of text from the SCCS file. It is primarily used to
88432 generate an **l-file**, or to verify the existence of a particular SID.
- 88433 **-t** Use to access the most recently created (top) delta in a given release (for example,
88434 **-r 1**), or release and level (for example, **-r 1.2**).

88435 **OPERANDS**

88436 The following operands shall be supported:

88437 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get*
 88438 utility shall behave as though each file in the directory were specified as a named
 88439 file, except that non-SCCS files (last component of the pathname does not begin
 88440 with **s**.) and unreadable files shall be silently ignored.

88441 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;
 88442 each line of the standard input is taken to be the name of an SCCS file to be
 88443 processed. Non-SCCS files and unreadable files shall be silently ignored.

88444 **STDIN**

88445 The standard input shall be a text file used only if the *file* operand is specified as *'-'*. Each line
 88446 of the text file shall be interpreted as an SCCS pathname.

88447 **INPUT FILES**

88448 The SCCS files shall be files of an unspecified format.

88449 **ENVIRONMENT VARIABLES**88450 The following environment variables shall affect the execution of *get*:

88451 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 88452 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 88453 variables used to determine the values of locale categories.)

88454 *LC_ALL* If set to a non-empty string value, override the values of all the other
 88455 internationalization variables.

88456 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 88457 characters (for example, single-byte as opposed to multi-byte characters in
 88458 arguments and input files).

88459 *LC_MESSAGES*
 88460 Determine the locale that should be used to affect the format and contents of
 88461 diagnostic messages written to standard error, and informative messages written
 88462 to standard output (or standard error, if the *-p* option is used).

88463 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

88464 *TZ* Determine the timezone in which the times and dates written in the SCCS file are
 88465 evaluated. If the *TZ* variable is unset or NULL, an unspecified system default
 88466 timezone is used.

88467 **ASYNCHRONOUS EVENTS**

88468 Default.

88469 **STDOUT**

88470 For each file processed, *get* shall write to standard output the SID being accessed and the
 88471 number of lines retrieved from the SCCS file, in the following format:

88472 "%s\n%d lines\n", <SID>, <number of lines>

88473 If the *-e* option is used, the SID of the delta to be made shall appear after the SID accessed and
 88474 before the number of lines generated, in the POSIX locale:

88475 "%s\nnew delta %s\n%d lines\n", <SID accessed>,
 88476 <SID to be made>, <number of lines>

88477 If there is more than one named file or if a directory or standard input is named, each pathname
 88478 shall be written before each of the lines shown in one of the preceding formats:

88479 "\n%s:\n", <pathname>

88480 If the **-L** option is used, a delta summary shall be written following the format specified below
88481 for **l-files**.

88482 If the **-i** option is used, included deltas shall be listed following the notation, in the POSIX
88483 locale:

88484 "Included:\n"

88485 If the **-x** option is used, excluded deltas shall be listed following the notation, in the POSIX
88486 locale:

88487 "Excluded:\n"

88488 If the **-p** or **-L** options are specified, the standard output shall consist of the text retrieved from
88489 the SCCS file.

88490 STDERR

88491 The standard error shall be used only for diagnostic messages, except if the **-p** or **-L** options are
88492 specified, it shall include all informative messages normally sent to standard output.

88493 OUTPUT FILES

88494 Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-**
88495 **file**, **p-file**, and **z-file**. The letter before the hyphen is called the *tag*. An auxiliary filename shall
88496 be formed from the SCCS filename: the application shall ensure that the last component of all
88497 SCCS filenames is of the form *s.module-name*; the auxiliary files shall be named by replacing the
88498 leading *s* with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by
88499 removing the *s*. prefix. For example, for *s.xyz.c*, the auxiliary filenames would be *xyz.c*, *l.xyz.c*,
88500 *p.xyz.c*, and *z.xyz.c*, respectively.

88501 The **g-file**, which contains the generated text, shall be created in the current directory (unless the
88502 **-p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were
88503 generated by the *get*. It shall be owned by the real user. If the **-k** option is used or implied, the
88504 **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be
88505 read-only. Only the real user need have write permission in the current directory.

88506 The **l-file** shall contain a table showing which deltas were applied in generating the retrieved
88507 text. The **l-file** shall be created in the current directory if the **-l** option is used; it shall be read-
88508 only and it is owned by the real user. Only the real user need have write permission in the
88509 current directory.

88510 Lines in the **l-file** shall have the following format:

88511 "%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,
88512 <SID>, <date-time>, <login>

88513 where the entries are:

88514 <code1> A <space> if the delta was applied; ' * ' otherwise.

88515 <code2> A <space> if the delta was applied or was not applied and ignored; ' * ' if the delta
88516 was not applied and was not ignored.

88517 <code3> A character indicating a special reason why the delta was or was not applied:

88518 I Included.

88519 X Excluded.

88520 C Cut off (by a **-c** option).

88521 <date-time> Date and time (using the format of the *date* utility's %Y/%m/%d %T conversion
88522 specification format) of creation.

88523 <*login*> Login name of person who created *delta*.

88524 The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line
88525 shall terminate each entry.

88526 The **p-file** shall be used to pass information resulting from a *get* with a **-e** option along to *delta*.
88527 Its contents shall also be used to prevent a subsequent execution of *get* with a **-e** option for the
88528 same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be
88529 created in the directory containing the SCCS file and the application shall ensure that the
88530 effective user has write permission in that directory. It shall be writable by owner only, and
88531 owned by the effective user. Each line in the **p-file** shall have the following format:

```
88532 "%sΔ%sΔ%sΔ%s%s\n", <g-file SID>,  
88533           <SID of new delta>, <login-name of real user>,  
88534           <date-time>, <i-value>, <x-value>
```

88535 where <*i-value*> uses the format " " if no **-i** option was specified, and shall use the format:

```
88536 "Δ-i%s", <-i option option-argument>
```

88537 if a **-i** option was specified and <*x-value*> uses the format " " if no **-x** option was specified, and
88538 shall use the format:

```
88539 "Δ-x%s", <-x option option-argument>
```

88540 if a **-x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time;
88541 no two lines shall have the same new delta SID.

88542 The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall
88543 be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created
88544 in the directory containing the SCCS file for the duration of *get*. The same protection restrictions
88545 as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.

88546

EXTENDED DESCRIPTION

88547

Determination of SCCS Identification String				
SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk successor	R.L	R.(L+1)
R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	-	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch successor	R.L.B.S	R.L.(mB+1).1

88569

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means "the maximum level number within release R"; R.L.(mB+1).1 means "the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

88570

88571

88572

88573

88574

88575

** hR is the highest existing release that is lower than the specified, nonexistent, release R.

88576

*** This is used to force creation of the first delta in a new release.

88577

† The -b option is effective only if the b flag is present in the file. An entry of '-' means "irrelevant".

88578

88579

‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

88580

88581

88582

System Date and Time

88583

When a g-file is generated, the creation time of deltas in the SCCS file may be taken into account. If any of these times are apparently in the future, the behavior is unspecified.

88584

88585

Identification Keywords

88586

Identifying information shall be inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

88587

88588

88589

%M% Module name: either the value of the **m** flag in the file, or if absent, the name of the SCCS file with the leading **s**. removed.

88590

88591

%I% SCCS identification (SID) (**%R%.%L%** or **%R%.%L%.%B%.%S%**) of the retrieved text.

88592

88593

%R% Release.

88594

%L% Level.

88595

%B% Branch.

88596

%S% Sequence.

88597

%D% Current date (**YY/MM/DD**).

88598

%H% Current date (**MM/DD/YY**).

88599

%T% Current time (**HH:MM:SS**).

88600

%E% Date newest applied delta was created (**YY/MM/DD**).

88601

%G% Date newest applied delta was created (**MM/DD/YY**).

88602

%U% Time newest applied delta was created (**HH:MM:SS**).

88603

%Y% Module type: value of the **t** flag in the SCCS file.

88604

%F% SCCS filename.

88605

%P% SCCS absolute pathname.

88606

%Q% The value of the **q** flag in the file.

88607

%C% Current line number. This keyword is intended for identifying messages output by the program, such as "this should not have happened" type errors. It is not intended to be used on every line to provide sequence numbers.

88608

88609

88610

%Z% The four-character string "@(#)" recognizable by *what*.

88611

%W% A shorthand notation for constructing *what* strings:

88612

%W% = **%Z%** **%M%** <tab> **%I%**

88613

%A% Another shorthand notation for constructing *what* strings:

88614

%A% = **%Z%** **%Y%** **%M%** **%I%** **%Z%**

88615

EXIT STATUS

88616

The following exit values shall be returned:

88617

0 Successful completion.

88618

>0 An error occurred.

88619

CONSEQUENCES OF ERRORS

88620

Default.

88621 **APPLICATION USAGE**

88622 Problems can arise if the system date and time have been modified (for example, put forward
88623 and then back again, or unsynchronized clocks across a network) and can also arise when
88624 different values of the *TZ* environment variable are used.

88625 Problems of a similar nature can also arise for the operation of the *delta* utility, which compares
88626 the previous file body against the working file as part of its normal operation.

88627 **EXAMPLES**

88628 None.

88629 **RATIONALE**

88630 None.

88631 **FUTURE DIRECTIONS**

88632 None.

88633 **SEE ALSO**

88634 *admin, delta, prs, what*

88635 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

88636 **CHANGE HISTORY**

88637 First released in Issue 2.

88638 **Issue 5**

88639 A correction is made to the first format string in `STDOUT`.

88640 The interpretation of the *YY* component of the `-c cutoff` argument is noted.

88641 **Issue 6**

88642 The obsolescent `SYNOPSIS` is removed, removing the `-lp` option.

88643 The normative text is reworded to avoid use of the term “must” for application requirements.

88644 The Open Group Corrigendum U025/5 is applied, correcting text in the `OPTIONS` section.

88645 The Open Group Corrigendum U048/1 is applied.

88646 The Open Group Interpretation PIN4C.00014 is applied.

88647 The Open Group Base Resolution bwg2001-007 is applied as follows:

- 88648 • The `EXTENDED DESCRIPTION` section is updated to make partial `SID` handling
88649 unspecified, reflecting common usage, and to clarify `SID` ranges.
- 88650 • New text is added to the `EXTENDED DESCRIPTION` and `APPLICATION USAGE` sections
88651 regarding how the system date and time may be taken into account.
- 88652 • The *TZ* environment variable is added to the `ENVIRONMENT VARIABLES` section.

88653 **Issue 7**

88654 `SD5-XCU-ERN-97` is applied, updating the `SYNOPSIS`.

88655 **NAME**88656 `getconf` — get configuration values88657 **SYNOPSIS**88658 `getconf [-v specification] system_var`88659 `getconf [-v specification] path_var pathname`88660 **DESCRIPTION**88661 In the first synopsis form, the `getconf` utility shall write to the standard output the value of the
88662 variable specified by the `system_var` operand.88663 In the second synopsis form, the `getconf` utility shall write to the standard output the value of the
88664 variable specified by the `path_var` operand for the path specified by the `pathname` operand.88665 The value of each configuration variable shall be determined as if it were obtained by calling the
88666 function from which it is defined to be available by this volume of POSIX.1-200x or by the
88667 System Interfaces volume of POSIX.1-200x (see the OPERANDS section). The value shall reflect
88668 conditions in the current operating environment.88669 **OPTIONS**88670 The `getconf` utility shall conform to XBD [Section 12.2](#) (on page 201).

88671 The following option shall be supported:

88672 **-v specification**88673 Indicate a specific specification and version for which configuration variables shall
88674 be determined. If this option is not specified, the values returned correspond to an
88675 implementation default conforming compilation environment.

88676 If the command:

88677 `getconf _POSIX_V7_ILP32_OFF32`88678 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
88679 the form:88680 `getconf -v POSIX_V7_ILP32_OFF32 ...`88681 determine values for configuration variables corresponding to the
88682 `POSIX_V7_ILP32_OFF32` compilation environment specified in [c99](#), the
88683 EXTENDED DESCRIPTION.

88684 If the command:

88685 `getconf _POSIX_V7_ILP32_OFFBIG`88686 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
88687 the form:88688 `getconf -v POSIX_V7_ILP32_OFFBIG ...`88689 determine values for configuration variables corresponding to the
88690 `POSIX_V7_ILP32_OFFBIG` compilation environment specified in [c99](#), the
88691 EXTENDED DESCRIPTION.

88692 If the command:

88693 `getconf _POSIX_V7_LP64_OFF64`88694 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of
88695 the form:

88696 `getconf -v POSIX_V7_LP64_OFF64 ...`

88697 determine values for configuration variables corresponding to the

88698 `POSIX_V7_LP64_OFF64` compilation environment specified in [c99](#), the

88699 EXTENDED DESCRIPTION.

88700 If the command:

88701 `getconf _POSIX_V7_LPBIG_OFFBIG`

88702 does not write `"-1\n"` or `"undefined\n"` to standard output, then commands of

88703 the form:

88704 `getconf -v POSIX_V7_LPBIG_OFFBIG ...`

88705 determine values for configuration variables corresponding to the

88706 `POSIX_V7_LPBIG_OFFBIG` compilation environment specified in [c99](#), the

88707 EXTENDED DESCRIPTION.

OPERANDS

88708 The following operands shall be supported:

88709 *path_var* A name of a configuration variable. All of the variables in the Variable column of

88710 the table in the DESCRIPTION of the `fpathconf()` function defined in the System

88711 Interfaces volume of POSIX.1-200x, without the enclosing braces, shall be

88712 supported. The implementation may add other local variables.

88713

88714 *pathname* A pathname for which the variable specified by *path_var* is to be determined.

88715 *system_var* A name of a configuration variable. All of the following variables shall be

88716 supported:

- 88717 • The names in the Variable column of the table in the DESCRIPTION of the
- 88718 `sysconf()` function in the System Interfaces volume of POSIX.1-200x, except
- 88719 for the entries corresponding to `_SC_CLK_TCK`, `_SC_GETGR_R_SIZE_MAX`,
- 88720 and `_SC_GETPW_R_SIZE_MAX`, without the enclosing braces.

88721 For compatibility with earlier versions, the following variable names shall

88722 also be supported:

88723 `POSIX2_C_BIND`

88724 `POSIX2_C_DEV`

88725 `POSIX2_CHAR_TERM`

88726 `POSIX2_FORT_DEV`

88727 `POSIX2_FORT_RUN`

88728 `POSIX2_LOCALEDEF`

88729 `POSIX2_SW_DEV`

88730 `POSIX2_UPE`

88731 `POSIX2_VERSION`

88732 and shall be equivalent to the same name prefixed with an underscore. This

88733 requirement may be removed in a future version.

- 88734 • The names of the symbolic constants used as the *name* argument of the
- 88735 `confstr()` function in the System Interfaces volume of POSIX.1-200x, without
- 88736 the `_CS_` prefix.
- 88737 • The names of the symbolic constants listed under the headings “Maximum
- 88738 Values” and “Minimum Values” in the description of the `<limits.h>` header
- 88739 in the Base Definitions volume of POSIX.1-200x, without the enclosing
- 88740 braces.

88741 For compatibility with earlier versions, the following variable names shall
88742 also be supported:

88743 POSIX2_BC_BASE_MAX
88744 POSIX2_BC_DIM_MAX
88745 POSIX2_BC_SCALE_MAX
88746 POSIX2_BC_STRING_MAX
88747 POSIX2_COLL_WEIGHTS_MAX
88748 POSIX2_EXPR_NEST_MAX
88749 POSIX2_LINE_MAX
88750 POSIX2_RE_DUP_MAX

88751 and shall be equivalent to the same name prefixed with an underscore. This
88752 requirement may be removed in a future version.

88753 The implementation may add other local values.

88754 **STDIN**

88755 Not used.

88756 **INPUT FILES**

88757 None.

88758 **ENVIRONMENT VARIABLES**

88759 The following environment variables shall affect the execution of *getconf*:

88760 *LANG* Provide a default value for the internationalization variables that are unset or null. |
88761 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
88762 variables used to determine the values of locale categories.)

88763 *LC_ALL* If set to a non-empty string value, override the values of all the other
88764 internationalization variables.

88765 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88766 characters (for example, single-byte as opposed to multi-byte characters in
88767 arguments).

88768 *LC_MESSAGES*
88769 Determine the locale that should be used to affect the format and contents of
88770 diagnostic messages written to standard error.

88771 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

88772 **ASYNCHRONOUS EVENTS**

88773 Default.

88774 **STDOUT**

88775 If the specified variable is defined on the system and its value is described to be available from
88776 the *confstr()* function defined in the System Interfaces volume of POSIX.1-200x, its value shall be
88777 written in the following format:

88778 "%s\n", <value>

88779 Otherwise, if the specified variable is defined on the system, its value shall be written in the
88780 following format:

88781 "%d\n", <value>

88782 If the specified variable is valid, but is undefined on the system, *getconf* shall write using the
88783 following format:

88784 "undefined\n"

88785 If the variable name is invalid or an error occurs, nothing shall be written to standard output.

88786 **STDERR**

88787 The standard error shall be used only for diagnostic messages.

88788 **OUTPUT FILES**

88789 None.

88790 **EXTENDED DESCRIPTION**

88791 None.

88792 **EXIT STATUS**

88793 The following exit values shall be returned:

88794 0 The specified variable is valid and information about its current state was written
88795 successfully.

88796 >0 An error occurred.

88797 **CONSEQUENCES OF ERRORS**

88798 Default.

88799 **APPLICATION USAGE**

88800 None.

88801 **EXAMPLES**

88802 The following example illustrates the value of {NGROUPS_MAX}:

88803 `getconf NGROUPS_MAX`

88804 The following example illustrates the value of {NAME_MAX} for a specific directory:

88805 `getconf NAME_MAX /usr`

88806 The following example shows how to deal more carefully with results that might be unspecified:

```
88807 if value=$(getconf PATH_MAX /usr); then
88808     if [ "$value" = "undefined" ]; then
88809         echo PATH_MAX in /usr is indeterminate.
88810     else
88811         echo PATH_MAX in /usr is $value.
88812     fi
88813 else
88814     echo Error in getconf.
88815 fi
```

88816 Note that:

88817 `sysconf(_SC_2_C_BIND);`

88818 and:

88819 `system("getconf _POSIX2_C_BIND");`

88820 in a C program could give different answers. The *sysconf()* call supplies a value that corresponds
88821 to the conditions when the program was either compiled or executed, depending on the
88822 implementation; the *system()* call to *getconf* always supplies a value corresponding to conditions
88823 when the program is executed.

88824 **RATIONALE**

88825 The original need for this utility, and for the *confstr()* function, was to provide a way of finding
88826 the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be
88827 modified by the user to include directories that could contain utilities replacing the standard
88828 utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable
88829 value that contains the correct search path for the standard utilities. It was later suggested that

88830 access to the other variables described in this volume of POSIX.1-200x could also be useful to
88831 applications.

88832 This functionality of *getconf* would not be adequately subsumed by another command such as:

88833 `grep var /etc/conf`

88834 because such a strategy would provide correct values for neither those variables that can vary at
88835 runtime, nor those that can vary depending on the path.

88836 Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid,
88837 but not defined on the system. The output string "undefined" is now used to specify this case
88838 with exit code 0 because so many things depend on an exit code of zero when an invoked utility
88839 is successful.

88840 FUTURE DIRECTIONS

88841 None.

88842 SEE ALSO

88843 [c99](#)

88844 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [<limits.h>](#) |

88845 XSH [confstr\(\)](#), [fpathconf\(\)](#), [sysconf\(\)](#), [system\(\)](#)

88846 CHANGE HISTORY

88847 First released in Issue 4.

88848 Issue 5

88849 In the OPERANDS section:

- 88850 • {NL_MAX} is changed to {NL_NMAX}.
- 88851 • Entries beginning NL_ are deleted from the list of standard configuration variables.
- 88852 • The list of variables previously marked UX is merged with the list marked EX.
- 88853 • Operands are added to support new Option Groups.
- 88854 • Operands are added so that *getconf* can determine supported programming environments.

88855 Issue 6

88856 The Open Group Corrigendum U029/4 is applied, correcting the example command in the last
88857 paragraph of the OPTIONS section.

88858 The following new requirements on POSIX implementations derive from alignment with the
88859 Single UNIX Specification:

- 88860 • Operands are added to determine supported programming environments.

88861 This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically,
88862 new macros for *c99* programming environments are introduced.

88863 XSI marked *system_var* (XBS5_*) values are marked LEGACY.

88864 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/27 is applied, correcting the descriptions
88865 of *path_var* and *system_var* in the OPERANDS section.

88866 Issue 7

88867 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

88868 The EXAMPLES section is corrected. +

88869 **NAME**88870 `getopts` — parse utility options88871 **SYNOPSIS**88872 `getopts optstring name [arg...]`88873 **DESCRIPTION**88874 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall
88875 support the Utility Syntax Guidelines 3 to 10, inclusive, described in XBD [Section 12.2](#) (on page
88876 201).88877 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell
88878 variable specified by the *name* operand and the index of the next argument to be processed in the
88879 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.88880 When the option requires an option-argument, the *getopts* utility shall place it in the shell
88881 variable *OPTARG*. If no option was found, or if the option that was found does not have an
88882 option-argument, *OPTARG* shall be unset.88883 If an option character not contained in the *optstring* operand is found where an option character
88884 is expected, the shell variable specified by *name* shall be set to the question-mark ('?') character.
88885 In this case, if the first character in *optstring* is a colon (':'), the shell variable *OPTARG* shall be
88886 set to the option character found, but no output shall be written to standard error; otherwise, the
88887 shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard
88888 error. This condition shall be considered to be an error detected in the way arguments were
88889 presented to the invoking application, but shall not be an error in *getopts* processing.

88890 If an option-argument is missing:

- 88891
- If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set
88892 to the colon character and the shell variable *OPTARG* shall be set to the option character
88893 found.
 - Otherwise, the shell variable specified by *name* shall be set to the question-mark character,
88894 the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to
88895 standard error. This condition shall be considered to be an error detected in the way
88896 arguments were presented to the invoking application, but shall not be an error in *getopts*
88897 processing; a diagnostic message shall be written as stated, but the exit status shall be zero.
88898

88899 When the end of options is encountered, the *getopts* utility shall exit with a return value greater
88900 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,
88901 where the first "--" argument is considered to be an option-argument if there are no other non-
88902 option-arguments appearing before it, or the value "\$#+1" if there are no non-option-
88903 arguments; the *name* variable shall be set to the question-mark character. Any of the following
88904 shall identify the end of options: the special option "--", finding an argument that does not
88905 begin with a '-', or encountering an error.88906 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be
88907 exported by default.88908 The shell variable specified by the *name* operand, *OPTIND*, and *OPTARG* shall affect the current
88909 shell execution environment; see [Section 2.12](#) (on page 2277).88910 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the
88911 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple
88912 times in a single shell execution environment with parameters (positional parameters or *arg*
88913 operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a
88914 value other than 1, produces unspecified results.

88915 **OPTIONS**
88916 None.

88917 **OPERANDS**
88918 The following operands shall be supported:

88919 *optstring* A string containing the option characters recognized by the utility invoking *getopts*.
88920 If a character is followed by a colon, the option shall be expected to have an
88921 argument, which should be supplied as a separate argument. Applications should
88922 specify an option character and its option-argument as separate arguments, but
88923 *getopts* shall interpret the characters following an option character requiring
88924 arguments as an argument whether or not this is done. An explicit null option-
88925 argument need not be recognized if it is not supplied as a separate argument when
88926 *getopts* is invoked. (See also the *getopt()* function defined in the System Interfaces
88927 volume of POSIX.1-200x.) The characters question-mark and colon shall not be
88928 used as option characters by an application. The use of other option characters that
88929 are not alphanumeric produces unspecified results. If the option-argument is not
88930 supplied as a separate argument from the option character, the value in *OPTARG*
88931 shall be stripped of the option character and the '-'. The first character in
88932 *optstring* determines how *getopts* behaves if an option character is not known or an
88933 option-argument is missing.

88934 *name* The name of a shell variable that shall be set by the *getopts* utility to the option
88935 character that was found.

88936 The *getopts* utility by default shall parse positional parameters passed to the invoking shell
88937 procedure. If *args* are given, they shall be parsed instead of the positional parameters.

88938 **STDIN**
88939 Not used.

88940 **INPUT FILES**
88941 None.

88942 **ENVIRONMENT VARIABLES**
88943 The following environment variables shall affect the execution of *getopts*:

88944 *LANG* Provide a default value for the internationalization variables that are unset or null. |
88945 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
88946 variables used to determine the values of locale categories.)

88947 *LC_ALL* If set to a non-empty string value, override the values of all the other
88948 internationalization variables.

88949 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
88950 characters (for example, single-byte as opposed to multi-byte characters in
88951 arguments and input files).

88952 *LC_MESSAGES*
88953 Determine the locale that should be used to affect the format and contents of
88954 diagnostic messages written to standard error.

88955 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

88956 *OPTIND* This variable shall be used by the *getopts* utility as the index of the next argument
88957 to be processed.

88958 **ASYNCHRONOUS EVENTS**

88959 Default.

88960 **STDOUT**

88961 Not used.

88962 **STDERR**

88963 Whenever an error is detected and the first character in the *optstring* operand is not a colon
 88964 (' : '), a diagnostic message shall be written to standard error with the following information in
 88965 an unspecified format:

- 88966 • The invoking program name shall be identified in the message. The invoking program
 88967 name shall be the value of the shell special parameter 0 (see [Section 2.5.2](#), on page 2250) at
 88968 the time the *getopts* utility is invoked. A name equivalent to:

88969 `basename "$0"`

88970 may be used.

- 88971 • If an option is found that was not specified in *optstring*, this error is identified and the
 88972 invalid option character shall be identified in the message.

- 88973 • If an option requiring an option-argument is found, but an option-argument is not found,
 88974 this error shall be identified and the invalid option character shall be identified in the
 88975 message.

88976 **OUTPUT FILES**

88977 None.

88978 **EXTENDED DESCRIPTION**

88979 None.

88980 **EXIT STATUS**

88981 The following exit values shall be returned:

88982 `0` An option, specified or unspecified by *optstring*, was found.88983 `>0` The end of options was encountered or an error occurred.88984 **CONSEQUENCES OF ERRORS**

88985 Default.

88986 **APPLICATION USAGE**

88987 Since *getopts* affects the current shell execution environment, it is generally provided as a shell
 88988 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
 88989 of the following:

88990 `(getopts abc value "$@")`88991 `nohup getopts ...`88992 `find . -exec getopts ... \;`

88993 it does not affect the shell variables in the caller's environment.

88994 Note that shell functions share *OPTIND* with the calling shell even though the positional
 88995 parameters are changed. If the calling shell and any of its functions uses *getopts* to parse
 88996 arguments, the results are unspecified.

88997 **EXAMPLES**

88998 The following example script parses and displays its arguments:

88999 `aflag=`89000 `bflag=`89001 `while getopts ab: name`89002 `do`

```

89003     case $name in
89004     a)     aflag=1;;
89005     b)     bflag=1
89006           bval="$OPTARG";;
89007     ?)    printf "Usage: %s: [-a] [-b value] args\n" $0
89008           exit 2;;
89009     esac
89010 done
89011 if [ ! -z "$aflag" ]; then
89012     printf "Option -a specified\n"
89013 fi
89014 if [ ! -z "$bflag" ]; then
89015     printf 'Option -b "%s" specified\n' "$bval"
89016 fi
89017 shift $(( $OPTIND - 1 ))
89018 printf "Remaining arguments are: %s\n" "$*"

```

RATIONALE

89019 The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles
 89020 option-arguments containing <blank>s.
 89021

89022 The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it
 89023 does not affect the execution of *getopts*; it is one of the few "output-only" variables used by the
 89024 standard utilities.

89025 The colon is not allowed as an option character because that is not historical behavior, and it
 89026 violates the Utility Syntax Guidelines. The colon is now specified to behave as in the KornShell
 89027 version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables
 89028 diagnostics concerning missing option-arguments and unexpected option characters. This
 89029 replaces the use of the *OPTERR* variable that was specified in an early proposal.

89030 The formats of the diagnostic messages produced by the *getopts* utility and the *getopt()* function
 89031 are not fully specified because implementations with superior ("friendlier") formats objected to
 89032 the formats used by some historical implementations. The standard developers considered it
 89033 important that the information in the messages used be uniform between *getopts* and *getopt()*.
 89034 Exact duplication of the messages might not be possible, particularly if a utility is built on
 89035 another system that has a different *getopt()* function, but the messages must have specific
 89036 information included so that the program name, invalid option character, and type of error can
 89037 be distinguished by a user.

89038 Only a rare application program intercepts a *getopts* standard error message and wants to parse
 89039 it. Therefore, implementations are free to choose the most usable messages they can devise. The
 89040 following formats are used by many historical implementations:

```

89041 "%s: illegal option -- %c\n", <program name>, <option character>
89042 "%s: option requires an argument -- %c\n", <program name>, \
89043     <option character>

```

89044 Historical shells with built-in versions of *getopt()* or *getopts* have used different formats,
 89045 frequently not even indicating the option character found in error.

FUTURE DIRECTIONS

89046 None.
 89047

89048

SEE ALSO

89049

[Section 2.5.2](#) (on page 2250)

89050

[XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

89051

[XSH *getopt*\(\)](#)

89052

CHANGE HISTORY

89053

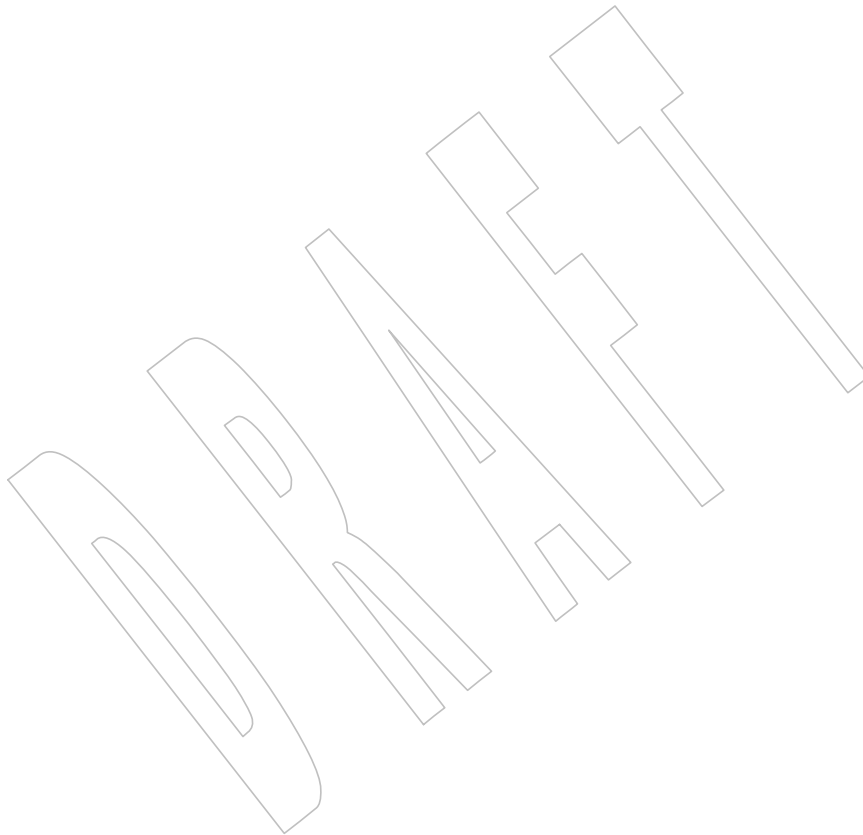
First released in Issue 4.

89054

Issue 6

89055

The normative text is reworded to avoid use of the term “must” for application requirements.



89056 **NAME**89057 `grep` — search a file for a pattern89058 **SYNOPSIS**89059 `grep [-E|-F] [-c|-l|-q] [-insvx] -e pattern_list`
89060 `[-e pattern_list...] [-f pattern_file]... [file...]`89061 `grep [-E|-F] [-c|-l|-q] [-insvx] [-e pattern_list]...`
89062 `-f pattern_file [-f pattern_file]... [file...]`89063 `grep [-E|-F] [-c|-l|-q] [-insvx] pattern_list [file...]`89064 **DESCRIPTION**89065 The *grep* utility shall search the input files, selecting lines matching one or more patterns; the
89066 types of patterns are controlled by the options specified. The patterns are specified by the `-e`
89067 option, `-f` option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more
89068 patterns separated by <newline>s; the *pattern_file*'s contents shall consist of one or more patterns
89069 terminated by <newline>. By default, an input line shall be selected if any pattern, treated as an
89070 entire basic regular expression (BRE) as described in XBD Section 9.3 (on page 169), matches any
89071 part of the line excluding the terminating <newline>; a null BRE shall match every line. By
89072 default, each selected input line shall be written to the standard output.89073 Regular expression matching shall be based on text lines. Since a <newline> separates or
89074 terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a
89075 <newline>. Similarly, since patterns are matched against individual lines (excluding the
89076 terminating <newline>s) of the input, there is no way for a pattern to match a <newline> found
89077 in the input.89078 **OPTIONS**89079 The *grep* utility shall conform to XBD Section 12.2 (on page 201).

89080 The following options shall be supported:

89081 `-E` Match using extended regular expressions. Treat each pattern specified as an ERE,
89082 as described in XBD Section 9.4 (on page 174). If any entire ERE pattern matches
89083 some part of an input line excluding the terminating <newline>, the line shall be
89084 matched. A null ERE shall match every line.89085 `-F` Match using fixed strings. Treat each pattern specified as a string instead of a
89086 regular expression. If an input line contains any of the patterns as a contiguous
89087 sequence of bytes, the line shall be matched. A null string shall match every line.89088 `-c` Write only a count of selected lines to standard output.89089 `-e pattern_list`89090 Specify one or more patterns to be used during the search for input. The
89091 application shall ensure that patterns in *pattern_list* are separated by a <newline>.
89092 A null pattern can be specified by two adjacent <newline>s in *pattern_list*. Unless
89093 the `-E` or `-F` option is also specified, each pattern shall be treated as a BRE, as
89094 described in XBD Section 9.3 (on page 169). Multiple `-e` and `-f` options shall be
89095 accepted by the *grep* utility. All of the specified patterns shall be used when
89096 matching lines, but the order of evaluation is unspecified.89097 `-f pattern_file`89098 Read one or more patterns from the file named by the pathname *pattern_file*.
89099 Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be
89100 specified by an empty line in *pattern_file*. Unless the `-E` or `-F` option is also
89101 specified, each pattern shall be treated as a BRE, as described in XBD Section 9.3

- 89102 (on page 169).
- 89103 **-i** Perform pattern matching in searches without regard to case; see XBD Section 9.2
89104 (on page 168).
- 89105 **-l** (The letter ell.) Write only the names of files containing selected lines to standard
89106 output. Pathnames shall be written once per file searched. If the standard input is
89107 searched, a pathname of "(standard input)" shall be written, in the POSIX
89108 locale. In other locales, "standard input" may be replaced by something more
89109 appropriate in those locales.
- 89110 **-n** Precede each output line by its relative line number in the file, each file starting at
89111 line 1. The line number counter shall be reset for each file processed.
- 89112 **-q** Quiet. Nothing shall be written to the standard output, regardless of matching
89113 lines. Exit with zero status if an input line is selected.
- 89114 **-s** Suppress the error messages ordinarily written for nonexistent or unreadable files.
89115 Other error messages shall not be suppressed.
- 89116 **-v** Select lines not matching any of the specified patterns. If the **-v** option is not
89117 specified, selected lines shall be those that match any of the specified patterns.
- 89118 **-x** Consider only input lines that use all characters in the line excluding the
89119 terminating <newline> to match an entire fixed string or regular expression to be
89120 matching lines.

OPERANDS

89121 The following operands shall be supported:

- 89123 *pattern_list* Specify one or more patterns to be used during the search for input. This operand
89124 shall be treated as if it were specified as **-e pattern_list**.
- 89125 *file* A pathname of a file to be searched for the patterns. If no *file* operands are
89126 specified, the standard input shall be used.

STDIN

89127 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
89128 section.
89129

INPUT FILES

89130 The input files shall be text files.
89131

ENVIRONMENT VARIABLES

89132 The following environment variables shall affect the execution of *grep*:

- 89134 **LANG** Provide a default value for the internationalization variables that are unset or null.
89135 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
89136 variables used to determine the values of locale categories.)
- 89137 **LC_ALL** If set to a non-empty string value, override the values of all the other
89138 internationalization variables.
- 89139 **LC_COLLATE**
89140 Determine the locale for the behavior of ranges, equivalence classes, and multi-
89141 character collating elements within regular expressions.
- 89142 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
89143 characters (for example, single-byte as opposed to multi-byte characters in
89144 arguments and input files) and the behavior of character classes within regular
89145 expressions.

89146 *LC_MESSAGES*
 89147 Determine the locale that should be used to affect the format and contents of
 89148 diagnostic messages written to standard error.

89149 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

89150 Default.
 89151

STDOUT

89152 If the *-l* option is in effect, the following shall be written for each file containing at least one
 89153 selected input line:
 89154

89155 *%s\n*, *<file>*

89156 Otherwise, if more than one *file* argument appears, and *-q* is not specified, the *grep* utility shall
 89157 prefix each output line by:

89158 *%s:*, *<file>*

89159 The remainder of each output line shall depend on the other options specified:

- If the *-c* option is in effect, the remainder of each output line shall contain:

89161 *%d\n*, *<count>*

- Otherwise, if *-c* is not in effect and the *-n* option is in effect, the following shall be written to standard output:

89164 *%d:*, *<line number>*

- Finally, the following shall be written to standard output:

89166 *%s*, *<selected-line contents>*

STDERR

89167 The standard error shall be used only for diagnostic messages.
 89168

OUTPUT FILES

89169 None.
 89170

EXTENDED DESCRIPTION

89171 None.
 89172

EXIT STATUS

89173 The following exit values shall be returned:
 89174

89175 0 One or more lines were selected.

89176 1 No lines were selected.

89177 >1 An error occurred.

CONSEQUENCES OF ERRORS

89178 If the *-q* option is specified, the exit status shall be zero if an input line is selected, even if an
 89179 error was detected. Otherwise, default actions shall be performed.
 89180

APPLICATION USAGE

Care should be taken when using characters in *pattern_list* that may also be meaningful to the command interpreter. It is safest to enclose the entire *pattern_list* argument in single quotes:

```
' . . . '
```

The `-e pattern_list` option has the same effect as the *pattern_list* operand, but is useful when *pattern_list* begins with the hyphen delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Multiple `-e` and `-f` options are accepted and *grep* uses all of the patterns it is given while matching input text lines. (Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first, matching every line, and effectively ignore any other patterns.)

The `-q` option provides a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it exits zero if it finds a match even if *grep* detected an access or read error on earlier *file* operands).

EXAMPLES

1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line numbers:

```
grep -i -n posix text.mm
```

2. To find all empty lines in the standard input:

```
grep ^$
```

or:

```
grep -v .
```

3. Both of the following commands print all lines containing strings "abc" or "def" or both:

```
grep -E 'abc|def'
```

```
grep -F 'abc
def'
```

4. Both of the following commands print all lines matching exactly "abc" or "def":

```
grep -E '^abc$|^def$'
```

```
grep -F -x 'abc
def'
```

RATIONALE

This *grep* has been enhanced in an upwards-compatible way to provide the exact functionality of the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard developers to consolidate the three *greps* into a single command.

The old *egrep* and *fgrep* commands are likely to be supported for many years to come as implementation extensions, allowing historical applications to operate unmodified.

Historical implementations usually silently ignored all but one of multiply-specified `-e` and `-f` options, but were not consistent as to which specification was actually used.

The `-b` option was omitted from the OPTIONS section because block numbers are implementation-defined.

- 89224 The System V restriction on using `-` to mean standard input was omitted.
- 89225 A definition of action taken when given a null BRE or ERE is specified. This is an error
89226 condition in some historical implementations.
- 89227 The `-I` option previously indicated that its use was undefined when no files were explicitly
89228 named. This behavior was historical and placed an unnecessary restriction on future
89229 implementations. It has been removed.
- 89230 The historical BSD `grep -s` option practice is easily duplicated by redirecting standard output to
89231 `/dev/null`. The `-s` option required here is from System V.
- 89232 The `-x` option, historically available only with `fgrep`, is available here for all of the non-
89233 obsolescent versions.
- 89234 **FUTURE DIRECTIONS**
- 89235 None.
- 89236 **SEE ALSO**
- 89237 [*sed*](#)
- 89238 XBD [Chapter 8](#) (on page 159), [Chapter 9](#) (on page 167), [Section 12.2](#) (on page 201) +
- 89239 **CHANGE HISTORY**
- 89240 First released in Issue 2.
- 89241 **Issue 6**
- 89242 The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.
- 89243 The normative text is reworded to avoid use of the term “must” for application requirements.
- 89244 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/28 is applied, correcting the examples
89245 using the `grep -F` option which did not match the normative description of the `-F` option.
- 89246 **Issue 7**
- 89247 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
- 89248 SD5-XCU-ERN-98 is applied, updating the STDOUT section.

89249 **NAME**

89250 hash — remember or report utility locations

89251 **SYNOPSIS**89252 hash [*utility*. . .]

89253 hash -r

89254 **DESCRIPTION**

89255 The *hash* utility shall affect the way the current shell environment remembers the locations of
 89256 utilities found as described in [Section 2.9.1.1](#) (on page 2264). Depending on the arguments
 89257 specified, it shall add utility locations to its list of remembered locations or it shall purge the
 89258 contents of the list. When no arguments are specified, it shall report on the contents of the list.

89259 Utilities provided as built-ins to the shell shall not be reported by *hash*.89260 **OPTIONS**89261 The *hash* utility shall conform to XBD [Section 12.2](#) (on page 201).

89262 The following option shall be supported:

89263 -r Forget all previously remembered utility locations.

89264 **OPERANDS**

89265 The following operand shall be supported:

89266 *utility* The name of a utility to be searched for and added to the list of remembered
 89267 locations. If *utility* contains one or more slashes, the results are unspecified.

89268 **STDIN**

89269 Not used.

89270 **INPUT FILES**

89271 None.

89272 **ENVIRONMENT VARIABLES**89273 The following environment variables shall affect the execution of *hash*:

89274 *LANG* Provide a default value for the internationalization variables that are unset or null.
 89275 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 89276 variables used to determine the values of locale categories.)

89277 *LC_ALL* If set to a non-empty string value, override the values of all the other
 89278 internationalization variables.

89279 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 89280 characters (for example, single-byte as opposed to multi-byte characters in
 89281 arguments).

89282 *LC_MESSAGES*

89283 Determine the locale that should be used to affect the format and contents of
 89284 diagnostic messages written to standard error.

89285 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

89286 *PATH* Determine the location of *utility*, as described in XBD [Chapter 8](#) (on page 159).

89287 **ASYNCHRONOUS EVENTS**

89288 Default.

89289 **STDOUT**

89290 The standard output of *hash* shall be used when no arguments are specified. Its format is
 89291 unspecified, but includes the pathname of each utility in the list of remembered locations for the
 89292 current shell environment. This list shall consist of those utilities named in previous *hash*
 89293 invocations that have been invoked, and may contain those invoked and found through the
 89294 normal command search process.

89295 **STDERR**

89296 The standard error shall be used only for diagnostic messages.

89297 **OUTPUT FILES**

89298 None.

89299 **EXTENDED DESCRIPTION**

89300 None.

89301 **EXIT STATUS**

89302 The following exit values shall be returned:

89303 0 Successful completion.

89304 >0 An error occurred.

89305 **CONSEQUENCES OF ERRORS**

89306 Default.

89307 **APPLICATION USAGE**

89308 Since *hash* affects the current shell execution environment, it is always provided as a shell
 89309 regular built-in. If it is called in a separate utility execution environment, such as one of the
 89310 following:

```
89311 nohup hash -r
89312 find . -type f | xargs hash
```

89313 it does not affect the command search process of the caller's environment.

89314 The *hash* utility may be implemented as an alias—for example, *alias -t -*, in which case utilities
 89315 found through normal command search are not listed by the *hash* command.

89316 The effects of *hash -r* can also be achieved portably by resetting the value of *PATH*; in the
 89317 simplest form, this can be:

89318 `PATH= "$PATH"`

89319 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
 89320 performance improvement on a few implementations; normally, the hashing process is included
 89321 by default.

89322 **EXAMPLES**

89323 None.

89324 **RATIONALE**

89325 None.

89326 **FUTURE DIRECTIONS**

89327 None.

hash*Utilities*

89328

SEE ALSO

89329

[Section 2.9.1.1](#) (on page 2264)

89330

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

89331

CHANGE HISTORY

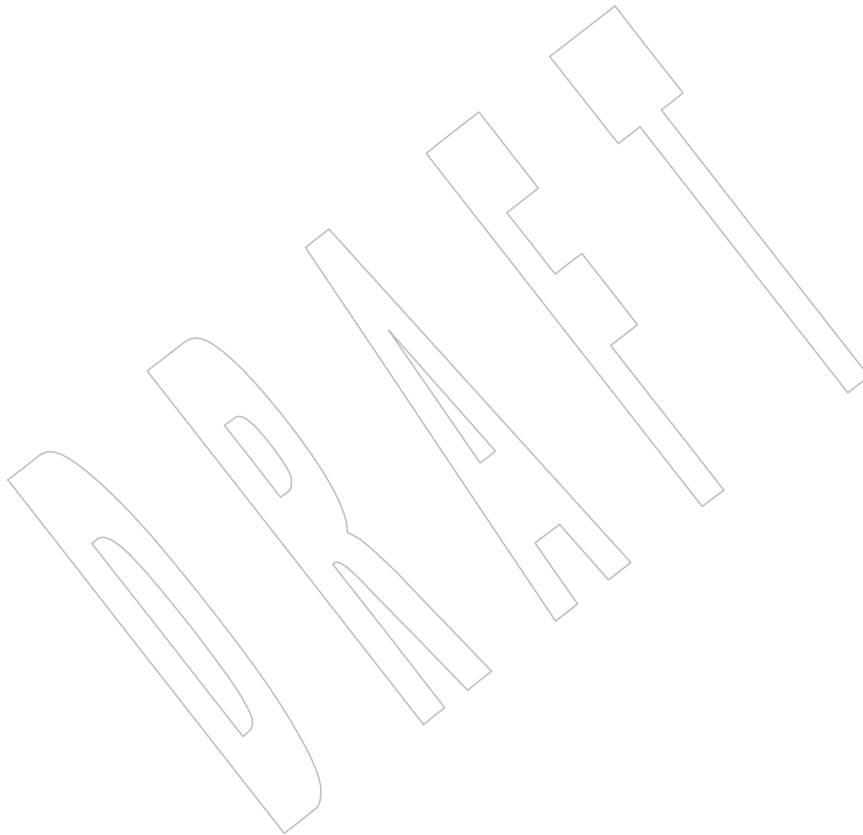
89332

First released in Issue 2.

89333

Issue 7

89334

The *hash* utility is moved from the XSI option to the Base.

89335 **NAME**

89336 head — copy the first part of files

89337 **SYNOPSIS**89338 head [*-n number*] [*file...*]89339 **DESCRIPTION**89340 The *head* utility shall copy its input files to the standard output, ending the output for each file at
89341 a designated point.89342 Copying shall end at the point in each input file indicated by the *-n number* option. The option-
89343 argument *number* shall be counted in units of lines.89344 **OPTIONS**89345 The *head* utility shall conform to XBD [Section 12.2](#) (on page 201).

89346 The following option shall be supported:

89347 *-n number* The first *number* lines of each input file shall be copied to standard output. The
89348 application shall ensure that the *number* option-argument is a positive decimal
89349 integer.89350 When a file contains less than *number* lines, it shall be copied to standard output in its entirety.
89351 This shall not be an error.89352 If no options are specified, *head* shall act as if *-n 10* had been specified.89353 **OPERANDS**

89354 The following operand shall be supported:

89355 *file* A pathname of an input file. If no *file* operands are specified, the standard input
89356 shall be used.89357 **STDIN**89358 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
89359 section.89360 **INPUT FILES**

89361 Input files shall be text files, but the line length is not restricted to {LINE_MAX} bytes.

89362 **ENVIRONMENT VARIABLES**89363 The following environment variables shall affect the execution of *head*:89364 *LANG* Provide a default value for the internationalization variables that are unset or null.
89365 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
89366 variables used to determine the values of locale categories.)89367 *LC_ALL* If set to a non-empty string value, override the values of all the other
89368 internationalization variables.89369 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
89370 characters (for example, single-byte as opposed to multi-byte characters in
89371 arguments and input files).89372 *LC_MESSAGES*89373 Determine the locale that should be used to affect the format and contents of
89374 diagnostic messages written to standard error.89375 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

89376 **ASYNCHRONOUS EVENTS**

89377 Default.

89378 **STDOUT**

89379 The standard output shall contain designated portions of the input files.

89380 If multiple *file* operands are specified, *head* shall precede the output for each with the header:89381 "`\n==> %s <==\n`", *<pathname>*89382 except that the first header written shall not include the initial `<newline>`.89383 **STDERR**

89384 The standard error shall be used only for diagnostic messages.

89385 **OUTPUT FILES**

89386 None.

89387 **EXTENDED DESCRIPTION**

89388 None.

89389 **EXIT STATUS**

89390 The following exit values shall be returned:

89391 0 Successful completion.

89392 >0 An error occurred.

89393 **CONSEQUENCES OF ERRORS**

89394 Default.

89395 **APPLICATION USAGE**

89396 None.

89397 **EXAMPLES**

89398 To write the first ten lines of all files (except those with a leading period) in the directory:

89399 `head -- *`89400 **RATIONALE**89401 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers
89402 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
89403 *tail*.89404 POSIX.1-200x version of *head* follows the Utility Syntax Guidelines. The `-n` option was added to
89405 this new interface so that *head* and *tail* would be more logically related. Earlier versions of this
89406 standard allowed a `-number` option. This form is no longer specified by POSIX.1-200x but may
89407 be present in some implementations.89408 There is no `-c` option (as there is in *tail*) because it is not historical practice and because other
89409 utilities in this volume of POSIX.1-200x provide similar functionality.89410 **FUTURE DIRECTIONS**

89411 None.

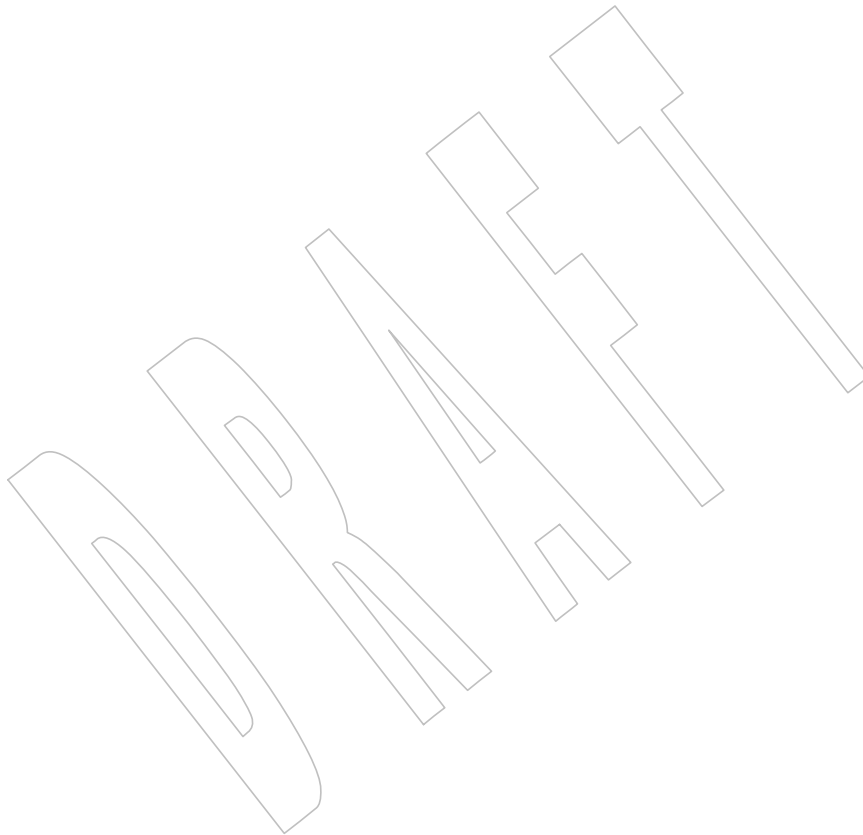
89412 **SEE ALSO**89413 *sed*, *tail*

89414 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

89415 **CHANGE HISTORY**

89416 First released in Issue 4.

89417	Issue 6		
89418		The obsolescent –number form is removed.	
89419		The normative text is reworded to avoid use of the term “must” for application requirements.	
89420		The DESCRIPTION is updated to clarify that when a file contains less than the number of lines requested, the entire file is copied to standard output.	
89421			
89422	Issue 7		
89423		Austin Group Interpretation 1003.1-2001 #027 is applied.	
89424		SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.	
89425		The APPLICATION USAGE section is removed and the EXAMPLES section is corrected.	+



89426 **NAME**

89427 iconv — codeset conversion

89428 **SYNOPSIS**89429 iconv [-cs] -f *frommap* -t *tomap* [*file...*]89430 iconv -f *fromcode* [-cs] [-t *tocode*] [*file...*]89431 iconv -t *tocode* [-cs] [-f *fromcode*] [*file...*]

89432 iconv -l

89433 **DESCRIPTION**89434 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and
89435 write the results to standard output.89436 When the options indicate that charmap files are used to specify the codesets (see **OPTIONS**),
89437 the codeset conversion shall be accomplished by performing a logical join on the symbolic
89438 character names in the two charmaps. The implementation need not support the use of charmap
89439 files for codeset conversion unless the POSIX2_LOCALEDEF symbol is defined on the system.89440 **OPTIONS**89441 The *iconv* utility shall conform to XBD [Section 12.2](#) (on page 201).

89442 The following options shall be supported:

89443 **-c** Omit any characters that are invalid in the codeset of the input file from the
89444 output. When **-c** is not used, the results of encountering invalid characters in the
89445 input stream (either those that are not characters in the codeset of the input file or
89446 that have no corresponding character in the codeset of the output file) shall be
89447 specified in the system documentation. The presence or absence of **-c** shall not
89448 affect the exit status of *iconv*.89449 **-f** *fromcodeset*89450 Identify the codeset of the input file. The implementation shall recognize the
89451 following two forms of the *fromcodeset* option-argument:89452 *fromcode* The *fromcode* option-argument must not contain a slash character. It
89453 shall be interpreted as the name of one of the codeset descriptions
89454 provided by the implementation in an unspecified format. Valid
89455 values of *fromcode* are implementation-defined.89456 *frommap* The *frommap* option-argument must contain a slash character. It shall
89457 be interpreted as the pathname of a charmap file as defined in XBD
89458 [Section 6.4](#) (on page 115). If the pathname does not represent a valid,
89459 readable charmap file, the results are undefined.

89460 If this option is omitted, the codeset of the current locale shall be used.

89461 **-l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified
89462 format.89463 **-s** Suppress any messages written to standard error concerning invalid characters.
89464 When **-s** is not used, the results of encountering invalid characters in the input
89465 stream (either those that are not valid characters in the codeset of the input file or
89466 that have no corresponding character in the codeset of the output file) shall be
89467 specified in the system documentation. The presence or absence of **-s** shall not
89468 affect the exit status of *iconv*.

- 89469 **-t tocodeset** Identify the codeset to be used for the output file. The implementation shall
89470 recognize the following two forms of the *tocodeset* option-argument:
- 89471 *tocode* The semantics shall be equivalent to the *-f fromcode* option.
- 89472 *tomap* The semantics shall be equivalent to the *tomap* option.
- 89473 If this option is omitted, the codeset of the current locale shall be used.
- 89474 If either *-f* or *-t* represents a charmap file, but the other does not (or is omitted), or both *-f* and
89475 *-t* are omitted, the results are undefined.

OPERANDS

89476 The following operand shall be supported:

- 89478 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
89479 *'-'*, the standard input shall be used.

STDIN

89480 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.
89481

INPUT FILES

89482 The input file shall be a text file.
89483

ENVIRONMENT VARIABLES

89484 The following environment variables shall affect the execution of *iconv*:

- 89486 **LANG** Provide a default value for the internationalization variables that are unset or null. |
89487 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
89488 variables used to determine the values of locale categories.)
- 89489 **LC_ALL** If set to a non-empty string value, override the values of all the other
89490 internationalization variables.
- 89491 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
89492 characters (for example, single-byte as opposed to multi-byte characters in
89493 arguments). During translation of the file, this variable is superseded by the use of
89494 the *fromcode* option-argument.
- 89495 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
89496 diagnostic messages written to standard error.
89497
- 89498 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

89499 Default.
89500

STDOUT

89501 When the *-l* option is used, the standard output shall contain all supported *fromcode* and *tocode*
89502 values, written in an unspecified format.
89503

89504 When the *-l* option is not used, the standard output shall contain the sequence of characters
89505 read from the input files, translated to the specified codeset. Nothing else shall be written to the
89506 standard output.

STDERR

89507 The standard error shall be used only for diagnostic messages.
89508

OUTPUT FILES

89509 None.
89510

89511 **EXTENDED DESCRIPTION**

89512 None.

89513 **EXIT STATUS**

89514 The following exit values shall be returned:

89515 0 Successful completion.

89516 >0 An error occurred.

89517 **CONSEQUENCES OF ERRORS**

89518 Default.

89519 **APPLICATION USAGE**89520 The user must ensure that both charmap files use the same symbolic names for characters the
89521 two codesets have in common.89522 **EXAMPLES**89523 The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:2001
89524 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file
89525 **mail.local**:89526 `iconv -f IS6937 -t IS8859 mail.x400 > mail.local`89527 **RATIONALE**89528 The *iconv* utility can be used portably only when the user provides two charmap files as option-
89529 arguments. This is because a single charmap provided by the user cannot reliably be joined with
89530 the names in a system-provided character set description. The valid values for *fromcode* and
89531 *tocode* are implementation-defined and do not have to have any relation to the charmap
89532 mechanisms. As an aid to interactive users, the **-I** option was adopted from the Plan 9 operating
89533 system. It writes information concerning these implementation-defined values. The format is
89534 unspecified because there are many possible useful formats that could be chosen, such as a
89535 matrix of valid combinations of *fromcode* and *tocode*. The **-I** option is not intended for shell script
89536 usage; conforming applications will have to use charmaps.89537 **FUTURE DIRECTIONS**

89538 None.

89539 **SEE ALSO**89540 *gencat*89541 XBD [Section 6.4](#) (on page 115), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +89542 **CHANGE HISTORY**

89543 First released in Issue 3.

89544 **Issue 6**89545 This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the
89546 ability to use charmap files for conversion has been added.89547 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/29 is applied, making changes to address
89548 inconsistencies with the *iconv()* function in the System Interfaces volume of POSIX.1-200x.89549 **Issue 7**

89550 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89551 **NAME**89552 `id` — return user identity89553 **SYNOPSIS**89554 `id` [*user*]89555 `id -G` [*-n*] [*user*]89556 `id -g` [*-nr*] [*user*]89557 `id -u` [*-nr*] [*user*]89558 **DESCRIPTION**

89559 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the
 89560 corresponding user and group names of the invoking process to standard output. If the effective
 89561 and real IDs do not match, both shall be written. If multiple groups are supported by the
 89562 underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of
 89563 POSIX.1-200x), the supplementary group affiliations of the invoking process shall also be
 89564 written.

89565 If a *user* operand is provided and the process has the appropriate privileges, the user and group
 89566 IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be
 89567 identical to real IDs. If the selected user has more than one allowable group membership listed
 89568 in the group database, these shall be written in the same manner as the supplementary groups
 89569 described in the preceding paragraph.

89570 **OPTIONS**89571 The *id* utility shall conform to XBD [Section 12.2](#) (on page 201).

89572 The following options shall be supported:

89573 **-G** Output all different group IDs (effective, real, and supplementary) only, using the
 89574 format "*%u\n*". If there is more than one distinct group affiliation, output each
 89575 such affiliation, using the format " *%u*", before the <newline> is output.

89576 **-g** Output only the effective group ID, using the format "*%u\n*".

89577 **-n** Output the name in the format "*%s*" instead of the numeric ID using the format
 89578 "*%u*".

89579 **-r** Output the real ID instead of the effective ID.

89580 **-u** Output only the effective user ID, using the format "*%u\n*".

89581 **OPERANDS**

89582 The following operand shall be supported:

89583 *user* The login name for which information is to be written.

89584 **STDIN**

89585 Not used.

89586 **INPUT FILES**

89587 None.

89588 **ENVIRONMENT VARIABLES**89589 The following environment variables shall affect the execution of *id*:

89590 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 89591 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 89592 variables used to determine the values of locale categories.)

89593	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
89594		
89595	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
89596		
89597		
89598	<i>LC_MESSAGES</i>	
89599		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
89600		
89601		
89602	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .

ASYNCHRONOUS EVENTS

Default.

STDOUT

The following formats shall be used when the *LC_MESSAGES* locale category specifies the POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with more appropriate strings corresponding to the locale.

```
"uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,
  <real group ID>, <group-name>
```

If the effective and real user IDs do not match, the following shall be inserted immediately before the '\n' character in the previous format:

```
" euid=%u(%s) "
```

with the following arguments added at the end of the argument list:

```
<effective user ID>, <effective user-name>
```

If the effective and real group IDs do not match, the following shall be inserted directly before the '\n' character in the format string (and after any addition resulting from the effective and real user IDs not matching):

```
" egid=%u(%s) "
```

with the following arguments added at the end of the argument list:

```
<effective group-ID>, <effective group name>
```

If the process has supplementary group affiliations or the selected user is allowed to belong to multiple groups, the first shall be added directly before the <newline> in the format string:

```
" groups=%u(%s) "
```

with the following arguments added at the end of the argument list:

```
<supplementary group ID>, <supplementary group name>
```

and the necessary number of the following added after that for any remaining supplementary group IDs:

```
", %u(%s) "
```

and the necessary number of the following arguments added at the end of the argument list:

```
<supplementary group ID>, <supplementary group name>
```

If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple group IDs cannot be mapped by the system into printable user or group names, the corresponding "(%s)" and *name* argument shall be omitted from the corresponding format string.

89636 When any of the options are specified, the output format shall be as described in the OPTIONS
89637 section.

89638 **STDERR**

89639 The standard error shall be used only for diagnostic messages.

89640 **OUTPUT FILES**

89641 None.

89642 **EXTENDED DESCRIPTION**

89643 None.

89644 **EXIT STATUS**

89645 The following exit values shall be returned:

89646 0 Successful completion.

89647 >0 An error occurred.

89648 **CONSEQUENCES OF ERRORS**

89649 Default.

89650 **APPLICATION USAGE**

89651 Output produced by the `-G` option and by the default case could potentially produce very long
89652 lines on systems that support large numbers of supplementary groups. (On systems with user
89653 and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per
89654 name, 93 supplementary groups plus distinct effective and real group and user IDs could
89655 theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case.
89656 It would take about 186 supplementary groups to overflow the 2048-byte barrier using `id -G`.
89657 This is not expected to be a problem in practice, but in cases where it is a concern, applications
89658 should consider using `fold -s` before postprocessing the output of `id`.

89659 **EXAMPLES**

89660 None.

89661 **RATIONALE**

89662 The functionality provided by the 4 BSD `groups` utility can be simulated using:

89663 `id -Gn [user]`

89664 The 4 BSD command `groups` was considered, but it was not included because it did not provide
89665 the functionality of the `id` utility of the SVID. Also, it was thought that it would be easier to
89666 modify `id` to provide the additional functionality necessary to systems with multiple groups than
89667 to invent another command.

89668 The options `-u`, `-g`, `-n`, and `-r` were added to ease the use of `id` with shell commands
89669 substitution. Without these options it is necessary to use some preprocessor such as `sed` to select
89670 the desired piece of information. Since output such as that produced by:

89671 `id -u -n`

89672 is frequently wanted, it seemed desirable to add the options.

89673 **FUTURE DIRECTIONS**

89674 None.

89675 **SEE ALSO**

89676 [fold](#), [logname](#), [who](#)

89677 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

89678 XSH [getgid\(\)](#), [getgroups\(\)](#), [getuid\(\)](#)

89679

CHANGE HISTORY

89680

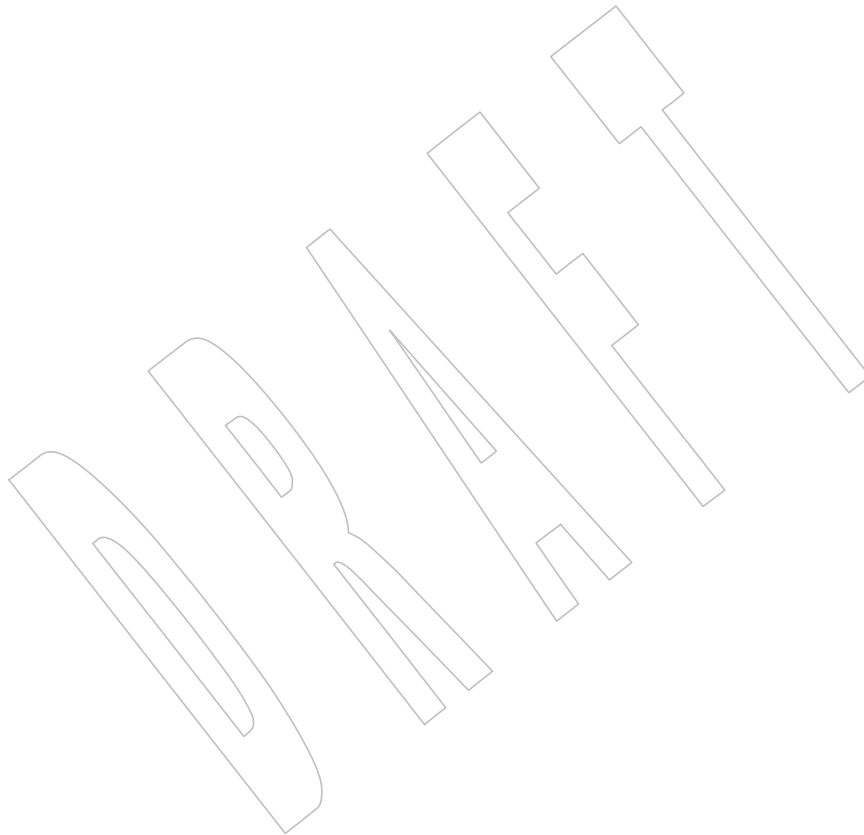
First released in Issue 2.

89681

Issue 7

89682

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



89683 **NAME**
 89684 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

89685 **SYNOPSIS**
 89686 XSI ipcrm [-q msgid|-Q msgkey|-s semid|-S semkey|-m shmid|-M shmkey]...

89687 **DESCRIPTION**
 89688 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory
 89689 segments. The interprocess communication facilities to be removed are specified by the options.

89690 Only a user with appropriate privilege shall be allowed to remove an interprocess
 89691 communication facility that was not created by or owned by the user invoking *ipcrm*.

89692 **OPTIONS**
 89693 The *ipcrm* utility shall conform to XBD [Section 12.2](#) (on page 201).

89694 The following options shall be supported:

89695 **-q msgid** Remove the message queue identifier *msgid* from the system and destroy the
 89696 message queue and data structure associated with it.

89697 **-m shmid** Remove the shared memory identifier *shmid* from the system. The shared memory
 89698 segment and data structure associated with it shall be destroyed after the last
 89699 detach.

89700 **-s semid** Remove the semaphore identifier *semid* from the system and destroy the set of
 89701 semaphores and data structure associated with it.

89702 **-Q msgkey** Remove the message queue identifier, created with key *msgkey*, from the system
 89703 and destroy the message queue and data structure associated with it.

89704 **-M shmkey** Remove the shared memory identifier, created with key *shmkey*, from the system.
 89705 The shared memory segment and data structure associated with it shall be
 89706 destroyed after the last detach.

89707 **-S semkey** Remove the semaphore identifier, created with key *semkey*, from the system and
 89708 destroy the set of semaphores and data structure associated with it.

89709 **OPERANDS**
 89710 None.

89711 **STDIN**
 89712 Not used.

89713 **INPUT FILES**
 89714 None.

89715 **ENVIRONMENT VARIABLES**
 89716 The following environment variables shall affect the execution of *ipcrm*:

89717 **LANG** Provide a default value for the internationalization variables that are unset or null.
 89718 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 89719 variables used to determine the values of locale categories.)

89720 **LC_ALL** If set to a non-empty string value, override the values of all the other
 89721 internationalization variables.

89722 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 89723 characters (for example, single-byte as opposed to multi-byte characters in
 89724 arguments).

- 89725 *LC_MESSAGES*
- 89726 Determine the locale that should be used to affect the format and contents of
89727 diagnostic messages written to standard error.
- 89728 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 89729 **ASYNCHRONOUS EVENTS**
- 89730 Default.
- 89731 **STDOUT**
- 89732 Not used.
- 89733 **STDERR**
- 89734 The standard error shall be used only for diagnostic messages.
- 89735 **OUTPUT FILES**
- 89736 None.
- 89737 **EXTENDED DESCRIPTION**
- 89738 None.
- 89739 **EXIT STATUS**
- 89740 The following exit values shall be returned:
- 89741 0 Successful completion.
- 89742 >0 An error occurred.
- 89743 **CONSEQUENCES OF ERRORS**
- 89744 Default.
- 89745 **APPLICATION USAGE**
- 89746 None.
- 89747 **EXAMPLES**
- 89748 None.
- 89749 **RATIONALE**
- 89750 None.
- 89751 **FUTURE DIRECTIONS**
- 89752 None.
- 89753 **SEE ALSO**
- 89754 *ipcs*
- 89755 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)
- 89756 XSH *msgctl()*, *semctl()*, *shmctl()*
- 89757 **CHANGE HISTORY**
- 89758 First released in Issue 5.
- 89759 **Issue 7**
- 89760 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

89761 **NAME**
 89762 `ipcs` — report XSI interprocess communication facilities status

89763 **SYNOPSIS**
 89764 XSI `ipcs [-qms] [-a|-bcopt]`

89765 **DESCRIPTION**
 89766 The `ipcs` utility shall write information about active interprocess communication facilities.
 89767 Without options, information shall be written in short format for message queues, shared
 89768 memory segments, and semaphore sets that are currently active in the system. Otherwise, the
 89769 information that is displayed is controlled by the options specified.

89770 **OPTIONS**
 89771 The `ipcs` utility shall conform to XBD [Section 12.2](#) (on page 201).

89772 The `ipcs` utility accepts the following options:

89773 `-q` Write information about active message queues.

89774 `-m` Write information about active shared memory segments.

89775 `-s` Write information about active semaphore sets.

89776 If `-q`, `-m`, or `-s` are specified, only information about those facilities shall be written. If none of
 89777 these three are specified, information about all three shall be written subject to the following
 89778 options:

89779 `-a` Use all print options. (This is a shorthand notation for `-b`, `-c`, `-o`, `-p`, and `-t`.)

89780 `-b` Write information on maximum allowable size. (Maximum number of bytes in
 89781 messages on queue for message queues, size of segments for shared memory, and
 89782 number of semaphores in each set for semaphores.)

89783 `-c` Write creator's user name and group name; see below.

89784 `-o` Write information on outstanding usage. (Number of messages on queue and total
 89785 number of bytes in messages on queue for message queues, and number of
 89786 processes attached to shared memory segments.)

89787 `-p` Write process number information. (Process ID of the last process to send a
 89788 message and process ID of the last process to receive a message on message
 89789 queues, process ID of the creating process, and process ID of the last process to
 89790 attach or detach on shared memory segments.)

89791 `-t` Write time information. (Time of the last control operation that changed the access
 89792 permissions for all facilities, time of the last `msgsnd()` and `msgrcv()` operations on
 89793 message queues, time of the last `shmat()` and `shmdt()` operations on shared
 89794 memory, and time of the last `semop()` operation on semaphores.)

89795 **OPERANDS**
 89796 None.

89797 **STDIN**
 89798 Not used.

89799 **INPUT FILES**

- 89800 • The group database
- 89801 • The user database

89802 **ENVIRONMENT VARIABLES**

89803 The following environment variables shall affect the execution of *ipcs*:

89804 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 89805 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 89806 variables used to determine the values of locale categories.)

89807 **LC_ALL** If set to a non-empty string value, override the values of all the other
 89808 internationalization variables.

89809 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 89810 characters (for example, single-byte as opposed to multi-byte characters in
 89811 arguments).

89812 **LC_MESSAGES**
 89813 Determine the locale that should be used to affect the format and contents of
 89814 diagnostic messages written to standard error.

89815 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

89816 **TZ** Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset
 89817 or null, an unspecified default timezone shall be used.

89818 **ASYNCHRONOUS EVENTS**

89819 Default.

89820 **STDOUT**

89821 An introductory line shall be written with the format:

89822 "IPC status from %s as of %s\n", <source>, <date>

89823 where <source> indicates the source used to gather the statistics and <date> is the information
 89824 that would be produced by the *date* command when invoked in the POSIX locale.

89825 The *ipcs* utility then shall create up to three reports depending upon the *-q*, *-m*, and *-s* options.
 89826 The first report shall indicate the status of message queues, the second report shall indicate the
 89827 status of shared memory segments, and the third report shall indicate the status of semaphore
 89828 sets.

89829 If the corresponding facility is not installed or has not been used since the last reboot, then the
 89830 report shall be written out in the format:

89831 "%s facility not in system.\n", <facility>

89832 where <facility> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has
 89833 been installed and has been used since the last reboot, column headings separated by one or
 89834 more spaces and followed by a <newline> shall be written as indicated below followed by the
 89835 facility name written out using the format:

89836 "%s:\n", <facility>

89837 where <facility> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second
 89838 and third reports the column headings need not be written if the last column headings written
 89839 already provide column headings for all information in that report.

89840 The column headings provided in the first column below and the meaning of the information in
 89841 those columns shall be given in order below; the letters in parentheses indicate the options that
 89842 shall cause the corresponding column to appear; "all" means that the column shall always

89843		appear. Each column is separated by one or more <space>s. Note that these options only
89844		determine what information is provided for each report; they do not determine which reports
89845		are written.
89846	T (all)	Type of facility:
89847		q Message queue.
89848		m Shared memory segment.
89849		s Semaphore.
89850		This field is a single character written using the format %c.
89851	ID (all)	The identifier for the facility entry. This field shall be written using the format
89852		%d.
89853	KEY (all)	The key used as an argument to <i>msgget()</i> , <i>semget()</i> , or <i>shmget()</i> to create the
89854		facility entry.
89855		Note: The key of a shared memory segment is changed to IPC_PRIVATE when the
89856		segment has been removed until all processes attached to the segment
89857		detach it.
89858		This field shall be written using the format 0x%x.
89859	MODE (all)	The facility access modes and flags. The mode shall consist of 11 characters
89860		that are interpreted as follows.
89861		The first character shall be:
89862		S If a process is waiting on a <i>msgsnd()</i> operation.
89863		– If the above is not true.
89864		The second character shall be:
89865		R If a process is waiting on a <i>msgrcv()</i> operation.
89866		C or – If the associated shared memory segment is to be cleared when the
89867		first attach operation is executed.
89868		– If none of the above is true.
89869		The next nine characters shall be interpreted as three sets of three bits each.
89870		The first set refers to the owner's permissions; the next to permissions of
89871		others in the usergroup of the facility entry; and the last to all others. Within
89872		each set, the first character indicates permission to read, the second character
89873		indicates permission to write or alter the facility entry, and the last character is
89874		a minus sign ('-').
89875		The permissions shall be indicated as follows:
89876		r If read permission is granted.
89877		w If write permission is granted.
89878		a If alter permission is granted.
89879		– If the indicated permission is not granted.
89880		The first character following the permissions specifies if there is an alternate or
89881		additional access control method associated with the facility. If there is no
89882		alternate or additional access control method associated with the facility, a
89883		single <space> shall be written; otherwise, another printable character is
89884		written.

89885	OWNER (all)	The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the owner shall be written using the format %d.
89886		
89887		
89888		
89889	GROUP (all)	The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the owner shall be written using the format %d.
89890		
89891		
89892		
89893	The following nine columns shall be only written out for message queues:	
89894	CREATOR (a,c)	The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.
89895		
89896		
89897		
89898	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.
89899		
89900		
89901		
89902	CBYTES (a,o)	The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format %d.
89903		
89904	QNUM (a,o)	The number of messages currently outstanding on the associated message queue. This field shall be written using the format %d.
89905		
89906	QBYTES (a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format %d.
89907		
89908	LSPID (a,p)	The process ID of the last process to send a message to the associated queue. This field shall be written using the format:
89909		
89910		"%d", <pid>
89911		where <pid> is 0 if no message has been sent to the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to send a message to the queue.
89912		
89913		
89914	LRPID (a,p)	The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:
89915		
89916		"%d", <pid>
89917		where <pid> is 0 if no message has been received from the corresponding message queue; otherwise, <pid> shall be the process ID of the last process to receive a message from the queue.
89918		
89919		
89920	STIME (a,t)	The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be written.
89921		
89922		
89923		
89924		
89925	RTIME (a,t)	The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format "no-entry" shall be written.
89926		
89927		
89928		
89929		
89930	The following eight columns shall be only written out for shared memory segments.	

89931	CREATOR (a,c)	The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.
89932		
89933		
89934		
89935	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.
89936		
89937		
89938		
89939	NATTCH (a,o)	The number of processes attached to the associated shared memory segment. This field shall be written using the format %d.
89940		
89941	SEGSZ (a,b)	The size of the associated shared memory segment. This field shall be written using the format %d.
89942		
89943	CPID (a,p)	The process ID of the creator of the shared memory entry. This field shall be written using the format %d.
89944		
89945	LPID (a,p)	The process ID of the last process to attach or detach the shared memory segment. This field shall be written using the format:
89946		
89947		"%d", <pid>
89948		where <pid> is 0 if no process has attached the corresponding shared memory segment; otherwise, <pid> shall be the process ID of the last process to attach or detach the segment.
89949		
89950		
89951	ATIME (a,t)	The time the last attach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been attached, the hour, minute, and second of the last time the segment was attached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
89952		
89953		
89954		
89955		
89956	DTIME (a,t)	The time the last detach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been detached, the hour, minute, and second of the last time the segment was detached shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
89957		
89958		
89959		
89960		
89961		The following four columns shall be only written out for semaphore sets:
89962	CREATOR (a,c)	The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the user ID of the creator shall be written using the format %d.
89963		
89964		
89965		
89966	CGROUP (a,c)	The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format %s. Otherwise, the group ID of the creator shall be written using the format %d.
89967		
89968		
89969		
89970	NSEMS (a,b)	The number of semaphores in the set associated with the semaphore entry. This field shall be written using the format %d.
89971		
89972	OTIME (a,t)	The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.
89973		
89974		
89975		
89976		
89977		

89978 The following column shall be written for all three reports when it is requested:

89979 **CTIME (a,t)** The time the associated entry was created or changed. The hour, minute, and
 89980 second of the time when the associated entry was created shall be written
 89981 using the format %d:%2 . 2d:%2 . 2d.

89982 **STDERR**

89983 The standard error shall be used only for diagnostic messages.

89984 **OUTPUT FILES**

89985 None.

89986 **EXTENDED DESCRIPTION**

89987 None.

89988 **EXIT STATUS**

89989 The following exit values shall be returned:

89990 0 Successful completion.

89991 >0 An error occurred.

89992 **CONSEQUENCES OF ERRORS**

89993 Default.

89994 **APPLICATION USAGE**

89995 Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate
 89996 only when it was retrieved.

89997 **EXAMPLES**

89998 None.

89999 **RATIONALE**

90000 None.

90001 **FUTURE DIRECTIONS**

90002 None.

90003 **SEE ALSO**

90004 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

90005 XSH [msgrcv\(\)](#), [msgsnd\(\)](#), [semget\(\)](#), [semop\(\)](#), [shmat\(\)](#), [shmdt\(\)](#), [shmget\(\)](#)

90006 **CHANGE HISTORY**

90007 First released in Issue 5.

90008 **Issue 6**

90009 The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

90010 The Open Group Corrigenda U032/1 and U032/2 are applied, clarifying the output format.

90011 The Open Group Base Resolution bwg98-004 is applied.

90012 **Issue 7**

90013 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

90014 **NAME**
 90015 `jobs` — display status of jobs in the current session

90016 **SYNOPSIS**
 90017 UP `jobs [-l|-p] [job_id...]`

90018 **DESCRIPTION**
 90019 The *jobs* utility shall display the status of jobs that were started in the current shell environment;
 90020 see [Section 2.12](#) (on page 2277).

90021 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the
 90022 list of those “known in the current shell execution environment”; see [Section 2.9.3.1](#) (on page
 90023 2266).

90024 **OPTIONS**
 90025 The *jobs* utility shall conform to XBD [Section 12.2](#) (on page 201).

90026 The following options shall be supported:

90027 `-l` (The letter ell.) Provide more information about each job listed. This information
 90028 shall include the job number, current job, process group ID, state, and the
 90029 command that formed the job.

90030 `-p` Display only the process IDs for the process group leaders of the selected jobs.

90031 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs
 90032 and all jobs whose status has changed and have not been reported by the shell.

90033 **OPERANDS**
 90034 The following operand shall be supported:

90035 *job_id* Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the
 90036 status information for all jobs shall be displayed. The format of *job_id* is described
 90037 in XBD [Section 3.202](#) (on page 61).

90038 **STDIN**
 90039 Not used.

90040 **INPUT FILES**
 90041 None.

90042 **ENVIRONMENT VARIABLES**
 90043 The following environment variables shall affect the execution of *jobs*:

90044 *LANG* Provide a default value for the internationalization variables that are unset or null.
 90045 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 90046 variables used to determine the values of locale categories.)

90047 *LC_ALL* If set to a non-empty string value, override the values of all the other
 90048 internationalization variables.

90049 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 90050 characters (for example, single-byte as opposed to multi-byte characters in
 90051 arguments).

90052 *LC_MESSAGES*
 90053 Determine the locale that should be used to affect the format and contents of
 90054 diagnostic messages written to standard error and informative messages written to
 90055 standard output.

90056 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

90057 ASYNCHRONOUS EVENTS

90058 Default.

90059 STDOUT

90060 If the `-p` option is specified, the output shall consist of one line for each process ID:

90061 "`%d\n`", *<process ID>*

90062 Otherwise, if the `-l` option is not specified, the output shall be a series of lines of the form:

90063 "`[%d] %c %s %s\n`", *<job-number>*, *<current>*, *<state>*, *<command>*

90064 where the fields shall be as follows:

90065 *<current>* The character `'+'` identifies the job that would be used as a default for the *fg* or *bg*
 90066 utilities; this job can also be specified using the *job_id* `%+` or `"%%"`. The character
 90067 `'-'` identifies the job that would become the default if the current default job were
 90068 to exit; this job can also be specified using the *job_id* `%-`. For other jobs, this field is
 90069 a *<space>*. At most one job can be identified with `'+'` and at most one job can be
 90070 identified with `'-'`. If there is any suspended job, then the current job shall be a
 90071 suspended job. If there are at least two suspended jobs, then the previous job also
 90072 shall be a suspended job.

90073 *<job-number>* A number that can be used to identify the process group to the *wait*, *fg*, *bg*, and *kill*
 90074 utilities. Using these utilities, the job can be identified by prefixing the job number
 90075 with `'%'`.

90076 *<state>* One of the following strings (in the POSIX locale):

90077 **Running** Indicates that the job has not been suspended by a signal and has not
 90078 exited.

90079 **Done** Indicates that the job completed and returned exit status zero.

90080 **Done(*code*)** Indicates that the job completed normally and that it exited with the
 90081 specified non-zero exit status, *code*, expressed as a decimal number.

90082 **Stopped** Indicates that the job was suspended by the SIGTSTP signal.

90083 **Stopped (SIGTSTP)**

90084 Indicates that the job was suspended by the SIGTSTP signal.

90085 **Stopped (SIGSTOP)**

90086 Indicates that the job was suspended by the SIGSTOP signal.

90087 **Stopped (SIGTTIN)**

90088 Indicates that the job was suspended by the SIGTTIN signal.

90089 **Stopped (SIGTTOU)**

90090 Indicates that the job was suspended by the SIGTTOU signal.

90091 The implementation may substitute the string **Suspended** in place of **Stopped**. If
 90092 the job was terminated by a signal, the format of *<state>* is unspecified, but it shall
 90093 be visibly distinct from all of the other *<state>* formats shown here and shall
 90094 indicate the name or description of the signal causing the termination.

90095 *<command>* The associated command that was given to the shell.

90096 If the `-l` option is specified, a field containing the process group ID shall be inserted before the
 90097 *<state>* field. Also, more processes in a process group may be output on separate lines, using
 90098 only the process ID and *<command>* fields.

90099 **STDERR**

90100 The standard error shall be used only for diagnostic messages.

90101 **OUTPUT FILES**

90102 None.

90103 **EXTENDED DESCRIPTION**

90104 None.

90105 **EXIT STATUS**

90106 The following exit values shall be returned:

90107 0 Successful completion.

90108 >0 An error occurred.

90109 **CONSEQUENCES OF ERRORS**

90110 Default.

90111 **APPLICATION USAGE**

90112 The `-p` option is the only portable way to find out the process group of a job because different
 90113 implementations have different strategies for defining the process group of the job. Usage such
 90114 as `$(jobs -p)` provides a way of referring to the process group of the job in an implementation-
 90115 independent way.

90116 The `jobs` utility does not work as expected when it is operating in its own utility execution
 90117 environment because that environment has no applicable jobs to manipulate. See the
 90118 APPLICATION USAGE section for `bg`. For this reason, `jobs` is generally implemented as a shell
 90119 regular built-in.

90120 **EXAMPLES**

90121 None.

90122 **RATIONALE**

90123 Both "`%%`" and "`%+`" are used to refer to the current job. Both forms are of equal validity—the
 90124 "`%%`" mirroring "`$$`" and "`%+`" mirroring the output of `jobs`. Both forms reflect historical
 90125 practice of the KornShell and the C shell with job control.

90126 The job control features provided by `bg`, `fg`, and `jobs` are based on the KornShell. The standard
 90127 developers examined the characteristics of the C shell versions of these utilities and found that
 90128 differences exist. Despite widespread use of the C shell, the KornShell versions were selected for
 90129 this volume of POSIX.1-200x to maintain a degree of uniformity with the rest of the KornShell
 90130 features selected (such as the very popular command line editing features).

90131 The `jobs` utility is not dependent on the job control option, as are the seemingly related `bg` and `fg`
 90132 utilities because `jobs` is useful for examining background jobs, regardless of the condition of job
 90133 control. When the user has invoked a `set +m` command and job control has been turned off, `jobs`
 90134 can still be used to examine the background jobs associated with that current session. Similarly,
 90135 `kill` can then be used to kill background jobs with `kill% <background job number>`.

90136 The output for terminated jobs is left unspecified to accommodate various historical systems.
 90137 The following formats have been witnessed:

- 90138 1. **Killed**(*signal name*)
- 90139 2. *signal name*
- 90140 3. *signal name*(**coredump**)
- 90141 4. *signal description*– **core dumped**

90142 Most users should be able to understand these formats, although it means that applications have
 90143 trouble parsing them.

90144 The calculation of job IDs was not described since this would suggest an implementation, which
 90145 may impose unnecessary restrictions.

90146 In an early proposal, a `-n` option was included to “Display the status of jobs that have changed,
 90147 exited, or stopped since the last status report”. It was removed because the shell always writes
 90148 any changed status of jobs before each prompt.

90149 FUTURE DIRECTIONS

90150 None.

90151 SEE ALSO

90152 [Section 2.12](#) (on page 2277), *bg*, *fg*, *kill*, *wait*

90153 XBD [Section 3.202](#) (on page 61), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

90154 CHANGE HISTORY

90155 First released in Issue 4.

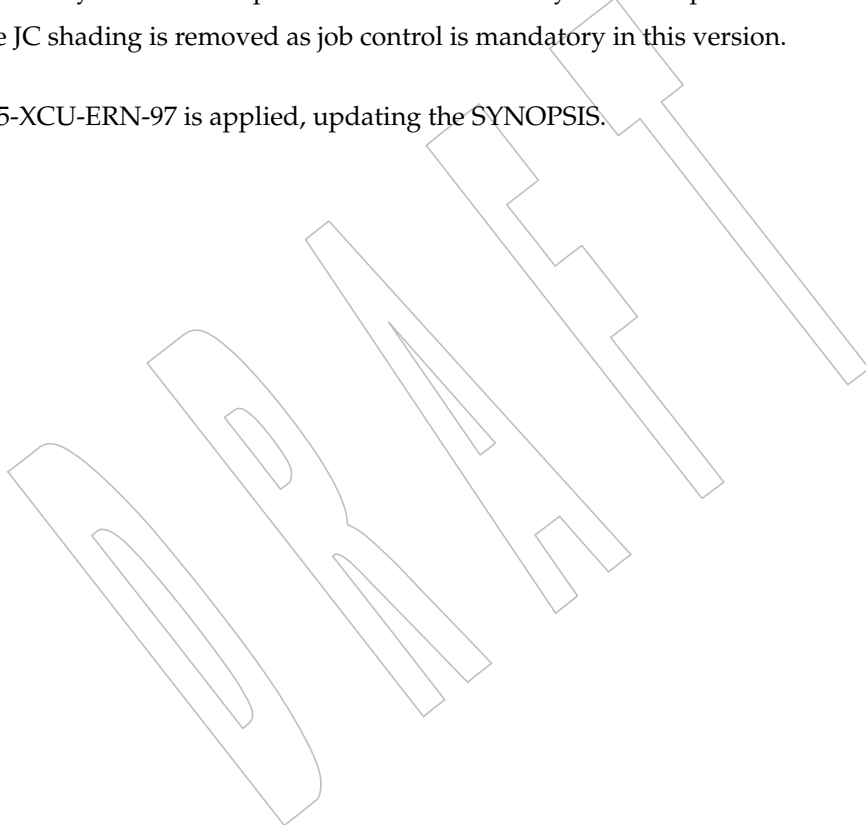
90156 Issue 6

90157 This utility is marked as part of the User Portability Utilities option.

90158 The JC shading is removed as job control is mandatory in this version. |

90159 Issue 7

90160 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



90161 **NAME**
 90162 join — relational database operator

90163 **SYNOPSIS**
 90164 join [-a *file_number*|-v *file_number*] [-e *string*] [-o *list*] [-t *char*]
 90165 [-1 *field*] [-2 *field*] *file1 file2*

90166 **DESCRIPTION**
 90167 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be
 90168 written to the standard output.

90169 The join field is a field in each file on which the files are compared. The *join* utility shall write
 90170 one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The
 90171 output line by default shall consist of the join field, then the remaining fields from *file1*, then the
 90172 remaining fields from *file2*. This format can be changed by using the **-o** option (see below). The
 90173 **-a** option can be used to add unmatched lines to the output. The **-v** option can be used to
 90174 output only unmatched lines.

90175 The files *file1* and *file2* shall be ordered in the collating sequence of *sort -b* on the fields on which
 90176 they shall be joined, by default the first in each line. All selected output shall be written in the
 90177 same collating sequence.

90178 The default input field separators shall be <blank>s. In this case, multiple separators shall count
 90179 as one field separator, and leading separators shall be ignored. The default output field
 90180 separator shall be a <space>.

90181 The field separator and collating sequence can be changed by using the **-t** option (see below).

90182 If the same key appears more than once in either file, all combinations of the set of remaining
 90183 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines
 90184 encountered.

90185 If the input files are not in the appropriate collating sequence, the results are unspecified.

90186 **OPTIONS**
 90187 The *join* utility shall conform to XBD [Section 12.2](#) (on page 201).

90188 The following options shall be supported:

90189 **-a** *file_number*
 90190 Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or
 90191 2, in addition to the default output. If both **-a1** and **-a2** are specified, all unpairable
 90192 lines shall be output.

90193 **-e** *string* Replace empty output fields in the list selected by **-o** with the string *string*.

90194 **-o** *list* Construct the output line to comprise the fields specified in *list*, each element of
 90195 which shall have one of the following two forms:

- 90196 1. *file_number.field*, where *file_number* is a file number and *field* is a decimal
 90197 integer field number
- 90198 2. 0 (zero), representing the join field

90199 The elements of *list* shall be either comma-separated or <blank>-separated, as
 90200 specified in Guideline 8 of XBD [Section 12.2](#) (on page 201). The fields specified by
 90201 *list* shall be written for all selected output lines. Fields selected by *list* that do not
 90202 appear in the input shall be treated as empty output fields. (See the **-e** option.)
 90203 Only specifically requested fields shall be written. The application shall ensure that
 90204 *list* is a single command line argument.

- 90205 **-t** *char* Use character *char* as a separator, for both input and output. Every appearance of
90206 *char* in a line shall be significant. When this option is specified, the collating
90207 sequence shall be the same as *sort* without the **-b** option.
- 90208 **-v** *file_number* Instead of the default output, produce a line only for each unpairable line in
90209 *file_number*, where *file_number* is 1 or 2. If both **-v1** and **-v2** are specified, all
90210 unpairable lines shall be output.
- 90212 **-1** *field* Join on the *field*th field of file 1. Fields are decimal integers starting with 1.
- 90213 **-2** *field* Join on the *field*th field of file 2. Fields are decimal integers starting with 1.

OPERANDS

- 90214 The following operands shall be supported:
- 90215 *file1, file2* A pathname of a file to be joined. If either of the *file1* or *file2* operands is '-', the
90216 standard input shall be used in its place.

STDIN

- 90218 The standard input shall be used only if the *file1* or *file2* operand is '-'. See the INPUT FILES
90219 section.

INPUT FILES

- 90221 The input files shall be text files.

ENVIRONMENT VARIABLES

- 90222 The following environment variables shall affect the execution of *join*:
- 90223 **LANG** Provide a default value for the internationalization variables that are unset or null. |
90224 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
90225 variables used to determine the values of locale categories.)
- 90226 **LC_ALL** If set to a non-empty string value, override the values of all the other
90227 internationalization variables.
- 90228 **LC_COLLATE** Determine the locale of the collating sequence *join* expects to have been used when
90229 the input files were sorted.
- 90230 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
90231 characters (for example, single-byte as opposed to multi-byte characters in
90232 arguments and input files).
- 90233 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
90234 diagnostic messages written to standard error.
- 90235 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
90236 diagnostic messages written to standard error.
- 90237 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
90238 diagnostic messages written to standard error.
- 90239 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

- 90240 Default.

STDOUT

- 90242 The *join* utility output shall be a concatenation of selected character fields. When the **-o** option
90243 is not specified, the output shall be:
90244 "*%s%s%s\n*", *<join field>*, *<other file1 fields>*,
90245 *<other file2 fields>*
- 90246 If the join field is not the first field in a file, the *<other file fields>* for that file shall be:
90247 *<fields preceding join field>*, *<fields following join field>*
90248 *<fields preceding join field>*, *<fields following join field>*

90249 When the `-o` option is specified, the output format shall be:

90250 `"%s\n", <concatenation of fields>`

90251 where the concatenation of fields is described by the `-o` option, above.

90252 For either format, each field (except the last) shall be written with its trailing separator character.
90253 If the separator is the default (`<blank>s`), a single `<space>` shall be written after each field
90254 (except the last).

90255 **STDERR**

90256 The standard error shall be used only for diagnostic messages.

90257 **OUTPUT FILES**

90258 None.

90259 **EXTENDED DESCRIPTION**

90260 None.

90261 **EXIT STATUS**

90262 The following exit values shall be returned:

90263 0 All input files were output successfully.

90264 >0 An error occurred.

90265 **CONSEQUENCES OF ERRORS**

90266 Default.

90267 **APPLICATION USAGE**

90268 Pathnames consisting of numeric digits or of the form `string.string` should not be specified
90269 directly following the `-o` list.

90270 **EXAMPLES**

90271 The `-o 0` field essentially selects the union of the join fields. For example, given file **phone**:

```
90272 !Name           Phone Number
90273 Don             +1 123-456-7890
90274 Hal            +1 234-567-8901
90275 Yasushi        +2 345-678-9012
```

90276 and file **fax**:

```
90277 !Name           Fax Number
90278 Don             +1 123-456-7899
90279 Keith          +1 456-789-0122
90280 Yasushi        +2 345-678-9011
```

90281 (where the large expanses of white space are meant to each represent a single `<tab>`), the
90282 command:

```
90283 join -t "<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

90284 would produce:

```
90285 !Name           Phone Number           Fax Number
90286 Don             +1 123-456-7890         +1 123-456-7899
90287 Hal            +1 234-567-8901         (unknown)
90288 Keith          (unknown)              +1 456-789-0122
90289 Yasushi        +2 345-678-9012         +2 345-678-9011
```

90290 Multiple instances of the same key will produce combinatorial results. The following:

```
90291 fa:
90292     a x
```

```

90293         a y
90294         a z
90295     fb:
90296         a p
90297
90297     will produce:
90298
90298     a x p
90299     a y p
90300     a z p
90301
90301     And the following:
90302
90302     fa:
90303         a b c
90304         a d e
90305     fb:
90306         a w x
90307         a y z
90308         a o p
90309
90309     will produce:
90310
90310     a b c w x
90311     a b c y z
90312     a b c o p
90313     a d e w x
90314     a d e y z
90315     a d e o p

```

RATIONALE

The `-e` option is only effective when used with `-o` because, unless specific fields are identified using `-o`, *join* is not aware of what fields might be empty. The exception to this is the join field, but identifying an empty join field with the `-e` string is not historical practice and some scripts might break if this were changed.

The 0 field in the `-o` list was adopted from the Tenth Edition version of *join* to satisfy international objections that the *join* in the base documents does not support the “full join” or “outer join” described in relational database literature. Although it has been possible to include a join field in the output (by default, or by field number using `-o`), the join field could not be included for an unpaired line selected by `-a`. The `-o 0` field essentially selects the union of the join fields.

This sort of outer join was not possible with the *join* commands in the base documents. The `-o 0` field was chosen because it is an upwards-compatible change for applications. An alternative was considered: have the join field represent the union of the fields in the files (where they are identical for matched lines, and one or both are null for unmatched lines). This was not adopted because it would break some historical applications.

The ability to specify *file2* as `-` is not historical practice; it was added for completeness.

The `-v` option is not historical practice, but was considered necessary because it permitted the writing of *only* those lines that do not match on the join field, as opposed to the `-a` option, which prints both lines that do and do not match. This additional facility is parallel with the `-v` option of *grep*.

Some historical implementations have been encountered where a blank line in one of the input files was considered to be the end of the file; the description in this volume of POSIX.1-200x does not cite this as an allowable case.

Earlier versions of this standard allowed `-j`, `-j1`, `-j2` options, and a form of the `-o` option that

90341 allowed the *list* option-argument to be multiple arguments. These forms are no longer specified |
 90342 by POSIX.1-200x but may be present in some implementations.

90343 FUTURE DIRECTIONS

90344 None.

90345 SEE ALSO

90346 *awk, comm, sort, uniq*

90347 XBD Chapter 8 (on page 159), Section 12.2 (on page 201) +

90348 CHANGE HISTORY

90349 First released in Issue 2.

90350 Issue 6

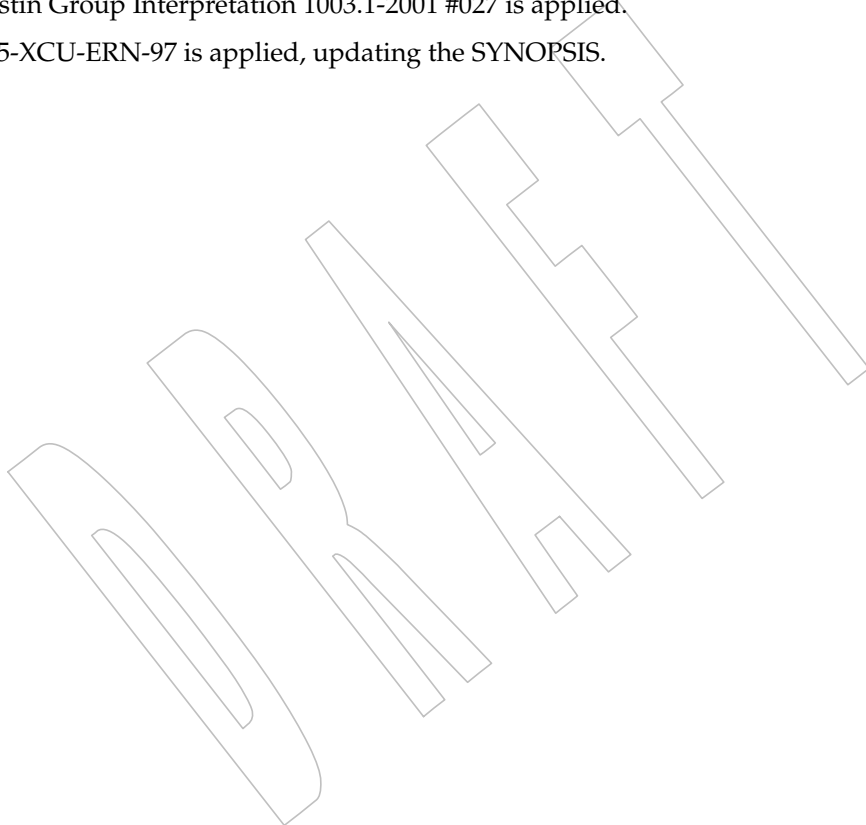
90351 The obsolescent `-j` options and the multi-argument `-o` option are removed in this version. |

90352 The normative text is reworded to avoid use of the term “must” for application requirements.

90353 Issue 7

90354 Austin Group Interpretation 1003.1-2001 #027 is applied.

90355 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



90356 **NAME**
 90357 kill — terminate or signal processes

90358 **SYNOPSIS**
 90359 kill *-s signal_name pid...*
 90360 kill *-l [exit_status]*
 90361 XSI kill *[-signal_name] pid...*
 90362 kill *[-signal_number] pid...*

90363 **DESCRIPTION**
 90364 The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.
 90365 For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill()* function
 90366 defined in the System Interfaces volume of POSIX.1-200x called with the following arguments:

- 90367 • The value of the *pid* operand shall be used as the *pid* argument.
- 90368 • The *sig* argument is the value specified by the *-s* option, *-signal_number* option, or the
 90369 *-signal_name* option, or by SIGTERM, if none of these options is specified.

90370 **OPTIONS**
 90371 XSI The *kill* utility shall conform to XBD [Section 12.2](#) (on page 201), except that in the last two
 90372 SYNOPSIS forms, the *-signal_number* and *-signal_name* options are usually more than a single
 90373 character.

90374 The following options shall be supported:

90375 **-l** (The letter ell.) Write all values of *signal_name* supported by the implementation, if
 90376 no operand is given. If an *exit_status* operand is given and it is a value of the ' ? '
 90377 shell special parameter (see [Section 2.5.2](#) (on page 2250) and *wait*) corresponding to
 90378 a process that was terminated by a signal, the *signal_name* corresponding to the
 90379 signal that terminated the process shall be written. If an *exit_status* operand is
 90380 given and it is the unsigned decimal integer value of a signal number, the
 90381 *signal_name* (the symbolic constant name without the **SIG** prefix defined in the
 90382 Base Definitions volume of POSIX.1-200x) corresponding to that signal shall be
 90383 written. Otherwise, the results are unspecified.

90384 **-s signal_name**
 90385 Specify the signal to send, using one of the symbolic names defined in the
 90386 **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent
 90387 fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be
 90388 recognized, representing the signal value zero. The corresponding signal shall be
 90389 sent instead of SIGTERM.

90390 XSI **-signal_name**
 90391 Equivalent to *-s signal_name*.

90392 XSI **-signal_number**
 90393 Specify a non-negative decimal integer, *signal_number*, representing the signal to be
 90394 used instead of SIGTERM, as the *sig* argument in the effective call to *kill()*. The
 90395 correspondence between integer values and the *sig* value used is shown in the
 90396 following list.
 90397 The effects of specifying any *signal_number* other than those listed below are
 90398 undefined.

90399	0	0
90400	1	SIGHUP
90401	2	SIGINT
90402	3	SIGQUIT
90403	6	SIGABRT
90404	9	SIGKILL
90405	14	SIGALRM
90406	15	SIGTERM

If the first argument is a negative integer, it shall be interpreted as a *-signal_number* option, not as a negative *pid* operand specifying a process group.

OPERANDS

The following operands shall be supported:

pid One of the following:

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative, and zero values of the *pid* operand shall be as described for the *kill()* function. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill()* function defined in the System Interfaces volume of POSIX.1-200x. If the first *pid* operand is negative, it should be preceded by "--" to keep it from being interpreted as an option.
2. A job control job ID (see XBD Section 3.202, on page 61) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of *kill* in the current shell execution environment; see Section 2.12 (on page 2277).

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *kill*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

90443 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

90444 ASYNCHRONOUS EVENTS

90445 Default.

90446 STDOUT

90447 When the *-l* option is not specified, the standard output shall not be used.

90448 When the *-l* option is specified, the symbolic name of each signal shall be written in the
90449 following format:

90450 "%s%c", <signal_name>, <separator>

90451 where the <signal_name> is in uppercase, without the **SIG** prefix, and the <separator> shall be
90452 either a <newline> or a <space>. For the last signal written, <separator> shall be a <newline>.

90453 When both the *-l* option and *exit_status* operand are specified, the symbolic name of the
90454 corresponding signal shall be written in the following format:

90455 "%s\n", <signal_name>

90456 STDERR

90457 The standard error shall be used only for diagnostic messages.

90458 OUTPUT FILES

90459 None.

90460 EXTENDED DESCRIPTION

90461 None.

90462 EXIT STATUS

90463 The following exit values shall be returned:

90464 0 At least one matching process was found for each *pid* operand, and the specified signal was
90465 successfully processed for at least one matching process.

90466 >0 An error occurred.

90467 CONSEQUENCES OF ERRORS

90468 Default.

90469 APPLICATION USAGE

90470 Process numbers can be found by using *ps*.

90471 The job control job ID notation is not required to work as expected when *kill* is operating in its
90472 own utility execution environment. In either of the following examples:

90473 nohup kill %1 &
90474 system("kill %1");

90475 the *kill* operates in a different environment and does not share the shell's understanding of job
90476 numbers.

90477 EXAMPLES

90478 Any of the commands:

90479 kill -9 100 -165
90480 kill -s kill 100 -165
90481 kill -s KILL 100 -165

90482 sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose
90483 process group ID is 165, assuming the sending process has permission to send that signal to the
90484 specified processes, and that they exist.

90485 The System Interfaces volume of POSIX.1-200x and this volume of POSIX.1-200x do not require

90486 specific signal numbers for any *signal_names*. Even the *-signal_number* option provides symbolic
 90487 (although numeric) names for signals. If a process is terminated by a signal, its exit status
 90488 indicates the signal that killed it, but the exact values are not specified. The *kill -l* option,
 90489 however, can be used to map decimal signal numbers and exit status values into the name of a
 90490 signal. The following example reports the status of a terminated job:

```
90491 job
90492 stat=$?
90493 if [ $stat -eq 0 ]
90494 then
90495     echo job completed successfully.
90496 elif [ $stat -gt 128 ]
90497 then
90498     echo job terminated by signal SIG$(kill -l $stat).
90499 else
90500     echo job terminated with error code $stat.
90501 fi
```

90502 To send the default signal to a process group (say 123), an application should use a command
 90503 similar to one of the following:

```
90504 kill -TERM -123
90505 kill -- -123
```

90506 RATIONALE

90507 The *-l* option originated from the C shell, and is also implemented in the KornShell. The C shell
 90508 output can consist of multiple output lines because the signal names do not always fit on a
 90509 single line on some terminal screens. The KornShell output also included the implementation-
 90510 defined signal numbers and was considered by the standard developers to be too difficult for
 90511 scripts to parse conveniently. The specified output format is intended not only to accommodate
 90512 the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing
 90513 on systems for which this is appropriate.

90514 An early proposal invented the name *SIGNULL* as a *signal_name* for signal 0 (used by the System
 90515 Interfaces volume of POSIX.1-200x to test for the existence of a process without sending it a
 90516 signal). Since the *signal_name* 0 can be used in this case unambiguously, *SIGNULL* has been
 90517 removed.

90518 An early proposal also required symbolic *signal_names* to be recognized with or without the **SIG**
 90519 prefix. Historical versions of *kill* have not written the **SIG** prefix for the *-l* option and have not
 90520 recognized the **SIG** prefix on *signal_names*. Since neither applications portability nor ease-of-use
 90521 would be improved by requiring this extension, it is no longer required.

90522 To avoid an ambiguity of an initial negative number argument specifying either a signal number
 90523 or a process group, POSIX.1-200x mandates that it is always considered the former by
 90524 implementations that support the XSI option. It also requires that conforming applications
 90525 always use the "--" options terminator argument when specifying a process group, unless an
 90526 option is also specified.

90527 The *-s* option was added in response to international interest in providing some form of *kill* that
 90528 meets the Utility Syntax Guidelines.

90529 The job control job ID notation is not required to work as expected when *kill* is operating in its
 90530 own utility execution environment. In either of the following examples:

```
90531 nohup kill %1 &
90532 system("kill %1");
```

90533 the *kill* operates in a different environment and does not understand how the shell has managed
 90534 its job numbers.

90535

FUTURE DIRECTIONS

90536

None.

90537

SEE ALSO

90538

[Chapter 2](#) (on page 2245), *ps*, *wait*

90539

[XBD Section 3.202](#) (on page 61), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [<signal.h>](#) |

90540

XSH *kill*

90541

CHANGE HISTORY

90542

First released in Issue 2.

90543

Issue 6

90544

The obsolescent versions of the SYNOPSIS are turned into non-obsolescent features of the XSI option, corresponding to a similar change in the *trap* special built-in.

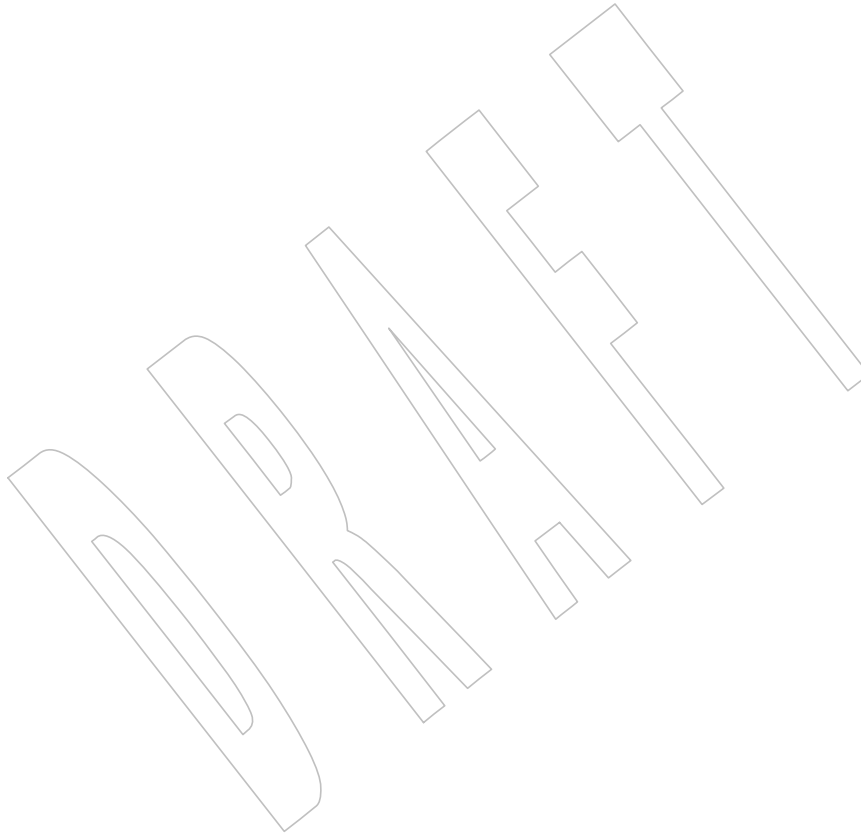
90545

90546

Issue 7

90547

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



90548 **NAME**90549 lex — generate programs for lexical tasks (**DEVELOPMENT**)90550 **SYNOPSIS**90551 CD `lex [-t] [-n|-v] [file...]`90552 **DESCRIPTION**

90553 The *lex* utility shall generate C programs to be used in lexical processing of character input, and
 90554 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code
 90555 and conform to the ISO C standard. Usually, the *lex* utility shall write the program it generates to
 90556 the file **lex.yy.c**; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the
 90557 EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

90558 **OPTIONS**90559 The *lex* utility shall conform to XBD [Section 12.2](#) (on page 201), except for Guideline 9. |

90560 The following options shall be supported:

90561 **-n** Suppress the summary of statistics usually written with the **-v** option. If no table
 90562 sizes are specified in the *lex* source code and the **-v** option is not specified, then **-n**
 90563 is implied.

90564 **-t** Write the resulting program to standard output instead of **lex.yy.c**.

90565 **-v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*
 90566 table sizes in [Definitions in lex](#) (on page 2753).) If the **-t** option is specified and **-n**
 90567 is not specified, this report shall be written to standard error. If table sizes are
 90568 specified in the *lex* source code, and if the **-n** option is not specified, the **-v** option
 90569 may be enabled.

90570 **OPERANDS**

90571 The following operand shall be supported:

90572 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be
 90573 concatenated to produce a single *lex* program. If no *file* operands are specified, or if
 90574 a *file* operand is '-', the standard input shall be used.

90575 **STDIN**

90576 The standard input shall be used if no *file* operands are specified, or if a *file* operand is '-'. See
 90577 INPUT FILES.

90578 **INPUT FILES**

90579 The input files shall be text files containing *lex* source code, as described in the EXTENDED
 90580 DESCRIPTION section.

90581 **ENVIRONMENT VARIABLES**90582 The following environment variables shall affect the execution of *lex*:

90583 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 90584 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 90585 variables used to determine the values of locale categories.)

90586 **LC_ALL** If set to a non-empty string value, override the values of all the other
 90587 internationalization variables.

90588 **LC_COLLATE**

90589 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 90590 character collating elements within regular expressions. If this variable is not set to

90591 the POSIX locale, the results are unspecified.

90592 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 90593 characters (for example, single-byte as opposed to multi-byte characters in
 90594 arguments and input files), and the behavior of character classes within regular
 90595 expressions. If this variable is not set to the POSIX locale, the results are
 90596 unspecified.

90597 **LC_MESSAGES**

90598 Determine the locale that should be used to affect the format and contents of
 90599 diagnostic messages written to standard error.

90600 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

90601 ASYNCHRONOUS EVENTS

90602 Default.

90603 STDOUT

90604 If the **-t** option is specified, the text file of C source code output of *lex* shall be written to
 90605 standard output.

90606 If the **-t** option is not specified:

- 90607 • Implementation-defined informational, error, and warning messages concerning the
 90608 contents of *lex* source code input shall be written to either the standard output or standard
 90609 error.
- 90610 • If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be
 90611 written to either the standard output or standard error, in an implementation-defined
 90612 format. These statistics may also be generated if table sizes are specified with a '%'
 90613 operator in the *Definitions* section, as long as the **-n** option is not specified.

90614 STDERR

90615 If the **-t** option is specified, implementation-defined informational, error, and warning messages
 90616 concerning the contents of *lex* source code input shall be written to the standard error.

90617 If the **-t** option is not specified:

- 90618 1. Implementation-defined informational, error, and warning messages concerning the
 90619 contents of *lex* source code input shall be written to either the standard output or
 90620 standard error.
- 90621 2. If the **-v** option is specified and the **-n** option is not specified, *lex* statistics shall also be
 90622 written to either the standard output or standard error, in an implementation-defined
 90623 format. These statistics may also be generated if table sizes are specified with a '%'
 90624 operator in the *Definitions* section, as long as the **-n** option is not specified.

90625 OUTPUT FILES

90626 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the **-t**
 90627 option is present.

90628 EXTENDED DESCRIPTION

90629 Each input file shall contain *lex* source code, which is a table of regular expressions with
 90630 corresponding actions in the form of C program fragments.

90631 When **lex.yy.c** is compiled and linked with the *lex* library (using the **-ll** operand with *c99*), the
 90632 resulting program shall read character input from the standard input and shall partition it into
 90633 strings that match the given expressions.

90634 When an expression is matched, these actions shall occur:

- 90635 • The input string that was matched shall be left in *yytext* as a null-terminated string; *yytext*
- 90636 shall either be an external character array or a pointer to a character string. As explained in
- 90637 [Definitions in lex](#), the type can be explicitly selected using the `%array` or `%pointer`
- 90638 declarations, but the default is implementation-defined.
- 90639 • The external `int yyleng` shall be set to the length of the matching string.
- 90640 • The expression's corresponding program fragment, or action, shall be executed.

90641 During pattern matching, *lex* shall search the set of patterns for the single longest possible

90642 match. Among rules that match the same number of characters, the rule given first shall be

90643 chosen.

90644 The general format of *lex* source shall be:

```

90645     Definitions
90646     %%
90647     Rules
90648     %%
90649     UserSubroutines

```

90650 The first "%%" is required to mark the beginning of the rules (regular expressions and actions);

90651 the second "%%" is required only if user subroutines follow.

90652 Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program

90653 fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly,

90654 anything in the *Definitions* section included between delimiter lines containing only "%{" and

90655 "%}" shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

90656 Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing

90657 at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c`

90658 after the declarations of variables for the `yylex()` function and before the first line of code in

90659 `yylex()`. Thus, user variables local to `yylex()` can be declared here, as well as application code to

90660 execute upon entry to `yylex()`.

90661 The action taken by *lex* when encountering any input beginning with a <blank> or within "%{"

90662 and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is

90663 undefined. The presence of such input may result in an erroneous definition of the `yylex()`

90664 function.

90665 Definitions in lex

90666 *Definitions* appear before the first "%%" delimiter. Any line in this section not contained between

90667 "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex*

90668 substitution string. The format of these lines shall be:

```

90669 name substitute

```

90670 If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is

90671 undefined. The string *substitute* shall replace the string `{name}` when it is used in a rule. The *name*

90672 string shall be recognized in this context only when the braces are provided and when it does

90673 not appear within a bracket expression or within double-quotes.

90674 In the *Definitions* section, any line beginning with a '%' (percent sign) character and followed by

90675 an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions.

90676 Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall

90677 define a set of exclusive start conditions. When the generated scanner is in a %s state, patterns

90678 with no state specified shall be also active; in a %x state, such patterns shall not be active. The

90679 rest of the line, after the first word, shall be considered to be one or more <blank>-separated

90680 names of start conditions. Start condition names shall be constructed in the same way as

90681 definition names. Start conditions can be used to restrict the matching of regular expressions to
90682 one or more states as described in [Regular Expressions in lex](#) (on page 2755).

90683 Implementations shall accept either of the following two mutually-exclusive declarations in the
90684 *Definitions* section:

90685 **%array** Declare the type of *yytext* to be a null-terminated character array.

90686 **%pointer** Declare the type of *yytext* to be a pointer to a null-terminated character string.

90687 The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of
90688 the scanner source file (that is, via an **extern**), the application shall include the appropriate
90689 **%array** or **%pointer** declaration in the scanner source file.

90690 Implementations shall accept declarations in the *Definitions* section for setting certain internal
90691 table sizes. The declarations are shown in the following table.

90692 **Table 4-9** Table Size Declarations in *lex*

Declaration	Description	Minimum Value
90693 %p <i>n</i>	Number of positions	2 500
90694 %n <i>n</i>	Number of states	500
90695 %a <i>n</i>	Number of transitions	2 000
90696 %e <i>n</i>	Number of parse tree nodes	1 000
90697 %k <i>n</i>	Number of packed character classes	1 000
90698 %o <i>n</i>	Size of the output array	3 000

90700 In the table, *n* represents a positive decimal integer, preceded by one or more <blank>s. The
90701 exact meaning of these table size numbers is implementation-defined. The implementation shall
90702 document how these numbers affect the *lex* utility and how they are related to any output that
90703 may be generated by the implementation should limitations be encountered during the
90704 execution of *lex*. It shall be possible to determine from this output which of the table size values
90705 needs to be modified to permit *lex* to successfully generate tables for the input language. The
90706 values in the column Minimum Value represent the lowest values conforming implementations
90707 shall provide.

90708 Rules in *lex*

90709 The rules in *lex* source files are a table in which the left column contains regular expressions and
90710 the right column contains actions (C program fragments) to be executed when the expressions
90711 are recognized.

90712 *ERE action*

90713 *ERE action*

90714 ...

90715 The extended regular expression (ERE) portion of a row shall be separated from *action* by one or
90716 more <blank>s. A regular expression containing <blank>s shall be recognized under one of the
90717 following conditions:

- 90718 • The entire expression appears within double-quotes.
- 90719 • The <blank>s appear within double-quotes or square brackets.
- 90720 • Each <blank> is preceded by a backslash character.

90721

User Subroutines in lex

90722

Anything in the user subroutines section shall be copied to `lex.yy.c` following `yylex()`.

90723

Regular Expressions in lex

90724

The `lex` utility shall support the set of extended regular expressions (see XBD [Section 9.4](#), on page 174), with the following additions and exceptions to the syntax:

90725

90726

" . . . " Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that backslash escapes (which appear in the following table) shall be recognized. Any backslash-escape sequence shall be terminated by the closing quote. For example, `"\01"1` represents a single string: the octal value 1 followed by the character `'1'`.

90727

90728

90729

90730

90731

`<state>r, <state1,state2,...>r`

90732

90733

90734

90735

90736

90737

The regular expression `r` shall be matched only when the program is in one of the start conditions indicated by `state`, `state1`, and so on; see [Actions in lex](#) (on page 2757). (As an exception to the typographical conventions of the rest of this volume of POSIX.1-200x, in this case `<state>` does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

90738

`r/x`

90739

90740

90741

90742

90743

90744

90745

90746

The regular expression `r` shall be matched only if it is followed by an occurrence of regular expression `x` (`x` is the instance of trailing context, further defined below). The token returned in `yytext` shall only match `r`. If the trailing portion of `r` matches the beginning of `x`, the result is unspecified. The `r` expression cannot include further trailing context or the `'$'` (match-end-of-line) operator; `x` cannot include the `'^'` (match-beginning-of-line) operator, nor trailing context, nor the `'$'` operator. That is, only one occurrence of trailing context is allowed in a `lex` regular expression, and the `'^'` operator only can be used at the beginning of such an expression.

90747

`{name}`

90748

90749

90750

90751

When `name` is one of the substitution symbols from the *Definitions* section, the string, including the enclosing braces, shall be replaced by the `substitute` value. The `substitute` value shall be treated in the extended regular expression as if it were enclosed in parentheses. No substitution shall occur if `{name}` occurs within a bracket expression or within double-quotes.

90752

Within an ERE, a backslash character shall be considered to begin an escape sequence as specified in the table in XBD [Chapter 5](#) (on page 107) (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences in the following table shall be recognized.

90753

90754

90755

A literal `<newline>` cannot occur within an ERE; the escape sequence `'\n'` can be used to represent a `<newline>`. A `<newline>` shall not be matched by a period operator.

90756

90757

Table 4-10 Escape Sequences in *lex*

90758

90759

90760

90761

90762

90763

90764

90765

90766

90767

90768

90769

90770

90771

90772

90773

90774

90775

90776

90777

90778

90779

90780

90781

Escape Sequence	Description	Meaning
<code>\digits</code>	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading <code>'\'</code> for each byte.
<code>\xdigits</code>	A backslash character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
<code>\c</code>	A backslash character followed by any character not described in this table or in the table in XBD Chapter 5 (on page 107) (<code>'\'</code> , <code>'\a'</code> , <code>'\b'</code> , <code>'\f'</code> , <code>'\n'</code> , <code>'\r'</code> , <code>'\t'</code> , <code>'\v'</code>).	The character <code>'c'</code> , unchanged.

90782

90783

90784

Note: If a `'\x'` sequence needs to be immediately followed by a hexadecimal digit character, a sequence such as `"\x1" "1"` can be used, which represents a character containing the value 1, followed by the character `'1'`.

90785

90786

90787

The order of precedence given to extended regular expressions for *lex* differs from that specified in XBD Section 9.4 (on page 174). The order of precedence for *lex* shall be as shown in the following table, from high to low.

90788

90789

90790

90791

90792

Note: The escaped characters entry is not meant to imply that these are operators, but they are included in the table to show their relationships to the true operators. The start condition, trailing context, and anchoring notations have been omitted from the table because of the placement restrictions described in this section; they can only appear at the beginning or ending of an ERE.

90793

Table 4-11 ERE Precedence in *lex*

90794

90795

90796

90797

90798

90799

90800

90801

90802

90803

90804

Extended Regular Expression	Precedence
<i>collation-related bracket symbols</i>	<code>[= =]</code> <code>[: :]</code> <code>[. .]</code>
<i>escaped characters</i>	<code>\<special character></code>
<i>bracket expression</i>	<code>[]</code>
<i>quoting</i>	<code>"..."</code>
<i>grouping</i>	<code>()</code>
<i>definition</i>	<code>{name}</code>
<i>single-character RE duplication</i>	<code>* + ?</code>
<i>concatenation</i>	
<i>interval expression</i>	<code>{m,n}</code>
<i>alternation</i>	<code> </code>

90805

The ERE anchoring operators `'^'` and `'$'` do not appear in the table. With *lex* regular

90806 expressions, these operators are restricted in their use: the '^' operator can only be used at the
 90807 beginning of an entire regular expression, and the '\$' operator only at the end. The operators
 90808 apply to the entire regular expression. Thus, for example, the pattern "(^abc)|(def\$)" is
 90809 undefined; it can instead be written as two separate rules, one with the regular expression
 90810 "^abc" and one with "def\$", which share a common action via the special '|' action (see
 90811 below). If the pattern were written "^abc|def\$", it would match either "abc" or "def" on a
 90812 line by itself.

90813 Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex*
 90814 implementations. An example of embedded anchoring would be for patterns such as
 90815 "(^|)foo(|\$)" to match "foo" when it exists as a complete word. This functionality can
 90816 be obtained using existing *lex* features:

```
90817 ^foo/[ \n]      |  
90818 " foo"/[ \n]    /* Found foo as a separate word. */
```

90819 Note also that '\$' is a form of trailing context (it is equivalent to "/\n") and as such cannot be
 90820 used with regular expressions containing another instance of the operator (see the preceding
 90821 discussion of trailing context).

90822 The additional regular expressions trailing-context operator '/' can be used as an ordinary
 90823 character if presented within double-quotes, "/"; preceded by a backslash, "\/"; or within a
 90824 bracket expression, "[/]". The start-condition '<' and '>' operators shall be special only in a
 90825 start condition at the beginning of a regular expression; elsewhere in the regular expression they
 90826 shall be treated as ordinary characters.

90827 Actions in lex

90828 The action to be taken when an ERE is matched can be a C program fragment or the special
 90829 actions described below; the program fragment can contain one or more C statements, and can
 90830 also include special actions. The empty C statement ';' shall be a valid action; any string in the
 90831 *lex.yy.c* input that matches the pattern portion of such a rule is effectively ignored or skipped.
 90832 However, the absence of an action shall not be valid, and the action *lex* takes in such a condition
 90833 is undefined.

90834 The specification for an action, including C statements and special actions, can extend across
 90835 several lines if enclosed in braces:

```
90836 ERE <one or more blanks> { program statement  
90837                          program statement }
```

90838 The default action when a string in the input to a *lex.yy.c* program is not matched by any
 90839 expression shall be to copy the string to the output. Because the default behavior of a program
 90840 generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that
 90841 has just "%%" shall generate a C program that simply copies the input to the output unchanged.

90842 Four special actions shall be available:

```
90843 | ECHO; REJECT; BEGIN
```

90844 | The action '|' means that the action for the next rule is the action for this rule.
 90845 Unlike the other three actions, '|' cannot be enclosed in braces or be semicolon-
 90846 terminated; the application shall ensure that it is specified alone, with no other
 90847 actions.

90848 **ECHO;** Write the contents of the string *yytext* on the output.

90849 **REJECT;** Usually only a single expression is matched by a given string in the input.
 90850 **REJECT** means "continue to the next expression that matches the current input",
 90851 and shall cause whatever rule was the second choice after the current rule to be
 90852 executed for the same input. Thus, multiple rules can be matched and executed for

90853 one input string or overlapping input strings. For example, given the regular
 90854 expressions "xyz" and "xy" and the input "xyz", usually only the regular
 90855 expression "xyz" would match. The next attempted match would start after **z**. If
 90856 the last action in the "xyz" rule is **REJECT**, both this rule and the "xy" rule
 90857 would be executed. The **REJECT** action may be implemented in such a fashion that
 90858 flow of control does not continue after it, as if it were equivalent to a **goto**
 90859 to another part of *yylex()*. The use of **REJECT** may result in somewhat larger and
 90860 slower scanners.

90861 **BEGIN** The action:

90862 `BEGIN newstate;`

90863 switches the state (start condition) to *newstate*. If the string *newstate* has not been
 90864 declared previously as a start condition in the *Definitions* section, the results are
 90865 unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

90866 The functions or macros described below are accessible to user code included in the *lex* input. It
 90867 is unspecified whether they appear in the C code output of *lex*, or are accessible only through the
 90868 `-l l` operand to *c99* (the *lex* library).

90869 **int *yylex*(void)**

90870 Performs lexical analysis on the input; this is the primary function generated by the *lex*
 90871 utility. The function shall return zero when the end of input is reached; otherwise, it shall
 90872 return non-zero values (tokens) determined by the actions that are selected.

90873 **int *yymore*(void)**

90874 When called, indicates that when the next input string is recognized, it is to be appended to
 90875 the current value of *yytext* rather than replacing it; the value in *yylen* shall be adjusted
 90876 accordingly.

90877 **int *yyless*(int *n*)**

90878 Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters as
 90879 if they had not been read; the value in *yylen* shall be adjusted accordingly.

90880 **int *input*(void)**

90881 Returns the next character from the input, or zero on end-of-file. It shall obtain input from
 90882 the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning
 90883 has begun, the effect of altering the value of *yyin* is undefined. The character read shall be
 90884 removed from the input stream of the scanner without any processing by the scanner.

90885 **int *unput*(int *c*)**

90886 Returns the character '*c*' to the input; *yytext* and *yylen* are undefined until the next
 90887 expression is matched. The result of using *unput()* for more characters than have been input
 90888 is unspecified.

90889 The following functions shall appear only in the *lex* library accessible through the `-l l` operand;
 90890 they can therefore be redefined by a conforming application:

90891 **int *yywrap*(void)**

90892 Called by *yylex()* at end-of-file; the default *yywrap()* shall always return 1. If the application
 90893 requires *yylex()* to continue processing with another source of input, then the application
 90894 can include a function *yywrap()*, which associates another file with the external variable
 90895 `FILE *yyin` and shall return a value of zero.

90896 **int *main*(int *argc*, char **argv*[])**

90897 Calls *yylex()* to perform lexical analysis, then exits. The user code can contain *main()* to
 90898 perform application-specific operations, calling *yylex()* as applicable.

90899 Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin
 90900 with the prefix **yy** or **YY**.

90901 **EXIT STATUS**

90902 The following exit values shall be returned:

90903 0 Successful completion.

90904 >0 An error occurred.

90905 **CONSEQUENCES OF ERRORS**

90906 Default.

90907 **APPLICATION USAGE**90908 Conforming applications are warned that in the *Rules* section, an ERE without an action is not
90909 acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or
90910 runtime errors.90911 The purpose of *input()* is to take characters off the input stream and discard them as far as the
90912 lexical analysis is concerned. A common use is to discard the body of a comment once the
90913 beginning of a comment is recognized.90914 The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex*
90915 source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer
90916 interpret the regular expressions given in the *lex* source according to the environment specified
90917 when the lexical analyzer is executed, but this is not possible with the current *lex* technology.
90918 Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the
90919 lexical requirements of the input language being described, which is frequently locale-specific
90920 anyway. (For example, writing an analyzer that is used for French text is not automatically
90921 useful for processing other languages.)90922 **EXAMPLES**90923 The following is an example of a *lex* program that implements a rudimentary scanner for a
90924 Pascal-like syntax:

```

90925 %{
90926 /* Need this for the call to atof() below. */
90927 #include <math.h>
90928 /* Need this for printf(), fopen(), and stdin below. */
90929 #include <stdio.h>
90930 %}
90931
90932 DIGIT    [0-9]
90933 ID       [a-z][a-z0-9]*
90934
90935 %%
90936 {DIGIT}+ {
90937     printf("An integer: %s (%d)\n", yytext,
90938         atoi(yytext));
90939 }
90940 {DIGIT}+"."{DIGIT}* {
90941     printf("A float: %s (%g)\n", yytext,
90942         atof(yytext));
90943 }
90944 if|then|begin|end|procedure|function {
90945     printf("A keyword: %s\n", yytext);
90946 }
90947 {ID}    printf("An identifier: %s\n", yytext);
90948 "+"|"-"|"*"|"/"    printf("An operator: %s\n", yytext);
90949 {"^[^}\n]*"} /* Eat up one-line comments. */

```

```

90948 [ \t\n]+      /* Eat up white space. */
90949 . printf("Unrecognized character: %s\n", yytext);
90950 %%
90951 int main(int argc, char *argv[])
90952 {
90953     ++argv, --argc; /* Skip over program name. */
90954     if (argc > 0)
90955         yyin = fopen(argv[0], "r");
90956     else
90957         yyin = stdin;
90958     yylex();
90959 }

```

RATIONALE

Even though the `-c` option and references to the C language are retained in this description, *lex* may be generalized to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since the *lex* input specification is essentially language-independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex* source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is assumed to be presented in the POSIX locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by *lex* would interpret EREs specified in the *lex* source in terms of the environment variables specified when *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology.

The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard use of escape sequences.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

There is no detailed output format specification. The observed behavior of *lex* under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified avoids these formatting questions and problems with internationalization.

Although the `%x` specifier for *exclusive* start conditions is not historical practice, it is believed to be a minor change to historical implementations and greatly enhances the usability of *lex* programs since it permits an application to obtain the expected functionality with fewer statements.

The `%array` and `%pointer` declarations were added as a compromise between historical systems. The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most historical programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by *flex* in its case is to null-terminate the token in place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multi-file programs with external references to *yytext* outside the scanner source file should continue to operate on their historical systems, but would require one of the new declarations to be considered strictly portable.

- 90997 The description of EREs avoids unnecessary duplication of ERE details because their meanings
90998 within a *lex* ERE are the same as that for the ERE in this volume of POSIX.1-200x.
- 90999 The reason for the undefined condition associated with text beginning with a <blank> or within
91000 "%{ " and "%}" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD
91001 and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the
91002 beginning) to unreachable areas of the *yylex()* function (the code is written directly after a *break*
91003 statement). In some cases, the System V *lex* generates an error message or a syntax error,
91004 depending on the form of indented input.
- 91005 The intention in breaking the list of functions into those that may appear in *lex.yy.c* versus those
91006 that only appear in *libl.a* is that only those functions in *libl.a* can be reliably redefined by a
91007 conforming application.
- 91008 The descriptions of standard output and standard error are somewhat complicated because
91009 historical *lex* implementations chose to issue diagnostic messages to standard output (unless *-t*
91010 was given). POSIX.1-200x allows this behavior, but leaves an opening for the more expected
91011 behavior of using standard error for diagnostics. Also, the System V behavior of writing the
91012 statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The
91013 programmer can always precisely obtain the desired results by using either the *-t* or *-n* options.
- 91014 The OPERANDS section does not mention the use of *-* as a synonym for standard input; not all
91015 historical implementations support such usage for any of the *file* operands.
- 91016 A description of the *translation table* was deleted from early proposals because of its relatively
91017 low usage in historical applications.
- 91018 The change to the definition of the *input()* function that allows buffering of input presents the
91019 opportunity for major performance gains in some applications.
- 91020 The following examples clarify the differences between *lex* regular expressions and regular
91021 expressions appearing elsewhere in this volume of POSIX.1-200x. For regular expressions of the
91022 form "*r/x*", the string matching *r* is always returned; confusion may arise when the beginning
91023 of *x* matches the trailing portion of *r*. For example, given the regular expression "*a*b/cc*" and
91024 the input "*aaabcc*", *ytext* would contain the string "*aaab*" on this match. But given the
91025 regular expression "*x*/xy*" and the input "*xxxxy*", the token *xxx*, not *xx*, is returned by some
91026 implementations because *xxx* matches "*x**".
- 91027 In the rule "*ab*/bc*", the "*b**" at the end of *r* extends *r*'s match into the beginning of the
91028 trailing context, so the result is unspecified. If this rule were "*ab/bc*", however, the rule
91029 matches the text "*ab*" when it is followed by the text "*bc*". In this latter case, the matching of *r*
91030 cannot extend into the beginning of *x*, so the result is specified.
- 91031 **FUTURE DIRECTIONS**
- 91032 None.
- 91033 **SEE ALSO**
- 91034 *c99*, *ed*, *yacc*
- 91035 XBD [Chapter 5](#) (on page 107), [Chapter 8](#) (on page 159), [Chapter 9](#) (on page 167), [Section 12.2](#) (on +
91036 page 201)
- 91037 **CHANGE HISTORY**
- 91038 First released in Issue 2.
- 91039 **Issue 6**
- 91040 This utility is marked as part of the C-Language Development Utilities option.
- 91041 The obsolescent *-c* option is removed.
- 91042 The normative text is reworded to avoid use of the term "must" for application requirements.

91043

91044

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/14 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.

91045

Issue 7

91046

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

91047

91048

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



91049 **NAME**91050 link — call *link()* function91051 **SYNOPSIS**91052 XSI `link file1 file2`91053 **DESCRIPTION**91054 The *link* utility shall perform the function call:91055 `link(file1, file2);`91056 A user may need appropriate privilege to invoke the *link* utility.91057 **OPTIONS**

91058 None.

91059 **OPERANDS**

91060 The following operands shall be supported:

91061 *file1* The pathname of an existing file.91062 *file2* The pathname of the new directory entry to be created.91063 **STDIN**

91064 Not used.

91065 **INPUT FILES**

91066 Not used.

91067 **ENVIRONMENT VARIABLES**91068 The following environment variables shall affect the execution of *link*:91069 *LANG* Provide a default value for the internationalization variables that are unset or null. |
91070 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
91071 variables used to determine the values of locale categories.)91072 *LC_ALL* If set to a non-empty string value, override the values of all the other
91073 internationalization variables.91074 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
91075 characters (for example, single-byte as opposed to multi-byte characters in
91076 arguments).91077 *LC_MESSAGES*
91078 Determine the locale that should be used to affect the format and contents of
91079 diagnostic messages written to standard error.91080 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.91081 **ASYNCHRONOUS EVENTS**

91082 Default.

91083 **STDOUT**

91084 None.

91085 **STDERR**

91086 The standard error shall be used only for diagnostic messages.

- 91087 **OUTPUT FILES**
- 91088 None.
- 91089 **EXTENDED DESCRIPTION**
- 91090 None.
- 91091 **EXIT STATUS**
- 91092 The following exit values shall be returned:
- 91093 0 Successful completion.
- 91094 >0 An error occurred.
- 91095 **CONSEQUENCES OF ERRORS**
- 91096 Default.
- 91097 **APPLICATION USAGE**
- 91098 None.
- 91099 **EXAMPLES**
- 91100 None.
- 91101 **RATIONALE**
- 91102 None.
- 91103 **FUTURE DIRECTIONS**
- 91104 None.
- 91105 **SEE ALSO**
- 91106 *ln*, *unlink*
- 91107 XBD [Chapter 8](#) (on page 159)
- 91108 XSH *link*
- 91109 **CHANGE HISTORY**
- 91110 First released in Issue 5.

91111 **NAME**91112 `ln` — link files91113 **SYNOPSIS**91114 `ln [-fs] source_file target_file`91115 `ln [-fs] source_file... target_dir`91116 **DESCRIPTION**

91117 In the first synopsis form, the `ln` utility shall create a new directory entry (link) at the destination
 91118 path specified by the `target_file` operand. If the `-s` option is specified, a symbolic link shall be
 91119 created for the file specified by the `source_file` operand. This first synopsis form shall be assumed
 91120 when the final operand does not name an existing directory; if more than two operands are
 91121 specified and the final is not an existing directory, an error shall result.

91122 In the second synopsis form, the `ln` utility shall create a new directory entry (link), or if the `-s`
 91123 option is specified a symbolic link, for each file specified by a `source_file` operand, at a
 91124 destination path in the existing directory named by `target_dir`.

91125 If the last operand specifies an existing file of a type not specified by the System Interfaces
 91126 volume of POSIX.1-200x, the behavior is implementation-defined.

91127 The corresponding destination path for each `source_file` shall be the concatenation of the target
 91128 directory pathname, a slash character, and the last pathname component of the `source_file`. The
 91129 second synopsis form shall be assumed when the final operand names an existing directory.

91130 For each `source_file`:

- 91131 1. If the destination path exists:
- 91132 a. If the `-f` option is not specified, `ln` shall write a diagnostic message to standard
 91133 error, do nothing more with the current `source_file`, and go on to any remaining
 91134 `source_files`.
 - 91135 b. Actions shall be performed equivalent to the `unlink()` function defined in the
 91136 System Interfaces volume of POSIX.1-200x, called using `destination` as the `path`
 91137 argument. If this fails for any reason, `ln` shall write a diagnostic message to
 91138 standard error, do nothing more with the current `source_file`, and go on to any
 91139 remaining `source_files`.
- 91140 2. If the `-s` option is specified, `ln` shall create a symbolic link named by the destination path
 91141 and containing as its pathname `source_file`. The `ln` utility shall do nothing more with
 91142 `source_file` and shall go on to any remaining files.
- 91143 3. If `source_file` is a symbolic link, actions shall be performed equivalent to the `link()` function
 91144 using the object that `source_file` references as the `path1` argument and the destination path
 91145 as the `path2` argument. The `ln` utility shall do nothing more with `source_file` and shall go on
 91146 to any remaining files.
- 91147 4. Actions shall be performed equivalent to the `link()` function defined in the System
 91148 Interfaces volume of POSIX.1-200x using `source_file` as the `path1` argument, and the
 91149 destination path as the `path2` argument.

91150 **OPTIONS**91151 The `ln` utility shall conform to XBD [Section 12.2](#) (on page 201).

91152 The following option shall be supported:

91153 **-f** Force existing destination pathnames to be removed to allow the link.

91154 **-s** Create symbolic links instead of hard links.

OPERANDS

91155 The following operands shall be supported:

91157 *source_file* A pathname of a file to be linked. If the **-s** option is specified, no restrictions on the type of file or on its existence shall be made. If the **-s** option is not specified, whether a directory can be linked is implementation-defined.

91160 *target_file* The pathname of the new directory entry to be created.

91161 *target_dir* A pathname of an existing directory in which the new directory entries are created.

STDIN

91162 Not used.

INPUT FILES

91163 None.

ENVIRONMENT VARIABLES

91164 The following environment variables shall affect the execution of *ln*:

91168 **LANG** Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)

91171 **LC_ALL** If set to a non-empty string value, override the values of all the other internationalization variables.

91173 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

91176 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

91179 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

91180 Default.

STDOUT

91182 Not used.

STDERR

91183 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

91184 None.

EXTENDED DESCRIPTION

91185 None.

EXIT STATUS

91186 The following exit values shall be returned:

91191 0 All the specified files were linked successfully.

91192 >0 An error occurred.

91194 **CONSEQUENCES OF ERRORS**

91195 Default.

91196 **APPLICATION USAGE**

91197 None.

91198 **EXAMPLES**

91199 None.

91200 **RATIONALE**91201 The CONSEQUENCES OF ERRORS section does not require *ln -f a b* to remove *b* if a
91202 subsequent link operation would fail.91203 Some historic versions of *ln* (including the one specified by the SVID) unlink the destination file,
91204 if it exists, by default. If the mode does not permit writing, these versions prompt for
91205 confirmation before attempting the unlink. In these versions the *-f* option causes *ln* not to
91206 attempt to prompt for confirmation.91207 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself
91208 is not writable (although the directory must be). Early proposals specified this functionality.91209 This volume of POSIX.1-200x does not allow the *ln* utility to unlink existing destination paths by
91210 default for the following reasons:

- 91211 • The *ln* utility has historically been used to provide locking for shell applications, a usage
- 91212 that is incompatible with *ln* unlinking the destination path by default. There was no
- 91213 corresponding technical advantage to adding this functionality.
- 91214 • This functionality gave *ln* the ability to destroy the link structure of files, which changes
- 91215 the historical behavior of *ln*.
- 91216 • This functionality is easily replicated with a combination of *rm* and *ln*.
- 91217 • It is not historical practice in many systems; BSD and BSD-derived systems do not support
- 91218 this behavior. Unfortunately, whichever behavior is selected can cause scripts written
- 91219 expecting the other behavior to fail.
- 91220 • It is preferable that *ln* perform in the same manner as the *link()* function, which does not
- 91221 permit the target to exist already.

91222 This volume of POSIX.1-200x retains the *-f* option to provide support for shell scripts depending
91223 on the SVID semantics. It seems likely that shell scripts would not be written to handle
91224 prompting by *ln* and would therefore have specified the *-f* option.91225 The *-f* option is an undocumented feature of many historical versions of the *ln* utility, allowing
91226 linking to directories. These versions require modification.91227 Early proposals of this volume of POSIX.1-200x also required a *-i* option, which behaved like the
91228 *-i* options in *cp* and *mv*, prompting for confirmation before unlinking existing files. This was not
91229 historical practice for the *ln* utility and has been omitted.91230 **FUTURE DIRECTIONS**

91231 None.

91232 **SEE ALSO**91233 *chmod*, *find*, *pax*, *rm*

91234 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

91235 XSH *link*, *unlink*

91236

CHANGE HISTORY

91237

First released in Issue 2.

91238

Issue 6

91239

The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b draft standard.

91240

91241

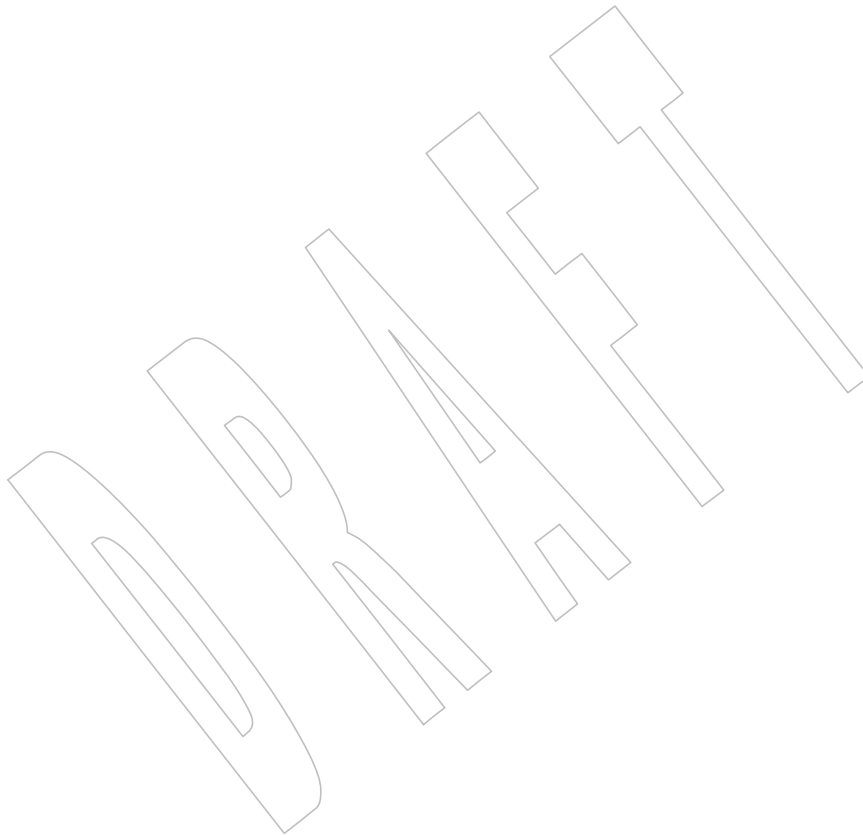
Issue 7

91242

SD5-XCU-ERN-27 is applied, adding a new paragraph to the RATIONALE.

91243

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



91244 **NAME**
 91245 locale — get locale-specific information

91246 **SYNOPSIS**
 91247 locale [-a|-m]

91248 locale [-ck] *name*...

91249 DESCRIPTION

91250 The *locale* utility shall write information about the current locale environment, or all public
 91251 locales, to the standard output. For the purposes of this section, a *public locale* is one provided by
 91252 the implementation that is accessible to the application.

91253 When *locale* is invoked without any arguments, it shall summarize the current locale
 91254 environment for each locale category as determined by the settings of the environment variables
 91255 defined in XBD [Chapter 7](#) (on page 121).

91256 When invoked with operands, it shall write values that have been assigned to the keywords in
 91257 the locale categories, as follows:

- 91258 • Specifying a keyword name shall select the named keyword and the category containing
 91259 that keyword.
- 91260 • Specifying a category name shall select the named category and all keywords in that
 91261 category.

91262 OPTIONS

91263 The *locale* utility shall conform to XBD [Section 12.2](#) (on page 201).

91264 The following options shall be supported:

- 91265 **-a** Write information about all available public locales. The available locales shall
 91266 include **POSIX**, representing the POSIX locale. The manner in which the
 91267 implementation determines what other locales are available is implementation-
 91268 defined.
- 91269 **-c** Write the names of selected locale categories; see the **STDOUT** section. The **-c**
 91270 option increases readability when more than one category is selected (for example,
 91271 via more than one keyword name or via a category name). It is valid both with
 91272 and without the **-k** option.
- 91273 **-k** Write the names and values of selected keywords. The implementation may omit
 91274 values for some keywords; see the **OPERANDS** section.
- 91275 **-m** Write names of available charmaps; see XBD [Section 6.1](#) (on page 111).

91276 OPERANDS

91277 The following operand shall be supported:

- 91278 *name* The name of a locale category as defined in XBD [Chapter 7](#) (on page 121), the name
 91279 of a keyword in a locale category, or the reserved name **charmap**. The named
 91280 category or keyword shall be selected for output. If a single *name* represents both a
 91281 locale category name and a keyword name in the current locale, the results are
 91282 unspecified. Otherwise, both category and keyword names can be specified as
 91283 *name* operands, in any sequence. It is implementation-defined whether any
 91284 keyword values are written for the categories *LC_CTYPE* and *LC_COLLATE*.

locale

Utilities

91285 **STDIN**

91286 Not used.

91287 **INPUT FILES**

91288 None.

91289 **ENVIRONMENT VARIABLES**91290 The following environment variables shall affect the execution of *locale*:

91291 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 91292 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 91293 variables used to determine the values of locale categories.)

91294 **LC_ALL** If set to a non-empty string value, override the values of all the other |
 91295 internationalization variables.

91296 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as |
 91297 characters (for example, single-byte as opposed to multi-byte characters in |
 91298 arguments and input files).

91299 **LC_MESSAGES**

91300 Determine the locale that should be used to affect the format and contents of |
 91301 diagnostic messages written to standard error.

91302 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

91303 **XSI** The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify |
 91304 the current locale environment to be written out; they shall be used if the **-a** option is not |
 91305 specified.

91306 **ASYNCHRONOUS EVENTS**

91307 Default.

91308 **STDOUT**91309 The *LANG* variable shall be written first using the format:91310 "LANG=%s\n", *<value>*91311 If *LANG* is not set or is an empty string, the value is the empty string.

91312 If *locale* is invoked without any options or operands, the names and values of the *LC_** |
 91313 environment variables described in this volume of POSIX.1-200x shall be written to the standard |
 91314 output, one variable per line, and each line using the following format. Only those variables set |
 91315 in the environment and not overridden by *LC_ALL* shall be written using this format:

91316 "%s=%s\n", *<variable_name>*, *<value>*

91317 The names of those *LC_** variables associated with locale categories defined in this volume of |
 91318 POSIX.1-200x that are not set in the environment or are overridden by *LC_ALL* shall be written |
 91319 in the following format:

91320 "%s=\"%s\" \n", *<variable_name>*, *<implied value>*

91321 The *<implied value>* shall be the name of the locale that has been selected for that category by the |
 91322 implementation, based on the values in *LANG* and *LC_ALL*, as described in XBD [Chapter 8](#) (on |
 91323 page 159).

91324 The *<value>* and *<implied value>* shown above shall be properly quoted for possible later reentry |
 91325 to the shell. The *<value>* shall not be quoted using double-quotes (so that it can be distinguished |
 91326 by the user from the *<implied value>* case, which always requires double-quotes).

91327 The *LC_ALL* variable shall be written last, using the first format shown above. If it is not set, it |
 91328 shall be written as:

91329 "LC_ALL=\n"

91330 If any arguments are specified:

91331 1. If the `-a` option is specified, the names of all the public locales shall be written, each in the
91332 following format:

91333 "%s\n", <locale name>

91334 2. If the `-c` option is specified, the names of all selected categories shall be written, each in
91335 the following format:

91336 "%s\n", <category name>

91337 If keywords are also selected for writing (see following items), the category name output
91338 shall precede the keyword output for that category.

91339 If the `-c` option is not specified, the names of the categories shall not be written; only the
91340 keywords, as selected by the <name> operand, shall be written.

91341 3. If the `-k` option is specified, the names and values of selected keywords shall be written.
91342 If a value is non-numeric and is not a compound keyword value, it shall be written in the
91343 following format:

91344 "%s=\"%s\"\\n", <keyword name>, <keyword value>

91345 If a value is a non-numeric compound keyword value, it shall either be written in the
91346 format:

91347 "%s=\"%s\"\\n", <keyword name>, <keyword value>

91348 where the <keyword value> is a single string of values separated by semicolons, or it shall
91349 be written in the format:

91350 "%s=%s\\n", <keyword name>, <keyword value>

91351 where the <keyword value> is encoded as a set of strings, each enclosed in double-
91352 quotation marks, separated by semicolons.

91353 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
91354 `localedef -f` option when the locale was created shall be written, with the word **charmap** as
91355 <keyword name>.

91356 If a value is numeric, it shall be written in one of the following formats:

91357 "%s=%d\\n", <keyword name>, <keyword value>

91358 "%s=%c%o\\n", <keyword name>, <escape character>, <keyword value>

91359 "%s=%cx%x\\n", <keyword name>, <escape character>, <keyword value>

91360 where the <escape character> is that identified by the **escape_char** keyword in the current
91361 locale; see XBD [Section 7.3](#) (on page 122).

91362 Compound keyword values (list entries) shall be separated in the output by semicolons.
91363 When included in keyword values, the semicolon, the double-quote, the backslash, and
91364 any control character shall be preceded (escaped) with the escape character.

91365 4. If the `-k` option is not specified, selected keyword values shall be written, each in the
91366 following format:

91367 "%s\\n", <keyword value>

91368 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
91369 `localedef -f` option when the locale was created shall be written.

- 91370 5. If the `-m` option is specified, then a list of all available charmaps shall be written, each in
91371 the format:

91372 "%s\n", <charmap>

91373 where <charmap> is in a format suitable for use as the option-argument to the `localedef -f`
91374 option.

91375 **STDERR**

91376 The standard error shall be used only for diagnostic messages.

91377 **OUTPUT FILES**

91378 None.

91379 **EXTENDED DESCRIPTION**

91380 None.

91381 **EXIT STATUS**

91382 The following exit values shall be returned:

91383 0 All the requested information was found and output successfully.

91384 >0 An error occurred.

91385 **CONSEQUENCES OF ERRORS**

91386 Default.

91387 **APPLICATION USAGE**

91388 If the `LANG` environment variable is not set or set to an empty value, or one of the `LC_*`
91389 environment variables is set to an unrecognized value, the actual locales assumed (if any) are
91390 implementation-defined as described in XBD [Chapter 8](#) (on page 159).

91391 Implementations are not required to write out the actual values for keywords in the categories
91392 `LC_CTYPE` and `LC_COLLATE`; however, they must write out the categories (allowing an
91393 application to determine, for example, which character classes are available).

91394 **EXAMPLES**

91395 In the following examples, the assumption is that locale environment variables are set as
91396 follows:

91397 LANG=locale_x
91398 LC_COLLATE=locale_y

91399 The command `locale` would result in the following output:

91400 LANG=locale_x
91401 LC_CTYPE="locale_x"
91402 LC_COLLATE=locale_y
91403 LC_TIME="locale_x"
91404 LC_NUMERIC="locale_x"
91405 LC_MONETARY="locale_x"
91406 LC_MESSAGES="locale_x"
91407 LC_ALL=

91408 The order of presentation of the categories is not specified by this volume of POSIX.1-200x.

91409 The command:

91410 LC_ALL=POSIX locale -ck decimal_point

91411 would produce:

91412 LC_NUMERIC
91413 decimal_point="."

91414 The following command shows an application of *locale* to determine whether a user-supplied
91415 response is affirmative:

```
91416 if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
91417 then
91418     affirmative processing goes here
91419 else
91420     non-affirmative processing goes here
91421 fi
```

91422 RATIONALE

91423 The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined
91424 because there is a questionable value in having a shell script receive an entire array of characters.
91425 It is also difficult to return a logical collation description, short of returning a complete *localedef*
91426 source.

91427 The **-m** option was included to allow applications to query for the existence of charmaps. The
91428 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the
91429 system.

91430 The **-c** option was included for readability when more than one category is selected (for
91431 example, via more than one keyword name or via a category name). It is valid both with and
91432 without the **-k** option.

91433 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the
91434 current locale was created, was included to allow applications needing the information to
91435 retrieve it.

91436 FUTURE DIRECTIONS

91437 None.

91438 SEE ALSO

91439 *localedef*

91440 XBD [Section 6.1](#) (on page 111), [Chapter 7](#) (on page 121), [Chapter 8](#) (on page 159), [Section 12.2](#) (on
91441 page 201)

91442 CHANGE HISTORY

91443 First released in Issue 4.

91444 Issue 5

91445 The FUTURE DIRECTIONS section is added.

91446 Issue 6

91447 The normative text is reworded to avoid use of the term “must” for application requirements.

91448 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/30 is applied, correcting an editorial error
91449 in the STDOUT section.

91450 Issue 7

91451 Austin Group Interpretations 1003.1-2001 #017, #021, and #088 are applied, clarifying the
91452 standard output for the **-k** option when *LANG* is not set or is an empty string.

91453 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91454 **NAME**

91455 localedef — define locale environment

91456 **SYNOPSIS**91457 localedef [-c] [-f *charmap*] [-i *sourcefile*] [-u *code_set_name*] *name*91458 **DESCRIPTION**

91459 The *localedef* utility shall convert source definitions for locale categories into a format usable by
 91460 the functions and utilities whose operational behavior is determined by the setting of the locale
 91461 environment variables defined in XBD [Chapter 7](#) (on page 121). It is implementation-defined
 91462 whether users have the capability to create new locales, in addition to those supplied by the
 91463 implementation. If the symbolic constant POSIX2_LOCALEDEF is defined, the system supports
 91464 XSI the creation of new locales. On XSI-conformant systems, the symbolic constant
 91465 POSIX2_LOCALEDEF shall be defined.

91466 The utility shall read source definitions for one or more locale categories belonging to the same
 91467 locale from the file named in the *-i* option (if specified) or from standard input.

91468 The *name* operand identifies the target locale. The utility shall support the creation of *public*, or
 91469 generally accessible locales, as well as *private*, or restricted-access locales. Implementations may
 91470 restrict the capability to create or modify public locales to users with the appropriate privileges.

91471 Each category source definition shall be identified by the corresponding environment variable
 91472 name and terminated by an **END** *category-name* statement. The following categories shall be
 91473 supported. In addition, the input may contain source for implementation-defined categories.

91474 *LC_CTYPE* Defines character classification and case conversion.

91475 *LC_COLLATE*
 91476 Defines collation rules.

91477 *LC_MONETARY*
 91478 Defines the format and symbols used in formatting of monetary information.

91479 *LC_NUMERIC*
 91480 Defines the decimal delimiter, grouping, and grouping symbol for non-monetary
 91481 numeric editing.

91482 *LC_TIME* Defines the format and content of date and time information.

91483 *LC_MESSAGES*
 91484 Defines the format and values of affirmative and negative responses.

91485 **OPTIONS**

91486 The *localedef* utility shall conform to XBD [Section 12.2](#) (on page 201).

91487 The following options shall be supported:

91488 *-c* Create permanent output even if warning messages have been issued.

91489 *-f charmap* Specify the pathname of a file containing a mapping of character symbols and
 91490 collating element symbols to actual character encodings. The format of the
 91491 *charmap* is described in XBD [Section 6.4](#) (on page 115). The application shall ensure
 91492 that this option is specified if symbolic names (other than collating symbols
 91493 defined in a **collating-symbol** keyword) are used. If the *-f* option is not present, an
 91494 implementation-defined character mapping shall be used.

91495 *-i inputfile* The pathname of a file containing the source definitions. If this option is not
 91496 present, source definitions shall be read from standard input. The format of the
 91497 *inputfile* is described in XBD [Section 7.3](#) (on page 122).

91498 **-u** *code_set_name*
 91499 Specify the name of a codeset used as the target mapping of character symbols and
 91500 collating element symbols whose encoding values are defined in terms of the
 91501 ISO/IEC 10646-1: 2000 standard position constant values.

OPERANDS

91502 The following operand shall be supported:

91503 *name* Identifies the locale; see XBD [Chapter 7](#) (on page 121) for a description of the use of
 91504 this name. If the name contains one or more slash characters, *name* shall be
 91505 interpreted as a pathname where the created locale definitions shall be stored. If
 91506 *name* does not contain any slash characters, the interpretation of the name is
 91507 implementation-defined and the locale shall be public. The ability to create public
 91508 locales in this way may be restricted to users with appropriate privileges. (As a
 91509 consequence of specifying one *name*, although several categories can be processed
 91510 in one execution, only categories belonging to the same locale can be processed.)
 91511

STDIN

91512 Unless the **-i** option is specified, the standard input shall be a text file containing one or more
 91513 locale category source definitions, as described in XBD [Section 7.3](#) (on page 122). When lines are
 91514 continued using the escape character mechanism, there is no limit to the length of the
 91515 accumulated continued line.
 91516

INPUT FILES

91517 The character set mapping file specified as the *charmap* option-argument is described in XBD
 91518 [Section 6.4](#) (on page 115). If a locale category source definition contains a **copy** statement, as
 91519 defined in XBD [Chapter 7](#) (on page 121), and the **copy** statement names a valid, existing locale,
 91520 then *localedef* shall behave as if the source definition had contained a valid category source
 91521 definition for the named locale.
 91522

ENVIRONMENT VARIABLES

91523 The following environment variables shall affect the execution of *localedef*:

91524 **LANG** Provide a default value for the internationalization variables that are unset or null.
 91525 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 91526 variables used to determine the values of locale categories.)
 91527

91528 **LC_ALL** If set to a non-empty string value, override the values of all the other
 91529 internationalization variables.

91530 **LC_COLLATE**
 91531 (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

91532 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 91533 characters (for example, single-byte as opposed to multi-byte characters in
 91534 arguments and input files). This variable has no affect on the processing of *localedef*
 91535 input data; the POSIX locale is used for this purpose, regardless of the value of this
 91536 variable.

91537 **LC_MESSAGES**
 91538 Determine the locale that should be used to affect the format and contents of
 91539 diagnostic messages written to standard error.

91540 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

91541 Default.
 91542

91543 **STDOUT**

91544 The utility shall report all categories successfully processed, in an unspecified format.

91545 **STDERR**

91546 The standard error shall be used only for diagnostic messages.

91547 **OUTPUT FILES**91548 The format of the created output is unspecified. If the *name* operand does not contain a slash, the
91549 existence of an output file for the locale is unspecified.91550 **EXTENDED DESCRIPTION**91551 When the `-u` option is used, the *code_set_name* option-argument shall be interpreted as an
91552 implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard
91553 position constant values shall be converted via an implementation-defined method. Both the
91554 ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal,
91555 hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset
91556 represented by the implementation-defined name can be any codeset that is supported by the
91557 implementation.91558 When conflicts occur between the *charmap* specification of `<code_set_name>`, `<mb_cur_max>`, or
91559 `<mb_cur_min>` and the implementation-defined interpretation of these respective items for the
91560 codeset represented by the `-u` option-argument *code_set_name*, the result is unspecified.91561 When conflicts occur between the *charmap* encoding values specified for symbolic names of
91562 characters of the portable character set and the implementation-defined assignment of character
91563 encoding values, the result is unspecified.91564 If a non-printable character in the *charmap* has a width specified that is not `-1`, the result will be
91565 undefined.91566 **EXIT STATUS**

91567 The following exit values shall be returned:

- 91568 0 No errors occurred and the locales were successfully created.
- 91569 1 Warnings occurred and the locales were successfully created.
- 91570 2 The locale specification exceeded implementation limits or the coded character set or sets
91571 used were not supported by the implementation, and no locale was created.
- 91572 3 The capability to create new locales is not supported by the implementation.
- 91573 >3 Warnings or errors occurred and no output was created.

91574 **CONSEQUENCES OF ERRORS**

91575 If an error is detected, no permanent output shall be created.

91576 If warnings occur, permanent output shall be created if the `-c` option was specified. The
91577 following conditions shall cause warning messages to be issued:

- 91578 • If a symbolic name not found in the *charmap* file is used for the descriptions of the
91579 *LC_CTYPE* or *LC_COLLATE* categories (for other categories, this shall be an error
91580 condition).
- 91581 • If the number of operands to the **order** keyword exceeds the `{COLL_WEIGHTS_MAX}`
91582 limit.
- 91583 • If optional keywords not supported by the implementation are present in the source.

91584 Other implementation-defined conditions may also cause warnings.

91585
91586
91587
91588
91589
91590
91591
91592

APPLICATION USAGE

The *charmap* definition is optional, and is contained outside the locale definition. This allows both completely self-defined source files, and generic sources (applicable to more than one codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the portable character set. As explained in XBD [Section 6.4](#) (on page 115), it is implementation-defined whether or not users or applications can provide additional character set description files. Therefore, the `-f` option might be operable only when an implementation-defined *charmap* is named.

91593
91594

EXAMPLES

None.

91595
91596
91597
91598

RATIONALE

The output produced by the *localedef* utility is implementation-defined. The *name* operand is used to identify the specific locale. (As a consequence, although several categories can be processed in one execution, only categories belonging to the same locale can be processed.)

91599
91600

FUTURE DIRECTIONS

None.

91601
91602

SEE ALSO

[locale](#)

91603
91604

XBD [Section 6.4](#) (on page 115), [Chapter 7](#) (on page 121), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

91605
91606

CHANGE HISTORY

First released in Issue 4.

91607
91608

Issue 6

The `-u` option is added, as specified in the IEEE P1003.2b draft standard.

91609
91610
91611

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/15 is applied, rewording text in the OPERANDS section describing the ability to create public locales.

91612
91613
91614

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/16 is applied, making the text consistent with the descriptions of **WIDTH** and **WIDTH_DEFAULT** in the Base Definitions volume of POSIX.1-200x.

91615
91616

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91617 **NAME**91618 `logger` — log messages91619 **SYNOPSIS**91620 `logger string...`91621 **DESCRIPTION**

91622 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*
 91623 operands provided by the user. The messages are expected to be evaluated later by personnel
 91624 performing system administration tasks.

91625 It is implementation-defined whether messages written in locales other than the POSIX locale
 91626 are effective.

91627 **OPTIONS**

91628 None.

91629 **OPERANDS**

91630 The following operand shall be supported:

91631 *string* One of the string arguments whose contents are concatenated together, in the order
 91632 specified, separated by single <space>s.

91633 **STDIN**

91634 Not used.

91635 **INPUT FILES**

91636 None.

91637 **ENVIRONMENT VARIABLES**91638 The following environment variables shall affect the execution of *logger*:

91639 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 91640 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 91641 variables used to determine the values of locale categories.)

91642 *LC_ALL* If set to a non-empty string value, override the values of all the other
 91643 internationalization variables.

91644 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 91645 characters (for example, single-byte as opposed to multi-byte characters in
 91646 arguments).

91647 *LC_MESSAGES*

91648 Determine the locale that should be used to affect the format and contents of
 91649 diagnostic messages written to standard error. (This means diagnostics from *logger*
 91650 to the user or application, not diagnostic messages that the user is sending to the
 91651 system administrator.)

91652 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

91653 **ASYNCHRONOUS EVENTS**

91654 Default.

91655 **STDOUT**

91656 Not used.

91657 **STDERR**
 91658 The standard error shall be used only for diagnostic messages.

91659 **OUTPUT FILES**
 91660 Unspecified.

91661 **EXTENDED DESCRIPTION**
 91662 None.

91663 **EXIT STATUS**
 91664 The following exit values shall be returned:

91665 0 Successful completion.

91666 >0 An error occurred.

91667 **CONSEQUENCES OF ERRORS**
 91668 Default.

91669 **APPLICATION USAGE**
 91670 This utility allows logging of information for later use by a system administrator or programmer
 91671 in determining why non-interactive utilities have failed. The locations of the saved messages,
 91672 their format, and retention period are all unspecified. There is no method for a conforming
 91673 application to read messages, once written.

91674 **EXAMPLES**
 91675 A batch application, running non-interactively, tries to read a configuration file and fails; it may
 91676 attempt to notify the system administrator with:

91677 `logger myname: unable to read file foo. [timestamp]`

91678 **RATIONALE**
 91679 The standard developers believed strongly that some method of alerting administrators to errors
 91680 was necessary. The obvious example is a batch utility, running non-interactively, that is unable to
 91681 read its configuration files or that is unable to create or write its results file. However, the
 91682 standard developers did not wish to define the format or delivery mechanisms as they have
 91683 historically been (and will probably continue to be) very system-specific, as well as involving
 91684 functionality clearly outside the scope of this volume of POSIX.1-200x.

91685 The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the
 91686 user or application, not diagnostic messages that the user is sending to the system administrator.

91687 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

91688 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient
 91689 justification to exclude these utilities from this volume of POSIX.1-200x. It is also arguable that
 91690 they are, in fact, testable, but that the tests themselves are not portable.

91691 **FUTURE DIRECTIONS**
 91692 None.

91693 **SEE ALSO**
 91694 *lp*, *mailx*, *write*

91695 XBD Chapter 8 (on page 159)

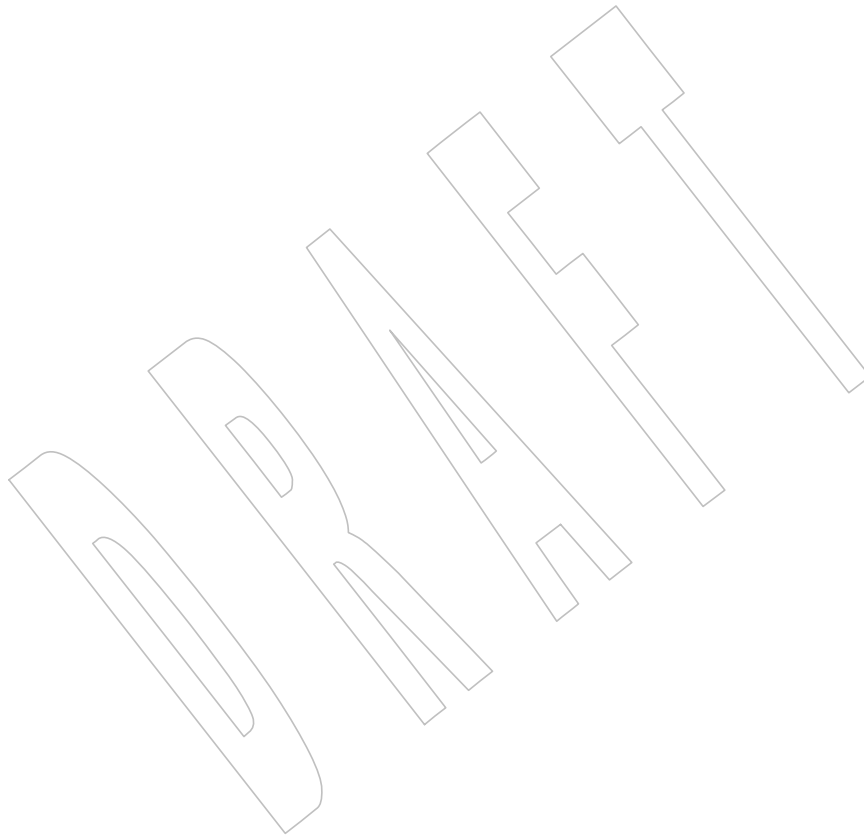
91696 **CHANGE HISTORY**
 91697 First released in Issue 4.

91698

91699

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



- 91700 **NAME**
- 91701 logname — return the user's login name
- 91702 **SYNOPSIS**
- 91703 logname
- 91704 **DESCRIPTION**
- 91705 The *logname* utility shall write the user's login name to standard output. The login name shall be
- 91706 the string that would be returned by the *getlogin()* function defined in the System Interfaces
- 91707 volume of POSIX.1-200x. Under the conditions where the *getlogin()* function would fail, the
- 91708 *logname* utility shall write a diagnostic message to standard error and exit with a non-zero exit
- 91709 status.
- 91710 **OPTIONS**
- 91711 None.
- 91712 **OPERANDS**
- 91713 None.
- 91714 **STDIN**
- 91715 Not used.
- 91716 **INPUT FILES**
- 91717 None.
- 91718 **ENVIRONMENT VARIABLES**
- 91719 The following environment variables shall affect the execution of *logname*:
- 91720 *LANG* Provide a default value for the internationalization variables that are unset or null. |
- 91721 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
- 91722 variables used to determine the values of locale categories.)
- 91723 *LC_ALL* If set to a non-empty string value, override the values of all the other
- 91724 internationalization variables.
- 91725 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
- 91726 characters (for example, single-byte as opposed to multi-byte characters in
- 91727 arguments).
- 91728 *LC_MESSAGES*
- 91729 Determine the locale that should be used to affect the format and contents of
- 91730 diagnostic messages written to standard error.
- 91731 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 91732 **ASYNCHRONOUS EVENTS**
- 91733 Default.
- 91734 **STDOUT**
- 91735 The *logname* utility output shall be a single line consisting of the user's login name:
- 91736 "%s\n", <login name>
- 91737 **STDERR**
- 91738 The standard error shall be used only for diagnostic messages.

91739
91740

91741
91742

91743
91744

91745
91746

91747
91748

91749
91750
91751

91752
91753

91754
91755
91756

91757
91758

91759
91760

91761
91762

91763
91764

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment changes could produce erroneous results.

EXAMPLES

None.

RATIONALE

The *passwd* file is not listed as required because the implementation may have other means of mapping login names.

FUTURE DIRECTIONS

None.

SEE ALSO

id, *who*

XBD [Chapter 8](#) (on page 159)

XSH *getlogin()*

CHANGE HISTORY

First released in Issue 2.

91765

NAME

91766

lp — send files to a printer

91767

SYNOPSIS

91768

lp [-c] [-d *dest*] [-n *copies*] [-msw] [-o *option*]... [-t *title*] [*file*...]

91769

DESCRIPTION

91770

The *lp* utility shall copy the input files to an output destination in an unspecified manner. The default output destination should be to a hardcopy device, such as a printer or microfilm recorder, that produces non-volatile, human-readable documents. If such a device is not available to the application, or if the system provides no such device, the *lp* utility shall exit with a non-zero exit status.

91771

91772

91773

91774

91775

The actual writing to the output device may occur some time after the *lp* utility successfully exits. During the portion of the writing that corresponds to each input file, the implementation shall guarantee exclusive access to the device.

91776

91777

91778

The *lp* utility shall associate a unique *request ID* with each request.

91779

Normally, a banner page is produced to separate and identify each print job. This page may be suppressed by implementation-defined conditions, such as an operator command or one of the *-o option* values.

91780

91781

91782

OPTIONS

91783

The *lp* utility shall conform to XBD [Section 12.2](#) (on page 201).

91784

The following options shall be supported:

91785

-c

Exit only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation. Normally, files are not copied, but are linked whenever possible. If the *-c* option is not given, then the user should be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the *-c* option, any changes made to the named files after the request is made but before it is printed may be reflected in the printed output. On some implementations, *-c* may be on by default.

91786

91787

91788

91789

91790

91791

91792

91793

-d *dest*

Specify a string that names the destination (*dest*). If *dest* is a printer, the request shall be printed only on that specific printer. If *dest* is a class of printers, the request shall be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, and so on), requests for specific destinations need not be accepted. Destination names vary between systems.

91794

91795

91796

91797

91798

91799

If *-d* is not specified, and neither the *LPDEST* nor *PRINTER* environment variable is set, an unspecified destination is used. The *-d dest* option shall take precedence over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are undefined when *dest* contains a value that is not a valid destination name.

91800

91801

91802

91803

-m

Send mail (see [mailx](#)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

91804

91805

-n *copies*

Write *copies* number of copies of the files, where *copies* is a positive decimal integer. The methods for producing multiple copies and for arranging the multiple copies when multiple *file* operands are used are unspecified, except that each file shall be output as an integral whole, not interleaved with portions of other files.

91806

91807

91808

- 91809 **-o option** Specify printer-dependent or class-dependent *options*. Several such *options* may be
91810 collected by specifying the **-o** option more than once.
- 91811 **-s** Suppress messages from *lp*.
- 91812 **-t title** Write *title* on the banner page of the output.
- 91813 **-w** Write a message on the user's terminal after the files have been printed. If the user
91814 is not logged in, then mail shall be sent instead.

OPERANDS

91815 The following operand shall be supported:

- 91816 *file* A pathname of a file to be output. If no *file* operands are specified, or if a *file*
91817 operand is '-', the standard input shall be used. If a *file* operand is used, but the
91818 **-c** option is not specified, the process performing the writing to the output device
91819 may have user and group permissions that differ from that of the process invoking
91820 *lp*.
91821

STDIN

91822 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
91823 See the INPUT FILES section.
91824

INPUT FILES

91825 The input files shall be text files.
91826

ENVIRONMENT VARIABLES

91827 The following environment variables shall affect the execution of *lp*:

- 91829 **LANG** Provide a default value for the internationalization variables that are unset or null. |
91830 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
91831 variables used to determine the values of locale categories.)
- 91832 **LC_ALL** If set to a non-empty string value, override the values of all the other
91833 internationalization variables.
- 91834 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
91835 characters (for example, single-byte as opposed to multi-byte characters in
91836 arguments and input files).
- 91837 **LC_MESSAGES** Determine the locale that should be used to affect the format and contents of
91838 diagnostic messages written to standard error and informative messages written to
91839 standard output.
91840
- 91841 **LC_TIME** Determine the format and contents of date and time strings displayed in the *lp*
91842 banner page, if any.
- 91843 **LPDEST** Determine the destination. If the **LPDEST** environment variable is not set, the
91844 **PRINTER** environment variable shall be used. The **-d dest** option takes precedence
91845 over **LPDEST**. Results are undefined when **-d** is not specified and **LPDEST**
91846 contains a value that is not a valid destination name.
- 91847 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 91848 **PRINTER** Determine the output device or destination. If the **LPDEST** and **PRINTER**
91849 environment variables are not set, an unspecified output device is used. The **-d**
91850 **dest** option and the **LPDEST** environment variable shall take precedence over
91851 **PRINTER**. Results are undefined when **-d** is not specified, **LPDEST** is unset, and
91852 **PRINTER** contains a value that is not a valid device or destination name.

91853 TZ Determine the timezone used to calculate date and time strings displayed in the *lp*
 91854 banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be
 91855 used.

91856 **ASYNCHRONOUS EVENTS**

91857 Default.

91858 **STDOUT**

91859 The *lp* utility shall write a *request ID* to the standard output, unless *-s* is specified. The format of
 91860 the message is unspecified. The request ID can be used on systems supporting the historical
 91861 *cancel* and *lpstat* utilities.

91862 **STDERR**

91863 The standard error shall be used only for diagnostic messages.

91864 **OUTPUT FILES**

91865 None.

91866 **EXTENDED DESCRIPTION**

91867 None.

91868 **EXIT STATUS**

91869 The following exit values shall be returned:

91870 0 All input files were processed successfully.

91871 >0 No output device was available, or an error occurred.

91872 **CONSEQUENCES OF ERRORS**

91873 Default.

91874 **APPLICATION USAGE**

91875 The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's
 91876 default page size.

91877 A conforming application can use one of the *file* operands only with the *-c* option or if the file is
 91878 publicly readable and guaranteed to be available at the time of printing. This is because
 91879 POSIX.1-200x gives the implementation the freedom to queue up the request for printing at
 91880 some later time by a different process that might not be able to access the file.

91881 **EXAMPLES**

91882 1. To print file *file*:

91883 `lp -c file`

91884 2. To print multiple files with headers:

91885 `pr file1 file2 | lp`

91886 **RATIONALE**

91887 The *lp* utility was designed to be a basic version of a utility that is already available in many
 91888 historical implementations. The standard developers considered that it should be implementable
 91889 simply as:

91890 `cat "$@" > /dev/lp`

91891 after appropriate processing of options, if that is how the implementation chose to do it and if
 91892 exclusive access could be granted (so that two users did not write to the device simultaneously).
 91893 Although in the future the standard developers may add other options to this utility, it should
 91894 always be able to execute with no options or operands and send the standard input to an
 91895 unspecified output device.

91896 This volume of POSIX.1-200x makes no representations concerning the format of the printed

91897 output, except that it must be “human-readable” and “non-volatile”. Thus, writing by default to
 91898 a disk or tape drive or a display terminal would not qualify. (Such destinations are not
 91899 prohibited when `-d dest`, `LPDEST`, or `PRINTER` are used, however.)

91900 This volume of POSIX.1-200x is worded such that a “print job” consisting of multiple input files,
 91901 possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with
 91902 another, but there is no statement that all the files or copies have to print out together.

91903 The `-c` option may imply a spooling operation, but this is not required. The utility can be
 91904 implemented to wait until the printer is ready and then wait until it is finished. Because of that,
 91905 there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

91906 On some historical systems, the request ID reported on the `STDOUT` can be used to later cancel
 91907 or find the status of a request using utilities not defined in this volume of POSIX.1-200x.

91908 Although the historical System V `lp` and BSD `lpr` utilities have provided similar functionality,
 91909 they used different names for the environment variable specifying the destination printer. Since
 91910 the name of the utility here is `lp`, `LPDEST` (used by the System V `lp` utility) was given precedence
 91911 over `PRINTER` (used by the BSD `lpr` utility). Since environments of users frequently contain one
 91912 or the other environment variable, the `lp` utility is required to recognize both. If this was not
 91913 done, many applications would send output to unexpected output devices when users moved
 91914 from system to system.

91915 Some have commented that `lp` has far too little functionality to make it worthwhile. Requests
 91916 have proposed additional options or operands or both that added functionality. The requests
 91917 included:

- 91918 • Wording *requiring* the output to be “hardcopy”
- 91919 • A requirement for multiple printers
- 91920 • Options for supporting various page-description languages

91921 Given that a compliant system is not required to even have a printer, placing further restrictions
 91922 upon the behavior of the printer is not useful. Since hardcopy format is so application-
 91923 dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that
 91924 should be required on all compliant systems.

91925 The term *unspecified* is used in this section in lieu of *implementation-defined* as most known
 91926 implementations would not be able to make definitive statements in their conformance
 91927 documents; the existence and usage of printers is very dependent on how the system
 91928 administrator configures each individual system.

91929 Since the default destination, device type, queuing mechanisms, and acceptable forms of input
 91930 are all unspecified, usage guidelines for what a conforming application can do are as follows:

- 91931 • Use the command in a pipeline, or with `-c`, so that there are no permission problems and
 91932 the files can be safely deleted or modified.
- 91933 • Limit output to text files of reasonable line lengths and printable characters and include no
 91934 device-specific formatting information, such as a page description language. The meaning
 91935 of “reasonable” in this context can only be answered as a quality-of-implementation issue,
 91936 but it should be apparent from historical usage patterns in the industry and the locale. The
 91937 `pr` and `fold` utilities can be used to achieve reasonable formatting for the default page size
 91938 of the implementation.

91939 Alternatively, the application can arrange its installation in such a way that it requires the system
 91940 administrator or operator to provide the appropriate information on `lp` options and environment
 91941 variable values.

91942 At a minimum, having this utility in this volume of POSIX.1-200x tells the industry that
 91943 conforming applications require a means to print output and provides at least a command name

91944 and *LPDEST* routing mechanism that can be used for discussions between vendors, application
 91945 developers, and users. The use of “should” in the DESCRIPTION of *lp* clearly shows the intent
 91946 of the standard developers, even if they cannot mandate that all systems (such as laptops) have
 91947 printers.

91948 This volume of POSIX.1-200x does not specify what the ownership of the process performing the
 91949 writing to the output device may be. If *-c* is not used, it is unspecified whether the process
 91950 performing the writing to the output device has permission to read *file* if there are any
 91951 restrictions in place on who may read *file* until after it is printed. Also, if *-c* is not used, the
 91952 results of deleting *file* before it is printed are unspecified.

91953 FUTURE DIRECTIONS

91954 None.

91955 SEE ALSO

91956 *mailx*

91957 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

91958 CHANGE HISTORY

91959 First released in Issue 2.

91960 Issue 6

91961 The following new requirements on POSIX implementations derive from alignment with the
 91962 Single UNIX Specification:

- 91963 • In the DESCRIPTION, the requirement to associate a unique request ID, and the normal
 91964 generation of a banner page is added.
- 91965 • In the OPTIONS section:
 - 91966 — The *-d dest* description is expanded, but references to *lpstat* are removed.
 - 91967 — The *-m*, *-o*, *-s*, *-t*, and *-w* options are added.
- 91968 • In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- 91969 • The STDOUT section is added.

91970 The normative text is reworded to avoid use of the term “must” for application requirements.

91971 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

91972 Issue 7

91973 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

91974 **NAME**

91975 ls — list directory contents

91976 **SYNOPSIS**

91977 xSI ls [-ACFRSacdilmnpqrtl] [-H|-L] [-fgos] [file...]

91978 **DESCRIPTION**

91979 For each operand that names a file of a type other than directory or symbolic link to a directory,
 91980 *ls* shall write the name of the file as well as any requested, associated information. For each
 91981 operand that names a file of type directory, *ls* shall write the names of files contained within the
 91982 directory as well as any requested, associated information. Filenames beginning with a period
 91983 ('.') and any associated information shall not be written out unless explicitly referenced, the
 91984 **-A** or **-a** option is supplied, or an implementation-defined condition causes them to be written.
 91985 If one of the **-d**, **-F**, or **-l** options are specified, and one of the **-H** or **-L** options are not specified,
 91986 for each operand that names a file of type symbolic link to a directory, *ls* shall write the name of
 91987 the file as well as any requested, associated information. If none of the **-d**, **-F**, or **-l** options are
 91988 specified, or the **-H** or **-L** options are specified, for each operand that names a file of type
 91989 symbolic link to a directory, *ls* shall write the names of files contained within the directory as
 91990 well as any requested, associated information.

91991 If no operands are specified, *ls* shall write the contents of the current directory. If more than one
 91992 operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-
 91993 directory operands separately according to the collating sequence in the current locale.

91994 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an
 91995 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic
 91996 message to standard error and shall either recover its position in the hierarchy or terminate.

91997 **OPTIONS**91998 The *ls* utility shall conform to XBD [Section 12.2](#) (on page 201).

91999 The following options shall be supported:

- 92000 **-A** Write out all directory entries, including those whose names begin with a period
 92001 ('.') but excluding the entries dot and dot-dot (if they exist).
- 92002 **-C** Write multi-text-column output with entries sorted down the columns, according
 92003 to the collating sequence. The number of text columns and the column separator
 92004 characters are unspecified, but should be adapted to the nature of the output
 92005 device.
- 92006 **-F** Do not follow symbolic links named as operands unless the **-H** or **-L** options are
 92007 specified. Write a slash ('/') immediately after each pathname that is a directory,
 92008 an asterisk ('*') after each that is executable, a vertical bar ('|') after each that is
 92009 a FIFO, and an at sign ('@') after each that is a symbolic link. For other file types,
 92010 other symbols may be written.
- 92011 **-H** If a symbolic link referencing a file of type directory is specified on the command
 92012 line, *ls* shall evaluate the file information and file type to be those of the file
 92013 referenced by the link, and not the link itself; however, *ls* shall write the name of
 92014 the link itself and not the file referenced by the link.
- 92015 **-L** Evaluate the file information and file type for all symbolic links (whether named
 92016 on the command line or encountered in a file hierarchy) to be those of the file
 92017 referenced by the link, and not the link itself; however, *ls* shall write the name of
 92018 the link itself and not the file referenced by the link. When **-L** is used with **-l**, write
 92019 the contents of symbolic links in the long format (see the STDOUT section).

92020		-R	Recursively list subdirectories encountered.
92021		-S	Sort with the primary key being file size (in decreasing order) and the secondary key being filename in the collating sequence (in increasing order).
92022			
92023		-a	Write out all directory entries, including those whose names begin with a period ('.').
92024			
92025		-c	Use time of last modification of the file status information (see <code><sys/stat.h></code> in the System Interfaces volume of POSIX.1-200x) instead of last modification of the file itself for sorting (-t) or writing (-l).
92026			
92027			
92028		-d	Do not follow symbolic links named as operands unless the -H or -L options are specified. Do not treat directories differently than other types of files. The use of -d with -R produces unspecified results.
92029			
92030			
92031		-f	List the entries in directory operands in the order they appear in the directory. The behavior for non-directory operands is unspecified. This option shall turn off -l , -t , -S , -s , and -r , and shall turn on -a .
92032			
92033			
92034	XSI	-g	The same as -l , except that the owner shall not be written.
92035		-i	For each file, write the file's file serial number (see <code>stat()</code> in the System Interfaces volume of POSIX.1-200x).
92036			
92037		-k	Set the block size for the -s option and the per-directory block count written for the -l , -n , -s , -g , and -o options (see the STDOUT section) to 1 024 bytes.
92038	XSI		
92039		-l	(The letter ell.) Do not follow symbolic links named as operands unless the -H or -L options are specified. Write out in long format (see the STDOUT section). When -l (ell) is specified, -1 (one) shall be assumed.
92040			
92041			
92042		-m	Stream output format; list files across the page, separated by commas.
92043		-n	The same as -l , except that the owner's UID and GID numbers shall be written, rather than the associated character strings.
92044			
92045	XSI	-o	The same as -l , except that the group shall not be written.
92046		-p	Write a slash ('/') after each filename if that file is a directory.
92047		-q	Force each instance of non-printable filename characters and <code><tab></code> s to be written as the question-mark ('?') character. Implementations may provide this option by default if the output is to a terminal device.
92048			
92049			
92050		-r	Reverse the order of the sort to get reverse collating sequence oldest first, or smallest file size first depending on the other options given.
92051			
92052		-s	Indicate the total number of file system blocks consumed by each file displayed. If the -k option is also specified, the block size shall be 1 024 bytes; otherwise, the block size is implementation-defined.
92053			
92054			
92055		-t	Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence.
92056			
92057		-u	Use time of last access (see <code><sys/stat.h></code>) instead of last modification of the file for sorting (-t) or writing (-l).
92058			
92059		-x	The same as -C , except that the multi-text-column output is produced with entries sorted across, rather than down, the columns.
92060			
92061		-1	(The numeric digit one.) Force output to be one entry per line.
92062			Specifying more than one of the options in the following mutually-exclusive pairs shall not be considered an error: -C and -l (ell), -m and -l (ell), -x and -l (ell), -C and -1 (one), -H and -L ,
92063			

92064 -c and -u, -t and -S. The last option specified in each pair shall determine the output format.

92065 OPERANDS

92066 The following operand shall be supported:

92067 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic
92068 message shall be output on standard error.

92069 STDIN

92070 Not used.

92071 INPUT FILES

92072 None.

92073 ENVIRONMENT VARIABLES

92074 The following environment variables shall affect the execution of *ls*:

92075 *COLUMNS* Determine the user's preferred column position width for writing multiple text-
92076 column output. If this variable contains a string representing a decimal integer, the
92077 *ls* utility shall calculate how many pathname text columns to write (see -C) based
92078 on the width provided. If *COLUMNS* is not set or invalid, an implementation-
92079 defined number of column positions shall be assumed, based on the
92080 implementation's knowledge of the output device. The column width chosen to
92081 write the names of files in any given directory shall be constant. Filenames shall
92082 not be truncated to fit into the multiple text-column output.

92083 *LANG* Provide a default value for the internationalization variables that are unset or null. |
92084 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
92085 variables used to determine the values of locale categories.)

92086 *LC_ALL* If set to a non-empty string value, override the values of all the other
92087 internationalization variables.

92088 *LC_COLLATE* Determine the locale for character collation information in determining the
92089 pathname collation sequence.

92091 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
92092 characters (for example, single-byte as opposed to multi-byte characters in
92093 arguments) and which characters are defined as printable (character class **print**).

92094 *LC_MESSAGES* Determine the locale that should be used to affect the format and contents of
92095 diagnostic messages written to standard error.
92096

92097 *LC_TIME* Determine the format and contents for date and time strings written by *ls*.

92098 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

92099 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or
92100 null, an unspecified default timezone shall be used.

92101 ASYNCHRONOUS EVENTS

92102 Default.

92103 STDOUT

92104 The default format shall be to list one entry per line to standard output; the exceptions are to
92105 terminals or when one of the -C, -m, or -x options is specified. If the output is to a terminal, the
92106 format is implementation-defined.

92107 When -m is specified, the format used shall be:

92108 "%s, %s, ...\n", <filename1>, <filename2>

- 92109 where the largest number of filenames shall be written without exceeding the length of the line.
- 92110 If the `-i` option is specified, the file's file serial number (see `<sys/stat.h>`) shall be written in the
92111 following format before any other output for the corresponding entry:
- 92112 `%u ", <file serial number>`
- 92113 If the `-l` option is specified without `-L`, the following information shall be written:
- 92114 `"%s %u %s %s %u %s %s\n", <file mode>, <number of links>, <owner name>, <group name>, <number of bytes in the file>, <date and time>, <pathname>`
92115
92116
- 92117 If the file is a symbolic link, this information shall be about the link itself and the `<pathname>`
92118 field shall be of the form:
- 92119 `"%s -> %s", <pathname of link>, <contents of link>`
- 92120 If both `-l` and `-L` are specified, the following information shall be written:
- 92121 `"%s %u %s %s %u %s %s\n", <file mode>, <number of links>, <owner name>, <group name>, <number of bytes in the file>, <date and time>, <pathname of link>`
92122
92123
- 92124 where all fields except `<pathname of link>` shall be for the file resolved from the symbolic link.
- 92125 XSI The `-n`, `-g`, and `-o` options use the same format as `-l`, but with omitted items and their
92126 associated `<blank>`s. See the OPTIONS section.
- 92127 In both the preceding `-l` forms, if `<owner name>` or `<group name>` cannot be determined, or if `-n`
92128 is given, they shall be replaced with their associated numeric values using the format `%u`.
- 92129 The `<date and time>` field shall contain the appropriate date and timestamp of when the file was
92130 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
92131 `date` command:
- 92132 `date "+%b %e %H:%M"`
- 92133 if the file has been modified in the last six months, or:
- 92134 `date "+%b %e %Y"`
- 92135 (where two `<space>`s are used between `%e` and `%Y`) if the file has not been modified in the last
92136 six months or if the modification date is in the future, except that, in both cases, the final
92137 `<newline>` produced by `date` shall not be included and the output shall be as if the `date`
92138 command were executed at the time of the last modification date of the file rather than the
92139 current time. When the `LC_TIME` locale category is not set to the POSIX locale, a different format
92140 and order of presentation of this field may be used.
- 92141 If the file is a character special or block special file, the size of the file may be replaced with
92142 implementation-defined information associated with the device in question.
- 92143 If the pathname was specified as a *file* operand, it shall be written as specified.
- 92144 XSI The file mode written under the `-l`, `-n`, `-g`, and `-o` options shall consist of the following format:
92145 `"%c%s%s%s", <entry type>, <owner permissions>, <group permissions>, <other permissions>, <optional alternate access method flag>`
92146
92147
- 92148 The `<optional alternate access method flag>` shall be the empty string if there is no alternate or
92149 additional access control method associated with the file; otherwise, it shall be a string
92150 containing a single printable character that is not a `<blank>`.
- 92151 The `<entry type>` character shall describe the type of file, as follows:

92152 d Directory.

92153 b Block special file.

92154 c Character special file.

92155 l (ell) Symbolic link.

92156 p FIFO.

92157 – Regular file.

92158 Implementations may add other characters to this list to represent other implementation-defined
92159 file types.

92160 The next three fields shall be three characters each:

92161 <owner permissions>
92162 Permissions for the file owner class (see XBD [Section 4.4](#), on page 96).

92163 <group permissions>
92164 Permissions for the file group class.

92165 <other permissions>
92166 Permissions for the file other class.

92167 Each field shall have three character positions:

92168 1. If 'r', the file is readable; if '-', the file is not readable.

92169 2. If 'w', the file is writable; if '-', the file is not writable.

92170 3. The first of the following that applies:

92171 S If in <owner permissions>, the file is not executable and set-user-ID mode is set. If in
92172 <group permissions>, the file is not executable and set-group-ID mode is set.

92173 s If in <owner permissions>, the file is executable and set-user-ID mode is set. If in
92174 <group permissions>, the file is executable and set-group-ID mode is set.

92175 XSI T If in <other permissions> and the file is a directory, search permission is not granted to
92176 others, and the restricted deletion flag is set.

92177 XSI t If in <other permissions> and the file is a directory, search permission is granted to
92178 others, and the restricted deletion flag is set.

92179 x The file is executable or the directory is searchable.

92180 – None of the attributes of 'S', 's', 'T', 't', or 'x' applies.

92181 Implementations may add other characters to this list for the third character position.
92182 Such additions shall, however, be written in lowercase if the file is executable or
92183 searchable, and in uppercase if it is not.

92184 XSI If any of the **-l**, **-n**, **-s**, **-g**, or **-o** options is specified, each list of files within the directory shall be
92185 preceded by a status line indicating the number of file system blocks occupied by files in the
92186 directory in 512-byte units if the **-k** option is not specified, or 1 024-byte units of the **-k** option is
92187 specified, rounded up to the next integral number of units, if necessary. In the POSIX locale, the
92188 format shall be:

92189 "total %u\n", <number of units in the directory>

92190 If more than one directory, or a combination of non-directory files and directories are written,
92191 either as a result of specifying multiple operands, or the **-R** option, each list of files within a
92192 directory shall be preceded by:

92193 "\n%s:\n", <directory name>

92194 If this string is the first thing to be written, the first <newline> shall not be written. This output
92195 shall precede the number of units in the directory.

92196 If the `-s` option is given, each file shall be written with the number of blocks used by the file. -
92197 XSI Along with `-C`, `-l`, `-m`, or `-x`, the number and a <space> shall precede the filename; with `-l`, `-n`, -
92198 `-g`, or `-o`, they shall precede each line describing a file. +

STDERR

92199 The standard error shall be used only for diagnostic messages.
92200

OUTPUT FILES

92201 None.
92202

EXTENDED DESCRIPTION

92203 None.
92204

EXIT STATUS

92205 The following exit values shall be returned:
92206

92207 0 Successful completion.

92208 >0 An error occurred.

CONSEQUENCES OF ERRORS

92209 Default.
92210

APPLICATION USAGE

92211 Many implementations use the equal sign (`'='`) to denote sockets bound to the file system for
92212 the `-F` option. Similarly, many historical implementations use the `'s'` character to denote
92213 sockets as the entry type characters for the `-l` option.
92214

92215 It is difficult for an application to use every part of the file modes field of `ls -l` in a portable
92216 manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as
92217 implementations may have extensions. Applications can use this field to pass directly to a user
92218 printout or prompt, but actions based on its contents should generally be deferred, instead, to
92219 the *test* utility.

92220 The output of `ls` (with the `-l` and related options) contains information that logically could be
92221 used by utilities such as *chmod* and *touch* to restore files to a known state. However, this
92222 information is presented in a format that cannot be used directly by those utilities or be easily
92223 translated into a format that can be used. A character has been added to the end of the
92224 permissions string so that applications at least have an indication that they may be working in
92225 an area they do not understand instead of assuming that they can translate the permissions
92226 string into something that can be used. Future versions or related documents may define one or
92227 more specific characters to be used based on different standard additional or alternative access
92228 control mechanisms.

92229 As with many of the utilities that deal with filenames, the output of `ls` for multiple files or in one
92230 of the long listing formats must be used carefully on systems where filenames can contain
92231 embedded white space. Systems and system administrators should institute policies and user
92232 training to limit the use of such filenames.

92233 The number of disk blocks occupied by the file that it reports varies depending on underlying
92234 file system type, block size units reported, and the method of calculating the number of blocks.
92235 On some file system types, the number is the actual number of blocks occupied by the file
92236 (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the
92237 file size (usually making an allowance for indirect blocks, but ignoring holes).

EXAMPLES

An example of a small directory tree being fully listed with `ls -laRF a` in the POSIX locale:

```
total 11
drwxr-xr-x  3 fox      prog           64 Jul  4 12:07 ./
drwxrwxrwx  4 fox      prog          3264 Jul  4 12:09 ../
drwxr-xr-x  2 fox      prog           48 Jul  4 12:07 b/
-rwxr--r--  1 fox      prog           572 Jul  4 12:07 foo*

a/b:
total 4
drwxr-xr-x  2 fox      prog           48 Jul  4 12:07 ./
drwxr-xr-x  3 fox      prog           64 Jul  4 12:07 ../
-rw-r--r--  1 fox      prog           700 Jul  4 12:07 bar
```

RATIONALE

Some historical implementations of the `ls` utility show all entries in a directory except dot and dot-dot when a superuser invokes `ls` without specifying the `-a` option. When “normal” users invoke `ls` without specifying `-a`, they should not see information about any files with names beginning with a period unless they were named as *file* operands.

Implementations are expected to traverse arbitrary depths when processing the `-R` option. The only limitation on depth should be based on running out of physical storage for keeping track of untraversed directories.

The `-1` (one) option was historically found in BSD and BSD-derived implementations only. It is required in this volume of POSIX.1-200x so that conforming applications might ensure that output is one entry per line, even if the output is to a terminal.

The `-S` option was added in Issue 7, but had been provided by several implementations for many years. The description given in the standard documents historic practice, but does not match much of the documentation that described its behavior. Historical documentation typically described it as something like:

```
-S      Sort by size (largest size first) instead of by name. Special character devices (listed last) are sorted by name.
```

even though the file type was never considered when sorting the output. Character special files do typically sort close to the end of the list because their file size on most implementations is zero. But they are sorted alphabetically with any other files that happen to have the same file size (zero), not sorted separately and added to the end.

Generally, this volume of POSIX.1-200x is silent about what happens when options are given multiple times. In the cases of `-C`, `-l`, and `-1`, however, it does specify the results of these overlapping options. Since `ls` is one of the most aliased commands, it is important that the implementation perform intuitively. For example, if the alias were:

```
alias ls="ls -C"
```

and the user typed `ls -1`, single-text-column output should result, not an error.

Earlier versions of this standard did not describe the BSD `-A` option (like `-a`, but dot and dot-dot are not written out). It has been added due to widespread implementation.

Implementations may make `-q` the default for terminals to prevent trojan horse attacks on terminals with special escape sequences. This is not required because:

- Some control characters may be useful on some terminals; for example, a system might write them as `"\001"` or `"^A"`.

- Special behavior for terminals is not relevant to applications portability.

An early proposal specified that the *<optional alternate access method flag>* had to be '+' if there was an alternate access method used on the file or <space> if there was not. This was changed to be <space> if there is not and a single printable character if there is. This was done for three reasons:

1. There are historical implementations using characters other than '+'.
2. There are implementations that vary this character used in that position to distinguish between various alternate access methods in use.
3. The standard developers did not want to preclude future specifications that might need a way to specify more than one alternate access method.

Nonetheless, implementations providing a single alternate access method are encouraged to use '+'.

Earlier versions of this standard did not have the `-k` option, which meant that the `-s` option could not be used portably as its block size was implementation-defined, and the units used to specify the number of blocks occupied by files in a directory in an `ls -l` listing were fixed as 512-byte units. The `-k` option has been added to provide a way for the `-s` option to be used portably, and for consistency it also changes the aforementioned units from 512-byte to 1024-byte.

The *<date and time>* field in the `-l` format is specified only for the POSIX locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this volume of POSIX.1-200x, as the appropriate vehicle is a messaging system; that is, the format should be specified as a "message".

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *find*

XBD [Section 4.4](#) (on page 96), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [<sys/stat.h>](#)

XSH [fstatat\(\)](#)

CHANGE HISTORY

First released in Issue 2.

Issue 5

A second FUTURE DIRECTION is added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the `-F` option, other symbols are allowed for other file types.

Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

The Open Group Base Resolution bwg2001-010 is applied, adding the `T` and `t` fields as part of the `XSI` option.

Issue 7

Austin Group Interpretation 1003.1-2001 #101 is applied, clarifying the optional alternate access method flag in the `STDOUT` section.

SD5-XCU-ERN-50 is applied, adding the `-A` option.

SD5-XCU-ERN-97 is applied, updating the `SYNOPSIS`.

92327

92328

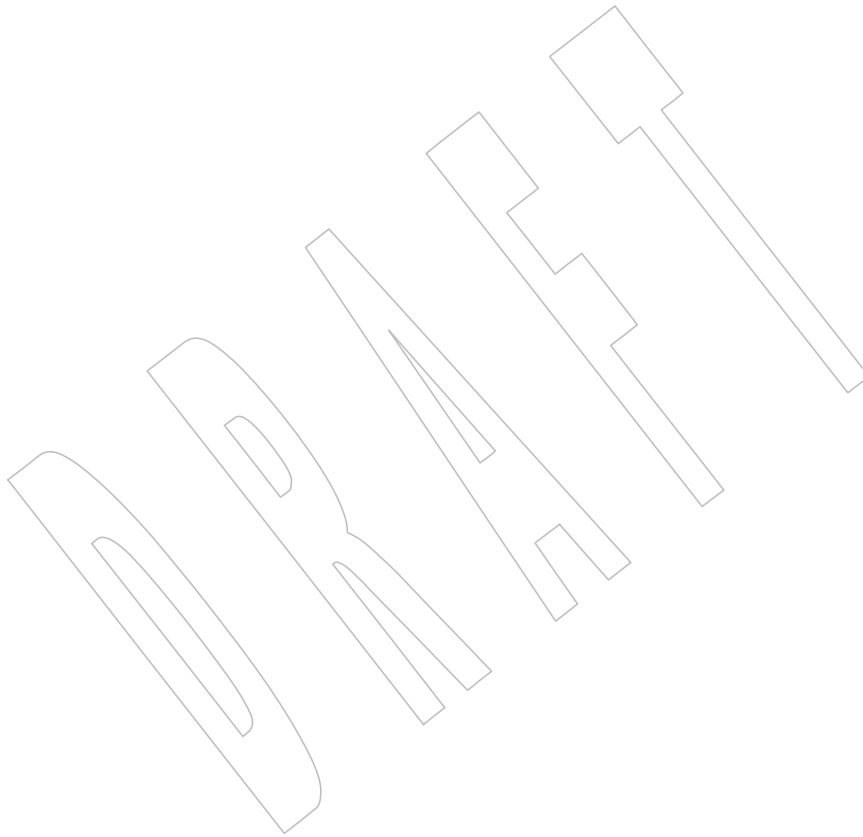
92329

92330

The **-S** option is added from The Open Group Technical Standard, 2006, Extended API Set Part 1.

The **-f**, **-m**, **-n**, **-p**, **-s**, and **-x** options are moved from the XSI option to the Base.

The description of the **-f** and **-s** options are revised and the **-k** option is added.



92331 **NAME**
 92332 m4 — macro processor

92333 **SYNOPSIS**
 92334 m4 [-s] [-D name[=val]]... [-U name]... file...

92335 **DESCRIPTION**
 92336 The *m4* utility is a macro processor that shall read one or more text files, process them according
 92337 to their included macro statements, and write the results to standard output.

92338 **OPTIONS**
 92339 The *m4* utility shall conform to XBD [Section 12.2](#) (on page 201), except that the order of the **-D**
 92340 and **-U** options shall be significant, and options can be interspersed with operands.

92341 The following options shall be supported:

92342 **-s** Enable line synchronization output for the *c99* preprocessor phase (that is, **#line**
 92343 directives).

92344 **-D name[=val]**
 92345 Define *name* to *val* or to null if *=val* is omitted.

92346 **-U name** Undefine *name*.

92347 **OPERANDS**
 92348 The following operand shall be supported:

92349 *file* A pathname of a text file to be processed. If no *file* is given, or if it is '-', the
 92350 standard input shall be read.

92351 **STDIN**
 92352 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.

92353 **INPUT FILES**
 92354 The input file named by the *file* operand shall be a text file.

92355 **ENVIRONMENT VARIABLES**
 92356 The following environment variables shall affect the execution of *m4*:

92357 **LANG** Provide a default value for the internationalization variables that are unset or null.
 92358 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 92359 variables used to determine the values of locale categories.)

92360 **LC_ALL** If set to a non-empty string value, override the values of all the other
 92361 internationalization variables.

92362 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 92363 characters (for example, single-byte as opposed to multi-byte characters in
 92364 arguments and input files).

92365 **LC_MESSAGES**
 92366 Determine the locale that should be used to affect the format and contents of
 92367 diagnostic messages written to standard error.

92368 **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

92369 **ASYNCHRONOUS EVENTS**
 92370 Default.

92371 **STDOUT**

92372 The standard output shall be the same as the input files, after being processed for macro
92373 expansion.

92374 **STDERR**

92375 The standard error shall be used to display strings with the **errprint** macro, macro tracing
92376 enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for
92377 diagnostic messages.

92378 **OUTPUT FILES**

92379 None.

92380 **EXTENDED DESCRIPTION**

92381 The *m4* utility shall compare each token from the input against the set of built-in and user-
92382 defined macros. If the token matches the name of a macro, then the token shall be replaced by
92383 the macro's defining text, if any, and rescanned for matching macro names. Once no portion of
92384 the token matches the name of a macro, it shall be written to standard output. Macros may have
92385 arguments, in which case the arguments shall be substituted into the defining text before it is
92386 rescanned.

92387 Macro calls have the form:

92388 *name*(*arg1*, *arg2*, ..., *argn*)

92389 Macro names shall consist of letters, digits, and underscores, where the first character is not a
92390 digit. Tokens not of this form shall not be treated as macros.

92391 The application shall ensure that the left parenthesis immediately follows the name of the macro.
92392 If a token matching the name of a macro is not followed by a left parenthesis, it is handled as a
92393 use of that macro without arguments.

92394 If a macro name is followed by a left parenthesis, its arguments are the comma-separated tokens
92395 between the left parenthesis and the matching right parenthesis. Unquoted <blank>s and
92396 <newline>s preceding each argument shall be ignored. All other characters, including trailing
92397 <blank>s and <newline>s, are retained. Commas enclosed between left and right parenthesis
92398 characters do not delimit arguments.

92399 Arguments are positionally defined and referenced. The string "\$1" in the defining text shall be
92400 replaced by the first argument. Systems shall support at least nine arguments; only the first nine
92401 can be referenced, using the strings "\$1" to "\$9", inclusive. The string "\$0" is replaced with
92402 the name of the macro. The string "\$#" is replaced by the number of arguments as a string. The
92403 string "\$*" is replaced by a list of all of the arguments, separated by commas. The string "\$@"
92404 is replaced by a list of all of the arguments separated by commas, and each argument is quoted
92405 using the current left and right quoting strings. The string "\${" produces unspecified behavior.

92406 If fewer arguments are supplied than are in the macro definition, the omitted arguments are
92407 taken to be null. It is not an error if more arguments are supplied than are in the macro
92408 definition.

92409 No special meaning is given to any characters enclosed between matching left and right quoting
92410 strings, but the quoting strings are themselves discarded. By default, the left quoting string
92411 consists of a grave accent (`) and the right quoting string consists of an acute accent ('); see
92412 also the **changequote** macro.

92413 Comments are written but not scanned for matching macro names; by default, the begin-
92414 comment string consists of the number sign character and the end-comment string consists of a
92415 <newline>. See also the **changecom** and **dnl** macros.

92416 The *m4* utility shall make available the following built-in macros. They can be redefined, but
92417 once this is done the original meaning is lost. Their values shall be null unless otherwise stated.
92418 In the descriptions below, the term *defining text* refers to the value of the macro: the second

92419		argument to the define macro, among other things. Except for the first argument to the eval
92420		macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The
92421		string values produced as the defining text of the decr , divnum , incr , index , len , and sysval
92422		built-in macros shall be in the form of a decimal-constant as defined in the C language.
92423	changecom	The changecom macro shall set the begin-comment and end-comment strings.
92424		With no arguments, the comment mechanism shall be disabled. With a single non-
92425		null argument, that argument shall become the begin-comment and the <newline>
92426		shall become the end-comment string. With two non-null arguments, the first
92427		argument shall become the begin-comment string and the second argument shall
92428		become the end-comment string. The behavior is unspecified if either argument is
92429		provided but null. Systems shall support comment strings of at least five
92430		characters.
92431	changequote	The changequote macro shall set the begin-quote and end-quote strings. With no
92432		arguments, the quote strings shall be set to the default values (that is, ` `'). The
92433		behavior is unspecified if there is a single argument or either argument is null.
92434		With two non-null arguments, the first argument shall become the begin-quote
92435		string and the second argument shall become the end-quote string. Systems shall
92436		support quote strings of at least five characters.
92437	decr	The defining text of the decr macro shall be its first argument decremented by 1. It
92438		shall be an error to specify an argument containing any non-numeric characters. +
92439		The behavior is unspecified if decr is not immediately followed by a left +
92440		parenthesis.
92441	define	The second argument shall become the defining text of the macro whose name is
92442		the first argument. It is unspecified whether the define macro deletes all prior
92443		definitions of the macro named by its first argument or preserves all but the
92444		current definition of the macro. The behavior is unspecified if define is not +
92445		immediately followed by a left parenthesis.
92446	defn	The defining text of the defn macro shall be the quoted definition (using the
92447		current quoting strings) of its arguments. The behavior is unspecified if defn is not
92448		immediately followed by a left parenthesis.
92449	divert	The <i>m4</i> utility maintains nine temporary buffers, numbered 1 to 9, inclusive.
92450		When the last of the input has been processed, any output that has been placed in
92451		these buffers shall be written to standard output in buffer-numerical order. The
92452		divert macro shall divert future output to the buffer specified by its argument.
92453		Specifying no argument or an argument of 0 shall resume the normal output
92454		process. Output diverted to a stream with a negative number shall be discarded.
92455		Behavior is implementation-defined if a stream number larger than 9 is specified. It
92456		shall be an error to specify an argument containing any non-numeric characters.
92457	divnum	The defining text of the divnum macro shall be the number of the current output
92458		stream as a string.
92459	dnl	The dnl macro shall cause <i>m4</i> to discard all input characters up to and including
92460		the next <newline>.
92461	dumpdef	The dumpdef macro shall write the defined text to standard error for each of the
92462		macros specified as arguments, or, if no arguments are specified, for all macros.
92463	errprint	The errprint macro shall write its arguments to standard error. The behavior is
92464		unspecified if errprint is not immediately followed by a left parenthesis.
92465	eval	The eval macro shall evaluate its first argument as an arithmetic expression, using
92466		signed integer arithmetic with at least 32-bit precision. At least the following C-
92467		language operators shall be supported, with precedence, associativity, and

92468		behavior as described in Section 1.1.2.1 (on page 2231):	
92469		()	
92470		unary +	
92471		unary -	
92472		~	
92473		!	
92474		binary *	
92475		/	
92476		%	
92477		binary +	
92478		binary -	
92479		<<	
92480		>>	
92481		<	
92482		<=	
92483		>	
92484		>=	
92485		= =	
92486		!=	
92487		binary &	
92488		^	
92489			
92490		&&	
92491			
92492		Systems shall support octal and hexadecimal numbers as in the ISO C standard.	
92493		The second argument, if specified, shall set the radix for the result; if the argument	
92494		is blank or unspecified, the default is 10. Behavior is unspecified if the radix falls	
92495		outside the range 2 to 36, inclusive. The third argument, if specified, sets the	
92496		minimum number of digits in the result. Behavior is unspecified if the third	
92497		argument is less than zero. It shall be an error to specify the second or third	
92498		argument containing any non-numeric characters.	
92499	ifdef	If the first argument to the ifdef macro is defined, the defining text shall be the	
92500		second argument. Otherwise, the defining text shall be the third argument, if	
92501		specified, or the null string, if not. The behavior is unspecified if ifdef is not	
92502		immediately followed by a left parenthesis.	
92503	ifelse	The ifelse macro takes three or more arguments. If the first two arguments	
92504		compare as equal strings (after macro expansion of both arguments), the defining	
92505		text shall be the third argument. If the first two arguments do not compare as equal	
92506		strings and there are three arguments, the defining text shall be null. If the first two	
92507		arguments do not compare as equal strings and there are four or five arguments,	
92508		the defining text shall be the fourth argument. If the first two arguments do not	
92509		compare as equal strings and there are six or more arguments, the first three	
92510		arguments shall be discarded and processing shall restart with the remaining	
92511		arguments. The behavior is unspecified if ifelse is not immediately followed by a	
92512		left parenthesis.	
92513	include	The defining text for the include macro shall be the contents of the file named by	
92514		the first argument. It shall be an error if the file cannot be read. The behavior is	
92515		unspecified if include is not immediately followed by a left parenthesis.	
92516	incr	The defining text of the incr macro shall be its first argument incremented by 1. It	
92517		shall be an error to specify an argument containing any non-numeric characters.	+
92518		The behavior is unspecified if incr is not immediately followed by a left	+

92519		parenthesis.	
92520	index	The defining text of the index macro shall be the first character position (as a string) in the first argument where a string matching the second argument begins (zero origin), or -1 if the second argument does not occur. The behavior is unspecified if index is not immediately followed by a left parenthesis.	+
92521			
92522			
92523			
92524	len	The defining text of the len macro shall be the length (as a string) of the first argument. The behavior is unspecified if len is not immediately followed by a left parenthesis.	+
92525			
92526			
92527	m4exit	Exit from the <i>m4</i> utility. If the first argument is specified, it is the exit code. The default is zero. It shall be an error to specify an argument containing any non-numeric characters.	
92528			
92529			
92530	m4wrap	The first argument shall be processed when EOF is reached. If the m4wrap macro is used multiple times, the arguments specified shall be processed in the order in which the m4wrap macros were processed. The behavior is unspecified if m4wrap is not immediately followed by a left parenthesis.	
92531			
92532			
92533			
92534	OB maketemp	The defining text shall be the first argument, with any trailing 'X' characters replaced with the current process ID as a string. The behavior is unspecified if maketemp is not immediately followed by a left parenthesis.	+
92535			
92536			
92537	mkstemp	The first argument shall be taken as a template for creating an empty file, with trailing 'X' characters replaced with characters from the portable filename character set. The behavior is unspecified if the first argument does not end in at least six 'X' characters. If a temporary file is successfully created, then the defining text of the macro shall be the name of the new file. The user ID of the file shall be set to the effective user ID of the process. The group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process. The file access permission bits are set such that only the owner can both read and write the file, regardless of the current <i>umask</i> of the process. If a file could not be created, the defining text of the macro shall be the empty string.	+
92538			
92539			
92540			
92541			
92542			
92543			
92544			
92545			
92546			
92547	popdef	The popdef macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined. The behavior is unspecified if popdef is not immediately followed by a left parenthesis.	
92548			
92549			
92550			
92551	pushdef	The pushdef macro shall be equivalent to the define macro with the exception that it shall preserve any current definition for future retrieval using the popdef macro. The behavior is unspecified if pushdef is not immediately followed by a left parenthesis.	
92552			
92553			
92554			
92555	shift	The defining text for the shift macro shall be all of its arguments except for the first one. The behavior is unspecified if shift is not immediately followed by a left parenthesis.	+
92556			
92557			
92558	sinclude	The sinclude macro shall be equivalent to the include macro, except that it shall not be an error if the file is inaccessible. The behavior is unspecified if sinclude is not immediately followed by a left parenthesis.	+
92559			
92560			
92561	substr	The defining text for the substr macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters. The	+
92562			
92563			
92564			
92565			
92566			
92567			

92568		behavior is unspecified if substr is not immediately followed by a left parenthesis.
92569	syscmd	The syscmd macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. The string result shall not be rescanned for macros while setting the defining text. No output redirection shall be performed by the <i>m4</i> utility. The exit status value from the command can be retrieved using the sysval macro. The behavior is unspecified if syscmd is not immediately followed by a left parenthesis.
92570		
92571		
92572		
92573		
92574		
92575	sysval	The defining text of the sysval macro shall be the exit value of the utility last invoked by the syscmd macro (as a string).
92576		
92577	traceon	The traceon macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.
92578		
92579		
92580	traceoff	The traceoff macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.
92581		
92582	translit	The defining text of the translit macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument. The behavior is unspecified if the '-' character appears within the second or third argument anywhere besides the first or last character. The behavior is unspecified if translit is not immediately followed by a left parenthesis.
92583		
92584		
92585		
92586		
92587		
92588	undefine	The undefine macro shall delete all definitions (including those preserved using the pushdef macro) of the macros named by its arguments. The behavior is unspecified if undefine is not immediately followed by a left parenthesis.
92589		
92590		
92591	undivert	The undivert macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. The behavior is unspecified if an argument contains any non-numeric characters.
92592		
92593		
92594		
92595		

EXIT STATUS

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred

If the **m4exit** macro is used, the exit value can be specified by the input file.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **defn** macro is useful for renaming macros, especially built-ins.

Since **eval** defers to the ISO C standard, some operations have undefined behavior. In some implementations, division or remainder by zero cause a fatal signal, even if the division occurs on the short-circuited branch of "&&" or "||". Any operation that overflows in signed arithmetic produces undefined behavior. Likewise, using the **shift** operators with a shift amount that is not positive and smaller than the precision is undefined, as is shifting a negative number to the right. Historically, not all implementations obeyed C-language precedence rules: '^' and '!' were lower than '=='; '==' and '!=' were not lower than '<'; and '|' was not lower than '^'; the liberal use of "(" can force the desired precedence even with these non-compliant implementations. Furthermore, some traditional implementations treated '^' as an exponentiation operator, although most implementations now use "***" as an extension for this

92615 purpose. +

92616 When a macro has been multiply defined via the **pushdef** macro, it is unspecified whether the

92617 **define** macro will alter only the most recent definition (as though by **popdef** and **pushdef**), or

92618 replace the entire stack of definitions with a single definition (as though by **undefine** and

92619 **pushdef**). An application desiring particular behavior for the **define** macro in this case can

92620 redefine it accordingly.

92621 Applications should use the **mkstemp** macro instead of the obsolescent **maketemp** macro for +

92622 creating temporary files.

92623 EXAMPLES

92624 If the file **m4src** contains the lines:

```
92625 The value of `VER' is "VER".
92626 #ifdef `VER', `VER' is defined to be VER., VER is not defined.)
92627 #ifndef `VER', 1, `VER' is `VER'.)
92628 #ifndef `VER', 2, `VER' is `VER'., `VER' is not 2.)
92629 #endif
```

92630 then the command

```
92631 m4 m4src
```

92632 or the command:

```
92633 m4 -U VER m4src
```

92634 produces the output:

```
92635 The value of VER is "VER".
92636 VER is not defined.
92637 VER is not 2.
92638 #endif
```

92639 The command:

```
92640 m4 -D VER m4src
```

92641 produces the output:

```
92642 The value of VER is ".
92643 VER is defined to be .
92644 VER is not 2.
92645 #endif
```

92646 The command:

```
92647 m4 -D VER=1 m4src
```

92648 produces the output:

```
92649 The value of VER is "1".
92650 VER is defined to be 1.
92651 VER is 1.
92652 VER is not 2.
92653 #endif
```

92654 The command:

```
92655 m4 -D VER=2 m4src
```

92656 produces the output:

92657 The value of VER is "2".
 92658 VER is defined to be 2.
 92659 VER is 2.
 92660 end

RATIONALE

92661 Historic System V-based behavior treated "\${" in a macro definition as two literal characters.
 92662 However, this sequence is left unspecified so that implementations may offer extensions such as
 92663 "\${11}" meaning the eleventh positional parameter. Macros can still be defined with
 92664 appropriate uses of nested quoting to result in a literal "\${" in the output after rescanning
 92665 removes the nested quotes.
 92666

92667 In the **translit** built-in, historic System V-based behavior treated '-' as a literal; GNU behavior
 92668 treats it as a range. This version of the standard allows either behavior.

FUTURE DIRECTIONS

92669 None.

SEE ALSO

92671 *c99*
 92672
 92673 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

CHANGE HISTORY

92674 First released in Issue 2.

Issue 5

92676 The phrase "the defined text for macros written by the **dumpdef** macro" is added to the
 92677 description of STDERR, and the description of **dumpdef** is updated to indicate that output is
 92678 written to standard error. The description of **eval** is updated to indicate that the list of excluded
 92679 C operators excludes unary '&' and '.'. In the description of **ifdef**, the phrase "and it is not
 92680 defined to be zero" is deleted.
 92681

Issue 6

92682 In the EXTENDED DESCRIPTION, the **eval** text is updated to include a '&' character in the
 92683 excepted list.
 92684
 92685 The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion
 92686 buffers.
 92687 The normative text is reworded to avoid use of the term "must" for application requirements.
 92688 The Open Group Base Resolution bwg2000-006 is applied.
 92689 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/31 is applied, replacing the EXAMPLES
 92690 section.

Issue 7

92691 Austin Group Interpretation 1003.1-2001 #117 is applied, marking the **maketemp** macro
 92692 obsolescent and adding a new **mkstemp** macro.
 92693
 92694 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not
 92695 apply (options can be interspersed with operands).
 92696 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
 92697 SD5-XCU-ERN-99 is applied, clarifying the definition of the **divert** macro in the EXTENDED
 92698 DESCRIPTION.
 92699 SD5-XCU-ERN-100 is applied, clarifying the definition of the **syscmd** macro in the EXTENDED
 92700 DESCRIPTION.
 92701 SD5-XCU-ERN-101 is applied, clarifying the definition of the **undivert** macro in the EXTENDED

92702	DESCRIPTION.	
92703	SD5-XCU-ERN-111 is applied to the EXTENDED DESCRIPTION, clarifying that the string "\${"	
92704	produces unspecified behavior.	
92705	SD5-XCU-ERN-112 is applied, updating the changequote macro.	
92706	SD5-XCU-ERN-118 is applied, clarifying the definition of the define macro in the EXTENDED	
92707	DESCRIPTION and APPLICATION USAGE sections.	
92708	SD5-XCU-ERN-118 is applied, clarifying the definition of the translit macro in the EXTENDED	
92709	DESCRIPTION and RATIONALE sections.	
92710	SD5-XCU-ERN-130, SD5-XCU-ERN-131, and SD5-XCU-ERN-137 are applied.	
92711	The <i>m4</i> utility is moved from the XSI option to the Base.	



92712 **NAME**

92713 mailx — process messages

92714 **SYNOPSIS**92715 **Send Mode**92716 mailx [-s *subject*] *address...*92717 **Receive Mode**

92718 UP mailx -e

92719 mailx [-HiNn] [-F] [-u *user*]92720 mailx -f [-HiNn] [-F] [*file*]92721 **DESCRIPTION**92722 The *mailx* utility provides a message sending and receiving facility. It has two major modes,
92723 selected by the options used: Send Mode and Receive Mode.92724 On systems that do not support the User Portability Utilities option, an application using *mailx*
92725 shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first
92726 character of one or more lines is tilde ('~'), all characters in the input message shall appear in
92727 the delivered message, but additional characters may be inserted in the message before it is
92728 retrieved.92729 UP On systems supporting the User Portability Utilities option, mail-receiving capabilities and other
92730 interactive features, Receive Mode, described below, also shall be enabled.92731 **Send Mode**92732 Send Mode can be used by applications or users to send messages from the text in standard
92733 input.92734 UP **Receive Mode**92735 Receive Mode is more oriented towards interactive users. Mail can be read and sent in this
92736 interactive mode.92737 When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to
92738 messages. When sending mail, *mailx* allows editing, reviewing, and other modification of the
92739 message as it is entered.92740 Incoming mail shall be stored in one or more unspecified locations for each user, collectively
92741 called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system
92742 mailbox shall be the default place to find new mail. As messages are read, they shall be marked
92743 to be moved to a secondary file for storage, unless specific action is taken. This secondary file is
92744 called the **mbox** and is normally located in the directory referred to by the *HOME* environment
92745 variable (see *MBOX* in the ENVIRONMENT VARIABLES section for a description of this file).
92746 Messages shall remain in this file until explicitly removed. When the -f option is used to read
92747 mail messages from secondary files, messages shall be retained in those files unless specifically
92748 removed. All three of these locations—system mailbox, **mbox**, and secondary file—are referred
92749 to in this section as simply “mailboxes”, unless more specific identification is required.

92750 **OPTIONS**92751 The *mailx* utility shall conform to XBD [Section 12.2](#) (on page 201).92752 The following options shall be supported. (Only the `-s subject` option shall be required on all
92753 systems. The other options are required only on systems supporting the User Portability Utilities
92754 option.)92755 UP `-e` Test for the presence of mail in the system mailbox. The *mailx* utility shall write
92756 nothing and exit with a successful return code if there is mail to read.92757 UP `-f` Read messages from the file named by the *file* operand instead of the system
92758 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from **mbox**
92759 instead of the system mailbox.92760 UP `-F` Record the message in a file named after the first recipient. The name is the login-
92761 name portion of the address found first on the **To:** line in the mail header.
92762 Overrides the **record** variable, if set (see [Internal Variables in mailx](#), on page 2813).92763 UP `-H` Write a header summary only.92764 UP `-i` Ignore interrupts. (See also **ignore**.)92765 UP `-n` Do not initialize from the system default start-up file. See the EXTENDED
92766 DESCRIPTION section.92767 UP `-N` Do not write an initial header summary.92768 `-s subject` Set the **Subject** header field to *subject*. All characters in the *subject* string shall
92769 appear in the delivered message. The results are unspecified if *subject* is longer
92770 than {LINE_MAX} – 10 bytes or contains a <newline>.92771 UP `-u user` Read the system mailbox of the login name *user*. This shall only be successful if
92772 the invoking user has the appropriate privileges to read the system mailbox of that
92773 user.92774 **OPERANDS**

92775 The following operands shall be supported:

92776 *address* Addressee of message. When `-n` is specified and no user start-up files are accessed
92777 (see the EXTENDED DESCRIPTION section), the user or application shall ensure
92778 this is an address to pass to the mail delivery system. Any system or user start-up
92779 files may enable aliases (see **alias** under [Commands in mailx](#), on page 2816) that
92780 may modify the form of *address* before it is passed to the mail delivery system.92781 UP *file* A pathname of a file to be read instead of the system mailbox when `-f` is specified.
92782 The meaning of the *file* option-argument shall be affected by the contents of the
92783 **folder** internal variable; see [Internal Variables in mailx](#) (on page 2813).92784 **STDIN**92785 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the
92786 message to be delivered to the specified addresses. When in Receive Mode, user commands
92787 shall be accepted from *stdin*. If the User Portability Utilities option is not supported, standard
92788 input lines beginning with a tilde ('~') character produce unspecified results.92789 UP If the User Portability Utilities option is supported, then in both Send and Receive Modes,
92790 standard input lines beginning with the escape character (usually tilde ('~')) shall affect
92791 processing as described in [Command Escapes in mailx](#) (on page 2824).

92792 INPUT FILES

92793 When *mailx* is used as described by this volume of POSIX.1-200x, the *file* option-argument (see
 92794 the **-f** option) and the **mbox** shall be text files containing mail messages, formatted as described
 92795 in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a
 92796 file.

92797 ENVIRONMENT VARIABLES

92798 UP Some of the functionality described in this section shall be provided on implementations that
 92799 support the User Portability Utilities option as described in the text, and is not further shaded
 92800 for this option.

92801 The following environment variables shall affect the execution of *mailx*:

92802 *DEAD* Determine the pathname of the file in which to save partial messages in case of
 92803 interrupts or delivery errors. The default shall be **dead.letter** in the directory
 92804 named by the *HOME* variable. The behavior of *mailx* in saving partial messages is
 92805 unspecified if the User Portability Utilities option is not supported and *DEAD* is
 92806 not defined with the value **/dev/null**.

92807 *EDITOR* Determine the name of a utility to invoke when the **edit** (see [Commands in mailx](#),
 92808 on page 2816) or **~e** (see [Command Escapes in mailx](#), on page 2824) command is
 92809 XSI used. The default editor is unspecified. [On XSI-conformant systems it is *ed*.](#) The
 92810 effects of this variable are unspecified if the User Portability Utilities option is not
 92811 supported.

92812 *HOME* Determine the pathname of the user's home directory.

92813 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 92814 (See [XBD Section 8.2](#) (on page 160) for the precedence of internationalization |
 92815 variables used to determine the values of locale categories.)

92816 *LC_ALL* If set to a non-empty string value, override the values of all the other
 92817 internationalization variables.

92818 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 92819 characters (for example, single-byte as opposed to multi-byte characters in
 92820 arguments and input files) and the handling of case-insensitive address and
 92821 header-field comparisons.

92822 *LC_TIME* This variable may determine the format and contents of the date and time strings
 92823 written by *mailx*. This volume of POSIX.1-200x specifies the effects of this variable
 92824 only for systems supporting the User Portability Utilities option.

92825 *LC_MESSAGES*

92826 Determine the locale that should be used to affect the format and contents of
 92827 diagnostic messages written to standard error and informative messages written to
 92828 standard output.

92829 *LISTER* Determine a string representing the command for writing the contents of the
 92830 **folder** directory to standard output when the **folders** command is given (see
 92831 **folders** in [Commands in mailx](#), on page 2816). Any string acceptable as a
 92832 *command_string* operand to the *sh -c* command shall be valid. If this variable is null
 92833 or not set, the output command shall be *ls*. The effects of this variable are
 92834 unspecified if the User Portability Utilities option is not supported.

92835 *MAILRC* Determine the pathname of the start-up file. The default shall be **.mailrc** in the
 92836 directory referred to by the *HOME* environment variable. The behavior of *mailx* is
 92837 unspecified if the User Portability Utilities option is not supported and *MAILRC* is
 92838 not defined with the value **/dev/null**.

92839		<i>MBOX</i>	Determine a pathname of the file to save messages from the system mailbox that have been read. The exit command shall override this function, as shall saving the message explicitly in another file. The default shall be mbox in the directory named by the <i>HOME</i> variable. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
92840			
92841			
92842			
92843			
92844	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
92845		<i>PAGER</i>	Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a <i>command_string</i> operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the <i>mailx</i> internal variable crf is set to a value less the number of lines in the message; see Internal Variables in mailx (on page 2813). If the <i>PAGER</i> variable is null or not set, the paginator shall be either <i>more</i> or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
92846			
92847			
92848			
92849			
92850			
92851			
92852			
92853			
92854		<i>SHELL</i>	Determine the name of a preferred command interpreter. The default shall be <i>sh</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
92855			
92856			
92857		<i>TERM</i>	If the internal variable screen is not specified, determine the name of the terminal type to indicate in an unspecified manner the number of lines in a screenful of headers. If <i>TERM</i> is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.
92858			
92859			
92860			
92861			
92862		<i>TZ</i>	This variable may determine the timezone used to calculate date and time strings written by <i>mailx</i> . If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
92863			
92864			
92865		<i>VISUAL</i>	Determine a pathname of a utility to invoke when the visual command (see Commands in mailx , on page 2816) or ~v command-escape (see Command Escapes in mailx , on page 2824) is used. If this variable is null or not set, the full-screen editor shall be <i>vi</i> . The effects of this variable are unspecified if the User Portability Utilities option is not supported.
92866			
92867			
92868			
92869			
92870		ASYNCHRONOUS EVENTS	
92871			When <i>mailx</i> is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals.
92872			
92873	UP	In Receive Mode, or in	Send Mode when standard input is a terminal, if a SIGINT signal is received:
92874			
92875	UP	1. If in command mode, the current command, if there is one, shall be aborted, and a	command-mode prompt shall be written.
92876			
92877		2. If in input mode:	
92878	UP	a. If <i>ignore</i> is set, <i>mailx</i> shall write "@\n", discard the current input line, and	continue processing, bypassing the message-abort mechanism described in item
92879			2b.
92880			
92881	UP	b. If the interrupt was received while sending mail, either when in Receive Mode or	in Send Mode, a message shall be written, and another subsequent interrupt, with
92882			no other intervening characters typed, shall be required to abort the mail message.
92883			
92884	UP	If in Receive Mode and another interrupt is received, a command-mode prompt	shall be written. If in Send Mode and another interrupt is received, <i>mailx</i> shall
92885			terminate with a non-zero status.
92886			

92887 In both cases listed in item b, if the message is not empty:

- 92888 UP i. If **save** is enabled and the file named by *DEAD* can be created, the message
92889 shall be written to the file named by *DEAD*. If the file exists, the message
92890 shall be written to replace the contents of the file.
- 92891 UP ii. If **save** is not enabled, or the file named by *DEAD* cannot be created, the
92892 message shall not be saved.

92893 The *mailx* utility shall take the standard action for all other signals.

92894 STDOUT

92895 In command and input modes, all output, including prompts and messages, shall be written to
92896 standard output.

92897 STDERR

92898 The standard error shall be used only for diagnostic messages.

92899 OUTPUT FILES

92900 Various *mailx* commands and command escapes can create or add to files, including the **mbox**,
92901 the dead-letter file, and secondary mailboxes. When *mailx* is used as described in this volume of
92902 POSIX.1-200x, these files shall be text files, formatted as follows:

92903 line beginning with **From**<space>
92904 [one or more *header-lines*; see [Commands in mailx](#) (on page 2816)]
92905 *empty line*
92906 [zero or more *body lines*
92907 *empty line*]
92908 [line beginning with **From**<space>...]

92909 where each message begins with the **From** <space> line shown, preceded by the beginning of
92910 the file or an empty line. (The **From** <space> line is considered to be part of the message header,
92911 but not one of the header-lines referred to in [Commands in mailx](#) (on page 2816); thus, it shall
92912 not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the
92913 **From** <space> line and any additional header lines are unspecified, except that none shall be
92914 empty. The format of a message body line is also unspecified, except that no line following an
92915 empty line shall start with **From** <space>; *mailx* shall modify any such user-entered message
92916 body lines (following an empty line and beginning with **From** <space>) by adding one or more
92917 characters to precede the 'F'; it may add these characters to **From** <space> lines that are not
92918 preceded by an empty line.

92919 When a message from the system mailbox or entered by the user is not a text file, it is
92920 implementation-defined how such a message is stored in files written by *mailx*.

92921 EXTENDED DESCRIPTION

92922 UP The functionality in the entire EXTENDED DESCRIPTION section shall be provided on
92923 implementations supporting the User Portability Utilities option. The functionality described in
92924 this section shall be provided on implementations that support the User Portability Utilities
92925 option (and the rest of this section is not further shaded for this option).

92926 The *mailx* utility need not support for all character encodings in all circumstances. For example,
92927 inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not
92928 be portable to non-internationalized systems, and so on. Under these circumstances, it is
92929 recommended that only characters defined in the ISO/IEC 646:1991 standard International
92930 Reference Version (equivalent to ASCII) 7-bit range of characters be used.

92931 When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of
92932 header-summary lines (if **-N** was not specified and there are messages, see below), followed by
92933 a prompt indicating that *mailx* can accept regular commands (see [Commands in mailx](#), on page
92934 2816); this is termed *command mode*. The page of header-summary lines shall contain the first

92935 new message if there are new messages, or the first unread message if there are unread
 92936 messages, or the first message. When *mailx* is invoked using the Send Mode synopsis and
 92937 standard input is a terminal, if no subject is specified on the command line and the **asksub**
 92938 variable is set, a prompt for the subject shall be written. At this point, *mailx* shall be in input
 92939 mode. This input mode shall also be entered when using one of the Receive Mode synopsis
 92940 forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or
 92941 **mail** commands and standard input is a terminal. When the message is typed and the end of the
 92942 message is encountered, the message shall be passed to the mail delivery software. Commands
 92943 can be entered by beginning a line with the escape character (by default, tilde ('~')) followed by
 92944 a single command letter and optional arguments. See [Commands in mailx](#) (on page 2816) for a
 92945 summary of these commands. It is unspecified what effect these commands will have if standard
 92946 input is not a terminal when a message is entered using either the Send Mode synopsis, or the
 92947 Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

92948 **Note:** For notational convenience, this section uses the default escape character, tilde, in all references
 92949 and examples.

92950 At any time, the behavior of *mailx* shall be governed by a set of environmental and internal
 92951 variables. These are flags and valued parameters that can be set and cleared via the *mailx* **set**
 92952 and **unset** commands.

92953 Regular commands are of the form:

92954 `[command] [msglist] [argument ...]`

92955 If no *command* is specified in command mode, **next** shall be assumed. In input mode, commands
 92956 shall be recognized by the escape character, and lines not treated as commands shall be taken as
 92957 input for the message.

92958 In command mode, each message shall be assigned a sequential number, starting with 1.

92959 All messages have a state that shall affect how they are displayed in the header summary and
 92960 how they are retained or deleted upon termination of *mailx*. There is at any time the notion of a
 92961 *current* message, which shall be marked by a '>' at the beginning of a line in the header
 92962 summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current
 92963 message shall be the first new message, if there is a new message, or the first unread message if
 92964 there is an unread message, or the first message if there are any messages, or unspecified if there
 92965 are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*)
 92966 or an optional single message (*message*) on which to operate shall leave the current message set
 92967 to the highest-numbered message of the messages specified, unless the command deletes
 92968 messages, in which case the current message shall be set to the first undeleted message (that is, a
 92969 message not in the deleted state) after the highest-numbered message deleted by the command,
 92970 if one exists, or the first undeleted message before the highest-numbered message deleted by the
 92971 command, if one exists, or to an unspecified value if there are no remaining undeleted messages.
 92972 All messages shall be in one of the following states:

92973 *new* The message is present in the system mailbox and has not been viewed by the user
 92974 or moved to any other state. Messages in state *new* when *mailx* quits shall be
 92975 retained in the system mailbox.

92976 *unread* The message has been present in the system mailbox for more than one invocation
 92977 of *mailx* and has not been viewed by the user or moved to any other state.
 92978 Messages in state *unread* when *mailx* quits shall be retained in the system mailbox.

92979 *read* The message has been processed by one of the following commands: **f**, **m**, **F**, **M**,
 92980 **copy**, **mbox**, **next**, **pipe**, **print**, **Print**, **top**, **type**, **Type**, **undelete**. The **delete**, **dp**,
 92981 and **dt** commands may also cause the next message to be marked as *read*,
 92982 depending on the value of the **autoprint** variable. Messages that are in the system
 92983 mailbox and in state *read* when *mailx* quits shall be saved in the **mbox**, unless the
 92984 internal variable **hold** was set. Messages that are in the **mbox** or in a secondary

- 92985 mailbox and in state *read* when *mailx* quits shall be retained in their current
92986 location.
- 92987 *deleted* The message has been processed by one of the following commands: **delete**, **dp**, **dt**.
92988 Messages in state *deleted* when *mailx* quits shall be deleted. Deleted messages shall
92989 be ignored until *mailx* quits or changes mailboxes or they are specified to the
92990 undelete command; for example, the message specification */string* shall only
92991 search the subject lines of messages that have not yet been deleted, unless the
92992 command operating on the list of messages is **undelete**. No deleted message or
92993 deleted message header shall be displayed by any *mailx* command other than
92994 **undelete**.
- 92995 *preserved* The message has been processed by a **preserve** command. When *mailx* quits, the
92996 message shall be retained in its current location.
- 92997 *saved* The message has been processed by one of the following commands: **save** or **write**.
92998 If the current mailbox is the system mailbox, and the internal variable **keepsave** is
92999 set, messages in the state *saved* shall be saved to the file designated by the *MBOX*
93000 variable (see the ENVIRONMENT VARIABLES section). If the current mailbox is
93001 the system mailbox, messages in the state *saved* shall be deleted from the current
93002 mailbox, when the **quit** or **file** command is used to exit the current mailbox.
- 93003 The header-summary line for each message shall indicate the state of the message.
- 93004 Many commands take an optional list of messages (*msglist*) on which to operate, which defaults
93005 to the current message. A *msglist* is a list of message specifications separated by <blank>s, which
93006 can include:
- 93007 *n* Message number *n*.
- 93008 **+** The next undeleted message, or the next deleted message for the **undelete** command.
- 93009 **-** The next previous undeleted message, or the next previous deleted message for the
93010 **undelete** command.
- 93011 **.** The current message.
- 93012 **^** The first undeleted message, or the first deleted message for the **undelete** command.
- 93013 **\$** The last message.
- 93014 ***** All messages.
- 93015 *n-m* An inclusive range of message numbers.
- 93016 *address* All messages from *address*; any address as shown in a header summary shall be
93017 matchable in this form.
- 93018 */string* All messages with *string* in the subject line (case ignored).
- 93019 **:c** All messages of type *c*, where *c* shall be one of:
- 93020 **d** Deleted messages.
- 93021 **n** New messages.
- 93022 **o** Old messages (any not in state *read* or *new*).
- 93023 **r** Read messages.
- 93024 **u** Unread messages.
- 93025 Other commands take an optional message (*message*) on which to operate, which defaults to the
93026 current message. All of the forms allowed for *msglist* are also allowed for *message*, but if more
93027 than one message is specified, only the first shall be operated on.

93028 Other arguments are usually arbitrary strings whose usage depends on the command involved.

93029 **Start-Up in mailx**

93030 At start-up time, *mailx* shall take the following steps in sequence:

- 93031 1. Establish all variables at their stated default values.
- 93032 2. Process command line options, overriding corresponding default values.
- 93033 3. Import any of the *DEAD*, *EDITOR*, *MBOX*, *LISTER*, *PAGER*, *SHELL*, or *VISUAL* variables
93034 that are present in the environment, overriding the corresponding default values.
- 93035 4. Read *mailx* commands from an unspecified system start-up file, unless the `-n` option is
93036 given, to initialize any internal *mailx* variables and aliases.
- 93037 5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

93038 Most regular *mailx* commands are valid inside start-up files, the most common use being to set
93039 up initial display options and alias lists. The following commands shall be invalid in the start-up
93040 file: **!**, **edit**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, **visual**, **Copy**, **followup**, and **Followup**.
93041 Any errors in the start-up file shall either cause *mailx* to terminate with a diagnostic message and
93042 a non-zero status or to continue after writing a diagnostic message, ignoring the remainder of
93043 the lines in the start-up file.

93044 A blank line in a start-up file shall be ignored.

93045 **Internal Variables in mailx**

93046 The following variables are internal *mailx* variables. Each internal variable can be set via the
93047 *mailx set* command at any time. The **unset** and **set no name** commands can be used to erase
93048 variables.

93049 In the following list, variables shown as:

93050 `variable`

93051 represent Boolean values. Variables shown as:

93052 `variable=value`

93053 shall be assigned string or numeric values. For string values, the rules in [Commands in mailx](#)
93054 (on page 2816) concerning filenames and quoting shall also apply.

93055 The defaults specified here may be changed by the unspecified system start-up file unless the
93056 user specifies the `-n` option.

93057 **allnet** All network names whose login name components match shall be treated as
93058 identical. This shall cause the *msglist* message specifications to behave similarly.
93059 The default shall be **noallnet**. See also the **alternates** command and the **metoo**
93060 variable.

93061 **append** Append messages to the end of the **mbox** file upon termination instead of placing
93062 them at the beginning. The default shall be **noappend**. This variable shall not
93063 affect the **save** command when saving to **mbox**.

93064 **ask**, **asksub** Prompt for a subject line on outgoing mail if one is not specified on the command
93065 line with the `-s` option. The **ask** and **asksub** forms are synonyms; the system shall
93066 refer to **asksub** and **noasksub** in its messages, but shall accept **ask** and **noask** as
93067 user input to mean **asksub** and **noasksub**. It shall not be possible to set both **ask**
93068 and **noasksub**, or **noask** and **asksub**. The default shall be **asksub**, but no
93069 prompting shall be done if standard input is not a terminal.

93070	askbcc	Prompt for the blind copy list. The default shall be noaskbcc .
93071	askcc	Prompt for the copy list. The default shall be noaskcc .
93072	autoprint	Enable automatic writing of messages after delete and undelete commands. The default shall be noautoprint .
93073		
93074	bang	Enable the special-case treatment of exclamation marks ('!') in escape command lines; see the escape command and Command Escapes in mailx (on page 2824). The default shall be nobang , disabling the expansion of '!' in the <i>command</i> argument to the ~! command and the ~< <i>command</i> escape.
93075		
93076		
93077		
93078	cmd=command	
93079		Set the default command to be invoked by the pipe command. The default shall be nocmd .
93080		
93081	crt=number	Pipe messages having more than <i>number</i> lines through the command specified by the value of the <i>PAGER</i> variable. The default shall be nocrt . If it is set to null, the value used is implementation-defined.
93082		
93083		
93084	XSI debug	Enable verbose diagnostics for debugging. Messages are not delivered. The default shall be nodebug .
93085		
93086	dot	When dot is set, a period on a line by itself during message input from a terminal shall also signify end-of-file (in addition to normal end-of-file). The default shall be nodot . If ignoreeof is set (see below), a setting of nodot shall be ignored and the period is the only method to terminate input mode.
93087		
93088		
93089		
93090	escape=c	Set the command escape character to be the character 'c'. By default, the command escape character shall be tilde. If escape is unset, tilde shall be used; if it is set to null, command escaping shall be disabled.
93091		
93092		
93093	flipr	Reverse the meanings of the R and r commands. The default shall be noflipr .
93094	folder=directory	
93095		The default directory for saving mail files. User-specified filenames beginning with a plus sign ('+') shall be expanded by preceding the filename with this directory name to obtain the real pathname. If <i>directory</i> does not start with a slash ('/'), the contents of <i>HOME</i> shall be prefixed to it. The default shall be nofolder . If folder is unset or set to null, user-specified filenames beginning with '+' shall refer to files in the current directory that begin with the literal '+' character. See also outfolder below. The folder value need not affect the processing of the files named in <i>MBOX</i> and <i>DEAD</i> .
93096		
93097		
93098		
93099		
93100		
93101		
93102		
93103	header	Enable writing of the header summary when entering <i>mailx</i> in Receive Mode. The default shall be header .
93104		
93105	hold	Preserve all messages that are read in the system mailbox instead of putting them in the mbox save file. The default shall be nohold .
93106		
93107	ignore	Ignore interrupts while entering messages. The default shall be noignore .
93108	ignoreeof	Ignore normal end-of-file during message input. Input can be terminated only by entering a period ('.') on a line by itself or by the ~. command escape. The default shall be noignoreeof . See also dot above.
93109		
93110		
93111	indentprefix=string	
93112		A string that shall be added as a prefix to each line that is inserted into the message by the ~m command escape. This variable shall default to one <tab>.
93113		

93114	keep	When a system mailbox, secondary mailbox, or mbox is empty, truncate it to zero length instead of removing it. The default shall be nokeep .
93115		
93116	keepsave	Keep the messages that have been saved from the system mailbox into other files in the file designated by the variable <i>MBOX</i> , instead of deleting them. The default shall be nokeepsave .
93117		
93118		
93119	metoo	Suppress the deletion of the login name of the user from the recipient list when replying to a message or sending to a group. The default shall be nometoo .
93120		
93121	XSI onehop	When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). The default shall be noonehop .
93122		
93123		
93124		
93125		
93126	outfolder	Cause the files used to record outgoing messages to be located in the directory specified by the folder variable unless the pathname is absolute. The default shall be nooutfolder . See the record variable.
93127		
93128		
93129	page	Insert a <form-feed> after each message sent through the pipe created by the pipe command. The default shall be nopage .
93130		
93131	prompt=string	
93132		Set the command-mode prompt to <i>string</i> . If <i>string</i> is null or if noprompt is set, no prompting shall occur. The default shall be to prompt with the string " ? ".
93133		
93134	quiet	Refrain from writing the opening message and version when entering <i>mailx</i> . The default shall be noquiet .
93135		
93136	record=file	Record all outgoing mail in the file with the pathname <i>file</i> . The default shall be norecord . See also outfolder above.
93137		
93138	save	Enable saving of messages in the dead-letter file on interrupt or delivery error. See the variable <i>DEAD</i> for the location of the dead-letter file. The default shall be save .
93139		
93140	screen=number	
93141		Set the number of lines in a screenful of headers for the headers and z commands.
93142		If screen is not specified, a value based on the terminal type identified by the <i>TERM</i> environment variable, the window size, the baud rate, or some combination of these shall be used.
93143		
93144		
93145	sendwait	Wait for the background mailer to finish before returning. The default shall be nosendwait .
93146		
93147	showto	When the sender of the message was the user who is invoking <i>mailx</i> , write the information from the To: line instead of the From: line in the header summary. The default shall be noshowto .
93148		
93149		
93150	sign=string	Set the variable inserted into the text of a message when the ~a command escape is given. The default shall be nosign . The character sequences ' \t ' and ' \n ' shall be recognized in the variable as <tab>s and <newline>s, respectively. (See also ~i in Command Escapes in mailx (on page 2824).)
93151		
93152		
93153		
93154	Sign=string	Set the variable inserted into the text of a message when the ~A command escape is given. The default shall be noSign . The character sequences ' \t ' and ' \n ' shall be recognized in the variable as <tab>s and <newline>s, respectively.
93155		
93156		
93157	toplines=number	
93158		Set the number of lines of the message to write with the top command. The default shall be 5.
93159		

93160

Commands in mailx93161
93162
93163
93164
93165
93166

The following *mailx* commands shall be provided. In the following list, header refers to lines from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines within the header that begin with one or more non-white-space characters, immediately followed by a colon and white space and continuing until the next line beginning with a non-white-space character or an empty line. Header-field refers to the portion of a header line prior to the first colon in that line.

93167
93168
93169
93170
93171

For each of the commands listed below, the command can be entered as the abbreviation (those characters in the Synopsis command word preceding the '['), the full command (all characters shown for the command word, omitting the '[' and ']'), or any truncation of the full command down to the abbreviation. For example, the **exit** command (shown as **ex[it]** in the Synopsis) can be entered as **ex**, **exi**, or **exit**.

93172

The arguments to commands can be quoted, using the following methods:

93173
93174
93175
93176
93177

- An argument can be enclosed between paired double-quotes (" ") or single-quotes (' '); any white space, shell word expansion, or backslash characters within the quotes shall be treated literally as part of the argument. A double-quote shall be treated literally within single-quotes and *vice versa*. These special properties of the quote marks shall occur only when they are paired at the beginning and end of the argument.

93178
93179

- A backslash outside of the enclosing quotes shall be discarded and the following character treated literally as part of the argument.

93180
93181

- An unquoted backslash at the end of a command line shall be discarded and the next line shall continue the command.

93182

Filenames, where expected, shall be subjected to the following transformations, in sequence:

93183
93184
93185
93186

- If the filename begins with an unquoted plus sign, and the **folder** variable is defined (see the **folder** variable), the plus sign shall be replaced by the value of the **folder** variable followed by a slash. If the **folder** variable is unset or is set to null, the filename shall be unchanged.

93187
93188
93189

- Shell word expansions shall be applied to the filename (see [Section 2.6](#), on page 2253). If more than a single pathname results from this expansion and the command is expecting one file, the effects are unspecified.

93190

Declare Aliases93191
93192

Synopsis: a[lias] [alias [address...]]
 g[roup] [alias [address...]]

93193
93194
93195
93196
93197
93198
93199
93200

Add the given addresses to the alias specified by *alias*. The names shall be substituted when *alias* is used as a recipient address specified by the user in an outgoing message (that is, other recipients addressed indirectly through the **reply** command shall not be substituted in this manner). Mail address alias substitution shall apply only when the alias string is used as a full address; for example, when **hlj** is an alias, *hlj@posix.com* does not trigger the alias substitution. If no arguments are given, write a listing of the current aliases to standard output. If only an *alias* argument is given, write a listing of the specified alias to standard output. These listings need not reflect the same order of addresses that were entered.

93201 **Declare Alternatives**93202 *Synopsis:* alt[ernates] name...

93203 (See also the **metoo** variable.) Declare a list of alternative names for the user's login. When
 93204 responding to a message, these names shall be removed from the list of recipients for the
 93205 response. The comparison of names shall be in a case-insensitive manner. With no arguments,
 93206 **alternates** shall write the current list of alternative names.

93207 **Change Current Directory**

93208 *Synopsis:* cd [directory]
 93209 ch[dir] [directory]

93210 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

93211 **Copy Messages**

93212 *Synopsis:* c[opy] [file]
 93213 c[opy] [msglist] file
 93214 C[opy] [msglist]

93215 Copy messages to the file named by the pathname *file* without marking the messages as saved.
 93216 Otherwise, it shall be equivalent to the **save** command.

93217 In the capitalized form, save the specified messages in a file whose name is derived from the
 93218 author of the message to be saved, without marking the messages as saved. Otherwise, it shall
 93219 be equivalent to the **Save** command.

93220 **Delete Messages**93221 *Synopsis:* d[el]ete [msglist]

93222 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see
 93223 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there
 93224 are messages remaining after the **delete** command, the current message shall be written as
 93225 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be
 93226 written.

93227 **Discard Header Fields**

93228 *Synopsis:* di[scard] [header-field...]
 93229 ig[nore] [header-field...]

93230 Suppress the specified header fields when writing messages. Specified *header-fields* shall be
 93231 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and
 93232 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall
 93233 override this command. The comparison of header fields shall be in a case-insensitive manner. If
 93234 no arguments are specified, write a list of the currently suppressed header fields to standard
 93235 output; the listing need not reflect the same order of header fields that were entered.

93236 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

93237

Delete Messages and Display

93238

Synopsis: `dp [msglist]`

93239

`dt [msglist]`

93240

Delete the specified messages as described for the **delete** command, except that the **autoprint** variable shall have no effect, and the current message shall be written only if it was set to a message after the last message deleted by the command. Otherwise, an informational message to the effect that there are no further messages in the mailbox shall be written, followed by the *mailx* prompt.

93241

93242

93243

93244

93245

Echo a String

93246

Synopsis: `ec[ho] string ...`

93247

Echo the given strings, equivalent to the shell *echo* utility.

93248

Edit Messages

93249

Synopsis: `e[dit] [msglist]`

93250

Edit the given messages. The messages shall be placed in a temporary file and the utility named by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is unspecified.

93251

93252

93253

The **edit** command does not modify the contents of those messages in the mailbox.

93254

Exit

93255

Synopsis: `ex[it]`

93256

`x[it]`

93257

Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also **quit**).

93258

93259

Change Folder

93260

Synopsis: `fi[le] [file]`

93261

`fold[er] [file]`

93262

Quit (see the **quit** command) from the current file of messages and read in the file named by the pathname *file*. If no argument is given, the name and status of the current mailbox shall be written.

93263

93264

93265

Several unquoted special characters shall be recognized when used as *file* names, with the following substitutions:

93266

93267

`%` The system mailbox for the invoking user.

93268

`%user` The system mailbox for *user*.

93269

`#` The previous file.

93270

`&` The current **mbox**.

93271

`+file` The named file in the **folder** directory. (See the **folder** variable.)

93272

The default file shall be the current mailbox.

93273

Display List of Folders

93274

Synopsis: *folders*

93275

93276

93277

Write the names of the files in the directory set by the **folder** variable. The command specified by the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).

93278

Follow Up Specified Messages

93279

Synopsis: *fo[llowup] [message]*

93280

F[ollowup] [msglist]

93281

93282

In the lowercase form, respond to a message, recording the response in a file whose name is derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.

93283

93284

93285

93286

In the capitalized form, respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line shall be taken from the first message and the response shall be recorded in a file whose name is derived from the author of the first message. See also the **Save** and **Copy** commands and **outfolder**.

93287

Both forms shall override the **record** variable, if set.

93288

Display Header Summary for Specified Messages

93289

Synopsis: *f[rom] [msglist]*

93290

Write the header summary for the specified messages.

93291

Display Header Summary

93292

Synopsis: *h[eaders] [message]*

93293

93294

93295

93296

93297

Write the page of headers that includes the message specified. If the *message* argument is not specified, the current message shall not change. However, if the *message* argument is specified, the current message shall become the message that appears at the top of the page of headers that includes the message specified. The **screen** variable sets the number of headers per page. See also the **z** command.

93298

Help

93299

Synopsis: *hel[p]*

93300

?

93301

Write a summary of commands.

93302

Hold Messages

93303

Synopsis: *ho[ld] [msglist]*

93304

pre[serve] [msglist]

93305

93306

93307

93308

Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall override any commands that might previously have marked the messages to be deleted. During the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve* marking of a message.

93309 **Execute Commands Conditionally**

93310 *Synopsis:* i[f] s|r
 93311 mail-commands
 93312 el[se]
 93313 mail-commands
 93314 en[dif]

93315 Execute commands conditionally, where **if s** executes the following *mail-commands*, up to an
 93316 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-commands* to be
 93317 executed only in Receive Mode.

93318 **List Available Commands**

93319 *Synopsis:* l[ist]

93320 Write a list of all commands available. No explanation shall be given.

93321 **Mail a Message**

93322 *Synopsis:* m[ail] address...

93323 Mail a message to the specified addresses or aliases.

93324 **Direct Messages to mbox**

93325 *Synopsis:* mb[ox] [msglist]

93326 Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.
 93327 See **MBOX**. See also the **exit** and **quit** commands.

93328 **Process Next Specified Message**

93329 *Synopsis:* n[ext] [message]

93330 If the current message has not been written (for example, by the **print** command) since *mailx*
 93331 started or since any other message was the current message, behave as if the **print** command
 93332 was entered. Otherwise, if there is an undeleted message after the current message, make it the
 93333 current message and behave as if the **print** command was entered. Otherwise, an informational
 93334 message to the effect that there are no further messages in the mailbox shall be written, followed
 93335 by the *mailx* prompt. Should the current message location be the result of an immediately
 93336 preceding **hold**, **mbox**, **preserve**, or **touch** command, **next** will act as if the current message has
 93337 already been written.

93338 **Pipe Message**

93339 *Synopsis:* pi[pe] [[msglist] command]
 93340 | [[msglist] command]

93341 Pipe the messages through the given *command* by invoking the command interpreter specified
 93342 by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) The application shall ensure
 93343 that the command is given as a single argument. Quoting, described previously, can be used to
 93344 accomplish this. If no arguments are given, the current message shall be piped through the
 93345 command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>
 93346 shall be inserted after each message.

93347

Display Message with Headers

93348

Synopsis: P[rint] [msglist]

93349

T[ype] [msglist]

93350

Write the specified messages, including all header lines, to standard output. Override suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable shall be paged through the command specified by the *PAGER* environment variable.

93351

93352

93353

93354

Display Message

93355

Synopsis: p[rint] [msglist]

93356

t[ype] [msglist]

93357

Write the specified messages to standard output. If **crt** is set, the messages longer than the number of lines specified by the **crt** variable shall be paged through the command specified by the *PAGER* environment variable.

93358

93359

93360

Quit

93361

Synopsis: q[uit]

93362

end-of-file

93363

Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages in the mailbox.

93364

93365

93366

93367

Reply to a Message List

93368

Synopsis: R[eply] [msglist]

93369

R[espond] [msglist]

93370

Mail a reply message to the sender of each message in the *msglist*. The subject line shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject from the first message. If **record** is set to a filename, the response shall be saved at the end of that file.

93371

93372

93373

93374

See also the **flpr** variable.

93375

Reply to a Message

93376

Synopsis: r[eply] [message]

93377

r[espond] [message]

93378

Mail a reply message to all recipients included in the header of the message. The subject line shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject from the message. If **record** is set to a filename, the response shall be saved at the end of that file.

93379

93380

93381

93382

See also the **flpr** variable.

93383

Retain Header Fields

93384

Synopsis: ret[ain] [*header-field...*]

93385

Retain the specified header fields when writing messages. This command shall override all **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive manner. If no arguments are specified, write a list of the currently retained header fields to standard output; the listing need not reflect the same order of header fields that were entered.

93386

93387

93388

93389

Save Messages

93390

Synopsis: s[ave] [*file*]

93391

s[ave] [*msglist*] *file*

93392

S[ave] [*msglist*]

93393

Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file* argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be appended to the file. The message shall be put in the state *saved*, and shall behave as specified in the description of the *saved* state when the current mailbox is exited by the **quit** or **file** command.

93394

93395

93396

93397

93398

In the capitalized form, save the specified messages in a file whose name is derived from the author of the first message. The name of the file shall be taken to be the author's name with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and **outfolder** variable.

93399

93400

93401

93402

Set Variables

93403

Synopsis: se[t] [*name*=[*string*]] ...] [*name=number* ...] [*noname* ...]

93404

Define one or more variables called *name*. The variable can be given a null, string, or numeric value. Quoting and backslash escapes can occur anywhere in *string*, as described previously, as if the *string* portion of the argument were the entire argument. The forms *name* and *name=* shall be equivalent to *name=""* for variables that take string values. The **set** command without arguments shall write a list of all defined variables and their values. The **no name** form shall be equivalent to **unset name**.

93405

93406

93407

93408

93409

93410

Invoke a Shell

93411

Synopsis: sh[ell]

93412

Invoke an interactive command interpreter (see also *SHELL*).

93413

Display Message Size

93414

Synopsis: si[ze] [*msglist*]

93415

Write the size in bytes of each of the specified messages.

93416

Read mailx Commands From a File

93417

Synopsis: so[urce] *file*

93418

Read and execute commands from the file named by the pathname *file* and return to command mode.

93419

93420

Display Beginning of Messages

93421

Synopsis: `to[p] [msglist]`

93422

Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to write. The default shall be 5.

93423

93424

Touch Messages

93425

Synopsis: `tou[ch] [msglist]`

93426

Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

93427

93428

Delete Aliases

93429

Synopsis: `una[lias] [alias]...`

93430

Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

93431

Undelete Messages

93432

Synopsis: `u[ndelete] [msglist]`

93433

Change the state of the specified messages from deleted to read. If **autoprint** is set, the last message of those restored shall be written. If *msglist* is not specified, the message shall be selected as follows:

93434

93435

93436

- If there are any deleted messages that follow the current message, the first of these shall be chosen.

93437

93438

- Otherwise, the last deleted message that also precedes the current message shall be chosen.

93439

Unset Variables

93440

Synopsis: `uns[et] name...`

93441

Cause the specified variables to be erased.

93442

Edit Message with Full-Screen Editor

93443

Synopsis: `v[isual] [msglist]`

93444

Edit the given messages with a screen editor. Each message shall be placed in a temporary file, and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The default editor shall be *vi*.

93445

93446

93447

The **visual** command does not modify the contents of those messages in the mailbox.

93448

Write Messages to a File

93449

Synopsis: `w[rite] [msglist] file`

93450

Write the given messages to the file specified by the pathname *file*, minus the message header. Otherwise, it shall be equivalent to the **save** command.

93451

93452

Scroll Header Display

93453

Synopsis: z[+|-]

93454

93455

93456

Scroll the header display forward (if '+' is specified or if no option is specified) or backward (if '-' is specified) one screenful. The number of headers written shall be set by the **screen** variable.

93457

Invoke Shell Command

93458

Synopsis: !command

93459

93460

93461

Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*. (See also *sh -c*.) If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be replaced with the command executed by the previous ! command or ~! command escape.

93462

Null Command

93463

Synopsis: # comment

93464

This null command (comment) shall be ignored by *mailx*.

93465

Display Current Message Number

93466

Synopsis: =

93467

Write the current message number.

93468

Command Escapes in mailx

93469

93470

93471

The following commands can be entered only from input mode, by beginning a line with the escape character (by default, tilde ('~')). See the **escape** variable description for changing this special character. The format for the commands shall be:

93472

```
<escape-character><command-char><separator>[<arguments>]
```

93473

where the *<separator>* can be zero or more *<blank>*s.

93474

93475

93476

93477

In the following descriptions, the application shall ensure that the argument *command* (but not *mailx-command*) is a shell command string. Any string acceptable to the command interpreter specified by the *SHELL* variable when it is invoked as *SHELL -c command_string* shall be valid. The command can be presented as multiple arguments (that is, quoting is not required).

93478

93479

Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send Mode and produce unspecified results.

93480

93481

93482

93483

~! *command* Invoke the command interpreter specified by *SHELL* with two arguments: **-c** and *command*; and then return to input mode. If the **bang** variable is set, each unescaped occurrence of '!' in *command* shall be replaced with the command executed by the previous ! command or ~! command escape.

93484

~. Simulate end-of-file (terminate message input).

93485

93486

~: *mailx-command*, ~_ *mailx-command*

Perform the command-level request.

93487

~? Write a summary of command escapes.

93488

~A This shall be equivalent to ~i **Sign**.

93489

~a This shall be equivalent to ~i **sign**.

- 93490 ~**b** *name...* Add the *names* to the blind carbon copy (**Bcc**) list.
- 93491 ~**c** *name...* Add the *names* to the carbon copy (**Cc**) list.
- 93492 ~**d** Read in the dead-letter file. See *DEAD* for a description of this file.
- 93493 ~**e** Invoke the editor, as specified by the *EDITOR* environment variable, on the partial message.
- 93494
- 93495 ~**f** [*msglist*] Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape also shall insert message headers into the message with field selection affected by the **discard**, **ignore**, and **retain** commands.
- 93496
- 93497
- 93498
- 93499 ~**F** [*msglist*] This shall be the equivalent of the ~**f** command escape, except that all headers shall be included in the message, regardless of previous **discard**, **ignore**, and **retain** commands.
- 93500
- 93501
- 93502 ~**h** If standard input is a terminal, prompt for a **Subject** line and the **To**, **Cc**, and **Bcc** lists. Other implementation-defined headers may also be presented for editing. If the field is written with an initial value, it can be edited as if it had just been typed.
- 93503
- 93504
- 93505 ~**i** *string* Insert the value of the named variable, followed by a <newline>, into the text of the message. If the string is unset or null, the message shall not be changed.
- 93506
- 93507 ~**m** [*msglist*] Insert the specified messages into the message, prefixing non-empty lines with the string in the **indentprefix** variable. This command escape also shall insert message headers into the message, with field selection affected by the **discard**, **ignore**, and **retain** commands.
- 93508
- 93509
- 93510
- 93511 ~**M** [*msglist*] This shall be the equivalent of the ~**m** command escape, except that all headers shall be included in the message, regardless of previous **discard**, **ignore**, and **retain** commands.
- 93512
- 93513
- 93514 ~**p** Write the message being entered. If the message is longer than **crt** lines (see [Internal Variables in mailx](#), on page 2813), the output shall be paginated as described for the *PAGER* variable.
- 93515
- 93516
- 93517 ~**q** Quit (see the **quit** command) from input mode by simulating an interrupt. If the body of the message is not empty, the partial message shall be saved in the dead-letter file. See *DEAD* for a description of this file.
- 93518
- 93519
- 93520 ~**r** *file*, ~< *file*, ~**r** !*command*, ~< !*command*
- 93521 Read in the file specified by the pathname *file*. If the argument begins with an exclamation mark (' ! '), the rest of the string shall be taken as an arbitrary system command; the command interpreter specified by *SHELL* shall be invoked with two arguments: **-c** and *command*. The standard output of *command* shall be inserted into the message.
- 93522
- 93523
- 93524
- 93525
- 93526 ~**s** *string* Set the subject line to *string*.
- 93527
- 93528 ~**t** *name...* Add the given *names* to the **To** list.
- 93529
- 93528 ~**v** Invoke the full-screen editor, as specified by the *VISUAL* environment variable, on the partial message.
- 93529
- 93530 ~**w** *file* Write the partial message, without the header, onto the file named by the pathname *file*. The file shall be created or the message shall be appended to it if the file exists.
- 93531
- 93532
- 93533 ~**x** Exit as with ~**q**, except the message shall not be saved in the dead-letter file.

93534 ~| *command* Pipe the body of the message through the given *command* by invoking the
 93535 command interpreter specified by *SHELL* with two arguments: *-c* and *command*. If
 93536 the *command* returns a successful exit status, the standard output of the command
 93537 shall replace the message. Otherwise, the message shall remain unchanged. If the
 93538 *command* fails, an error message giving the exit status shall be written.

93539 EXIT STATUS

93540 UP When the *-e* option is specified, the following exit values are returned:

93541 0 Mail was found.

93542 >0 Mail was not found or an error occurred.

93543 Otherwise, the following exit values are returned:

93544 0 Successful completion; note that this status implies that all messages were *sent*, but it gives
 93545 no assurances that any of them were actually *delivered*.

93546 >0 An error occurred.

93547 CONSEQUENCES OF ERRORS

93548 UP When in **input mode (Receive Mode)** or Send Mode:

93549 UP • If an error is encountered processing an input line beginning with a tilde (*'~'*) character,
 93550 (see [Command Escapes in mailx](#), on page 2824), a diagnostic message shall be written to
 93551 standard error, and the message being composed may be modified, but this condition shall
 93552 not prevent the message from being sent.

93553 • Other errors shall prevent the sending of the message.

93554 UP When in command mode:

93555 • Default.

93556 APPLICATION USAGE

93557 Delivery of messages to remote systems requires the existence of communication paths to such
 93558 systems. These need not exist.

93559 Input lines are limited to {*LINE_MAX*} bytes, but mailers between systems may impose more
 93560 severe line-length restrictions. This volume of POSIX.1-200x does not place any restrictions on
 93561 the length of messages handled by *mailx*, and for delivery of local messages the only limitations
 93562 should be the normal problems of available disk space for the target mail file. When sending
 93563 messages to external machines, applications are advised to limit messages to less than 100 000
 93564 bytes because some mail gateways impose message-length restrictions.

93565 The format of the system mailbox is intentionally unspecified. Not all systems implement
 93566 system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some
 93567 system mailboxes may be multiple files, others records in a database. The internal format of the
 93568 messages themselves is specified with the historical format from Version 7, but only after the
 93569 messages have been saved in some file other than the system mailbox. This was done so that
 93570 many historical applications expecting text-file mailboxes are not broken.

93571 Some new formats for messages can be expected in the future, probably including binary data,
 93572 bit maps, and various multimedia objects. As described here, *mailx* is not prohibited from
 93573 handling such messages, but it must store them as text files in secondary mailboxes (unless some
 93574 extension, such as a variable or command line option, is used to change the stored format). Its
 93575 method of doing so is implementation-defined and might include translating the data into text
 93576 file-compatible or readable form or omitting certain portions of the message from the stored
 93577 output.

93578 The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain**
 93579 command discards all header-fields except those explicitly retained. The **discard** command
 93580 keeps all header-fields except those explicitly discarded. If headers exist on the retained header
 93581 list, **discard** and **ignore** commands are ignored.

93582 EXAMPLES

93583 None.

93584 RATIONALE

93585 The standard developers felt strongly that a method for applications to send messages to specific
 93586 users was necessary. The obvious example is a batch utility, running non-interactively, that
 93587 wishes to communicate errors or results to a user. However, the actual format, delivery
 93588 mechanism, and method of reading the message are clearly beyond the scope of this volume of
 93589 POSIX.1-200x.

93590 The intent of this command is to provide a simple, portable interface for sending messages non-
 93591 interactively. It merely defines a “front-end” to the historical mail system. It is suggested that
 93592 implementations explicitly denote the sender and recipient in the body of the delivered message.
 93593 Further specification of formats for either the message envelope or the message itself were
 93594 deliberately not made, as the industry is in the midst of changing from the current standards to a
 93595 more internationalized standard and it is probably incorrect, at this time, to require either one.

93596 Implementations are encouraged to conform to the various delivery mechanisms described in
 93597 the CCITT X.400 standards or to the equivalent Internet standards, described in Internet Request
 93598 for Comment (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123.

93599 Many historical systems modified each body line that started with **From** by prefixing the ‘F’
 93600 with ‘>’. It is unnecessary, but allowed, to do that when the string does not follow a blank line
 93601 because it cannot be confused with the next header.

93602 The **edit** and **visual** commands merely edit the specified messages in a temporary file. They do
 93603 not modify the contents of those messages in the mailbox; such a capability could be added as an
 93604 extension, such as by using different command names.

93605 The restriction on a subject line being {LINE_MAX}–10 bytes is based on the historical format
 93606 that consumes 10 bytes for **Subject:** and the trailing <newline>. Many historical mailers that a
 93607 message may encounter on other systems are not able to handle lines that long, however.

93608 Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient
 93609 justification to exclude this utility from this volume of POSIX.1-200x. It is also arguable that it is,
 93610 in fact, testable, but that the tests themselves are not portable.

93611 When *mailx* is being used by an application that wishes to receive the results as if none of the
 93612 User Portability Utilities option features were supported, the *DEAD* environment variable must
 93613 be set to **/dev/null**. Otherwise, it may be subject to the file creations described in *mailx*
 93614 ASYNCHRONOUS EVENTS. Similarly, if the *MAILRC* environment variable is not set to
 93615 **/dev/null**, historical versions of *mailx* and *Mail* read initialization commands from a file before
 93616 processing begins. Since the initialization that a user specifies could alter the contents of
 93617 messages an application is trying to send, such applications must set *MAILRC* to **/dev/null**.

93618 The description of *LC_TIME* uses “may affect” because many historical implementations do not
 93619 or cannot manipulate the date and time strings in the incoming mail headers. Some headers
 93620 found in incoming mail do not have enough information to determine the timezone in which the
 93621 mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal
 93622 form that then is parsed by routines like *strftime()* that can take *LC_TIME* settings into account.
 93623 Changing all these times to a user-specified format is allowed, but not required.

93624 The paginator selected when *PAGER* is null or unset is partially unspecified to allow the System
 93625 V historical practice of using *pg* as the default. Bypassing the pagination function, such as by
 93626 declaring that *cat* is the paginator, would not meet with the intended meaning of this

93627 description. However, any “portable user” would have to set *PAGER* explicitly to get his or her
 93628 preferred paginator on all systems. The paginator choice was made partially unspecified, unlike
 93629 the *VISUAL* editor choice (mandated to be *vi*) because most historical pagers follow a common
 93630 theme of user input, whereas editors differ dramatically.

93631 Options to specify addresses as **cc** (carbon copy) or **bcc** (blind carbon copy) were considered to
 93632 be format details and were omitted.

93633 A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them
 93634 were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an
 93635 appropriate marketing distinction between systems.

93636 In order to conform to the Utility Syntax Guidelines, a solution was required to the optional *file*
 93637 option-argument to **-f**. By making *file* an operand, the guidelines are satisfied and users remain
 93638 portable. However, it does force implementations to support usage such as:

```
93639 mailx -fin mymail.box
```

93640 The **no name** method of unsetting variables is not present in all historical systems, but it is in
 93641 System V and provides a logical set of commands corresponding to the format of the display of
 93642 options from the *mailx set* command without arguments.

93643 The **ask** and **asksub** variables are the names selected by BSD and System V, respectively, for the
 93644 same feature. They are synonyms in this volume of POSIX.1-200x.

93645 The *mailx echo* command was not documented in the BSD version and has been omitted here
 93646 because it is not obviously useful for interactive users.

93647 The default prompt on the System V *mailx* is a question mark, on BSD *Mail* an ampersand. Since
 93648 this volume of POSIX.1-200x chose the *mailx* name, it kept the System V default, assuming that
 93649 BSD users would not have difficulty with this minor incompatibility (that they can override).

93650 The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again,
 93651 since this volume of POSIX.1-200x chose the *mailx* name, it kept the System V default, but allows
 93652 the SunOS user to achieve the desired results using **flpr**, an internal variable in System V *mailx*,
 93653 although it has not been documented in the SVID.

93654 The **indentprefix** variable, the **retain** and **unalias** commands, and the **~F** and **~M** command
 93655 escapes were adopted from 4.3 BSD *Mail*.

93656 The **version** command was not included because no sufficiently general specification of the
 93657 version information could be devised that would still be useful to a portable user. This
 93658 command name should be used by suppliers who wish to provide version information about the
 93659 *mailx* command.

93660 The “implementation-specific (unspecified) system start-up file” historically has been named
 93661 **/etc/mailx.rc**, but this specific name and location are not required.

93662 The intent of the wording for the **next** command is that if any command has already displayed
 93663 the current message it should display a following message, but, otherwise, it should display the
 93664 current message. Consider the command sequence:

```
93665 next 3
93666 delete 3
93667 next
```

93668 where the **autoprint** option was not set. The normative text specifies that the second **next**
 93669 command should display a message following the third message, because even though the
 93670 current message has not been displayed since it was set by the **delete** command, it has been
 93671 displayed since the current message was anything other than message number 3. This does not
 93672 always match historical practice in some implementations, where the command file address
 93673 followed by **next** (or the default command) would skip the message for which the user had

93674 searched.

93675 FUTURE DIRECTIONS

93676 None.

93677 SEE ALSO

93678 [Chapter 2](#) (on page 2245), *ed*, *ls*, *more*, *vi*

93679 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

93680 CHANGE HISTORY

93681 First released in Issue 2.

93682 Issue 5

93683 The description of the *EDITOR* environment variable is changed to indicate that *ed* is the default editor if this variable is not set. In previous issues, this default was not stated explicitly at this point but was implied further down in the text.

93686 The FUTURE DIRECTIONS section is added.

93687 Issue 6

93688 The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- 93690 • The `-F` option is added.
- 93691 • The `allnet`, `debug`, and `sendwait` internal variables are added.
- 93692 • The `C`, `ec`, `fo`, `F`, and `S` *mailx* commands are added.

93693 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating “HOME directory” is replaced by “directory referred to by the HOME environment variable”.

93695 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which includes various clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2:1993 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11, #103, #106, #108, #114, #115, #122, and #129.

93699 The normative text is reworded to avoid use of the term “must” for application requirements.

93700 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

93701 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/32 is applied, applying a change to the EXTENDED DESCRIPTION, raised by IEEE PASC Interpretation 1003.2 #122, which was overlooked in the first version of this standard. |

93704 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/17 is applied, updating the EXTENDED DESCRIPTION (Internal Variables in *mailx*). The system start-up file is changed from “implementation-defined” to “unspecified” for consistency with other text in the EXTENDED DESCRIPTION.

93708 Issue 7

93709 Austin Group Interpretation 1003.1-2001 #089 is applied, clarifying the effect of the *LC_TIME* - environment variable.

93711 Austin Group Interpretation 1003.1-2001 #090 is applied, updating the description of the `next` command.

93713 SD5-XCU-ERN-69 is applied. |

93714 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

93715 Shading to indicate support for the User Portability Utilities option is added. +

93716 **NAME**93717 make — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)93718 **SYNOPSIS**93719 SD make [-einpqrst] [-f *makefile*]. . . [-k|-S] [*macro=value*. . .]
93720 [*target_name*. . .]93721 **DESCRIPTION**93722 The *make* utility shall update files that are derived from other files. A typical case is one where
93723 object files are derived from the corresponding source files. The *make* utility examines time
93724 relationships and shall update those derived files (called targets) that have modified times
93725 earlier than the modified times of the files (called prerequisites) from which they are derived. A
93726 description file (makefile) contains a description of the relationships between files, and the
93727 commands that need to be executed to update the targets to reflect changes in their
93728 prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and
93729 optional commands to be executed when a prerequisite is newer than the target. There are two
93730 types of rule:

- 93731 1.
- Inference rules*
- , which have one target name with at least one period (' . ') and no slash
-
- 93732 (' / ')
-
- 93733 2.
- Target rules*
- , which can have more than one target name

93734 In addition, *make* shall have a collection of built-in macros and inference rules that infer
93735 prerequisite relationships to simplify maintenance of programs.93736 To receive exactly the behavior described in this section, the user shall ensure that a portable
93737 makefile shall:

- 93738 • Include the special target
- .POSIX**
-
- 93739 • Omit any special target reserved for implementations (a leading period followed by
-
- 93740 uppercase letters) that has not been specified by this section

93741 The behavior of *make* is unspecified if either or both of these conditions are not met.93742 **OPTIONS**93743 The *make* utility shall conform to XBD [Section 12.2](#) (on page 201), except for Guideline 9.

93744 The following options shall be supported:

- 93745
- e**
- Cause environment variables, including those with null values, to override macro
-
- 93746 assignments within makefiles.
-
- 93747
- f *makefile***
- Specify a different makefile. The argument
- makefile*
- is a pathname of a description
-
- 93748 file, which is also referred to as the
- makefile*
- . A pathname of '-' shall denote the
-
- 93749 standard input. There can be multiple instances of this option, and they shall be
-
- 93750 processed in the order specified. The effect of specifying the same option-argument
-
- 93751 more than once is unspecified.
-
- 93752
- i**
- Ignore error codes returned by invoked commands. This mode is the same as if the
-
- 93753 special target
- .IGNORE**
- were specified without prerequisites.
-
- 93754
- k**
- Continue to update other targets that do not depend on the current target if a non-
-
- 93755 ignored error occurs while executing the commands to bring a target up-to-date.
-
- 93756
- n**
- Write commands that would be executed on standard output, but do not execute
-
- 93757 them. However, lines with a plus sign ('+') prefix shall be executed. In this mode,
-
- 93758 lines with an at sign ('@ ') character prefix shall be written to standard output.

- 93759 **-p** Write to standard output the complete set of macro definitions and target
93760 descriptions. The output format is unspecified.
- 93761 **-q** Return a zero exit value if the target file is up-to-date; otherwise, return an exit
93762 value of 1. Targets shall not be updated if this option is specified. However, a
93763 makefile command line (associated with the targets) with a plus sign ('+') prefix
93764 shall be executed.
- 93765 **-r** Clear the suffix list and do not use the built-in rules.
- 93766 **-S** Terminate *make* if an error occurs while executing the commands to bring a target
93767 up-to-date. This shall be the default and the opposite of **-k**.
- 93768 **-s** Do not write makefile command lines or touch messages (see **-t**) to standard
93769 output before executing. This mode shall be the same as if the special target
93770 **.SILENT** were specified without prerequisites.
- 93771 **-t** Update the modification time of each target as though a *touch target* had been
93772 executed. Targets that have prerequisites but no commands (see **Target Rules**, on
93773 page 2835), or that are already up-to-date, shall not be touched in this manner.
93774 Write messages to standard output for each target file indicating the name of the
93775 file and that it was touched. Normally, the *makefile* command lines associated with
93776 each target are not executed. However, a command line with a plus sign ('+')
93777 prefix shall be executed.

93778 Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any
93779 options specified on the *make* utility command line. If the **-k** and **-S** options are both specified
93780 on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option
93781 specified shall take precedence. If the **-f** or **-p** options appear in the *MAKEFLAGS* environment
93782 variable, the result is undefined.

93783 OPERANDS

93784 The following operands shall be supported:

93785 *target_name* Target names, as defined in the EXTENDED DESCRIPTION section. If no target is
93786 specified, while *make* is processing the makefiles, the first target that *make*
93787 encounters that is not a special target or an inference rule shall be used.

93788 *macro=value* Macro definitions, as defined in **Macros** (on page 2836).

93789 If the *target_name* and *macro=value* operands are intermixed on the *make* utility command line,
93790 the results are unspecified.

93791 STDIN

93792 The standard input shall be used only if the *makefile* option-argument is '-'. See the INPUT
93793 FILES section.

93794 INPUT FILES

93795 The input file, otherwise known as the makefile, is a text file containing rules, macro definitions,
93796 and comments. See the EXTENDED DESCRIPTION section.

93797 ENVIRONMENT VARIABLES

93798 The following environment variables shall affect the execution of *make*:

93799 **LANG** Provide a default value for the internationalization variables that are unset or null. |
93800 (See XBD **Section 8.2** (on page 160) for the precedence of internationalization |
93801 variables used to determine the values of locale categories.)

93802 **LC_ALL** If set to a non-empty string value, override the values of all the other
93803 internationalization variables.

93804 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 93805 characters (for example, single-byte as opposed to multi-byte characters in
 93806 arguments and input files).

93807 **LC_MESSAGES**
 93808 Determine the locale that should be used to affect the format and contents of
 93809 diagnostic messages written to standard error.

93810 **MAKEFLAGS**
 93811 This variable shall be interpreted as a character string representing a series of
 93812 option characters to be used as the default options. The implementation shall
 93813 accept both of the following formats (but need not accept them when intermixed):

- 93814 • The characters are option letters without the leading hyphens or <blank>
 93815 separation used on a *make* utility command line.
- 93816 • The characters are formatted in a manner similar to a portion of the *make*
 93817 utility command line: options are preceded by hyphens and
 93818 <blank>-separated as described in XBD Section 12.2 (on page 201). The
 93819 *macro=value* macro definition operands can also be included. The difference
 93820 between the contents of **MAKEFLAGS** and the *make* utility command line is
 93821 that the contents of the variable shall not be subjected to the word expansions
 93822 (see Section 2.6, on page 2253) associated with parsing the command line
 93823 values.

93824 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

93825 XSI **PROJECTDIR**
 93826 Provide a directory to be used to search for SCCS files not found in the current
 93827 directory. In all of the following cases, the search for SCCS files is made in the
 93828 directory **SCCS** in the identified directory. If the value of **PROJECTDIR** begins with
 93829 a slash, it shall be considered an absolute pathname; otherwise, the value of
 93830 **PROJECTDIR** is treated as a user name and that user's initial working directory
 93831 shall be examined for a subdirectory **src** or **source**. If such a directory is found, it
 93832 shall be used. Otherwise, the value is used as a relative pathname.

93833 If **PROJECTDIR** is not set or has a null value, the search for SCCS files shall be
 93834 made in the directory **SCCS** in the current directory.

93835 The setting of **PROJECTDIR** affects all files listed in the remainder of this utility
 93836 description for files with a component named **SCCS**.

93837 The value of the **SHELL** environment variable shall not be used as a macro and shall not be
 93838 modified by defining the **SHELL** macro in a makefile or on the command line. All other
 93839 environment variables, including those with null values, shall be used as macros, as defined in
 93840 **Macros** (on page 2836).

93841 ASYNCHRONOUS EVENTS

93842 If not already ignored, *make* shall trap **SIGHUP**, **SIGTERM**, **SIGINT**, and **SIGQUIT** and remove
 93843 the current target unless the target is a directory or the target is a prerequisite of the special
 93844 target **.PRECIOUS** or unless one of the **-n**, **-p**, or **-q** options was specified. Any targets removed
 93845 in this manner shall be reported in diagnostic messages of unspecified format, written to
 93846 standard error. After this cleanup process, if any, *make* shall take the standard action for all other
 93847 signals.

93848 STDOUT

93849 The *make* utility shall write all commands to be executed to standard output unless the **-s** option
 93850 was specified, the command is prefixed with an at sign, or the special target **.SILENT** has either
 93851 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work
 93852 needing to be done, it shall write a message to standard output indicating that no action was

93853 taken. If the `-t` option is present and a file is touched, *make* shall write to standard output a
 93854 message of unspecified format indicating that the file was touched, including the filename of the
 93855 file.

93856 STDERR

93857 The standard error shall be used only for diagnostic messages.

93858 OUTPUT FILES

93859 Files can be created when the `-t` option is present. Additional files can also be created by the
 93860 utilities invoked by *make*.

93861 EXTENDED DESCRIPTION

93862 The *make* utility attempts to perform the actions required to ensure that the specified targets are
 93863 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it
 93864 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively
 93865 ensure that they are up-to-date, processing them in the order in which they appear in the rule.
 93866 The *make* utility shall use the modification times of files to determine whether the corresponding
 93867 targets are out-of-date.

93868 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is
 93869 out-of-date, the commands associated with the target entry shall be executed. If there are no
 93870 commands listed for the target, the target shall be treated as up-to-date.

93871 Makefile Syntax

93872 A makefile can contain rules, macro definitions (see [Macros](#), on page 2836), include lines, and
 93873 comments. There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall
 93874 contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be
 93875 used and the suffix list shall be cleared. Additional rules of both types can be specified in a
 93876 makefile. If a rule is defined more than once, the value of the rule shall be that of the last one
 93877 specified. Macros can also be defined more than once, and the value of the macro is specified in
 93878 [Macros](#) (on page 2836). Comments start with a number sign (`'#'`) and continue until an
 93879 unescaped `<newline>` is reached.

93880 By default, the following files shall be tried in sequence: `./makefile` and `./Makefile`. If neither
 93881 XSI `./makefile` or `./Makefile` are found, other implementation-defined files may also be tried. On
 93882 XSI-conformant systems, the additional files `./s.makefile`, `./SCCS/s.makefile`, `./s.Makefile`, and
 93883 `./SCCS/s.Makefile` shall also be tried.

93884 The `-f` option shall direct *make* to ignore any of these default files and use the specified argument
 93885 as a makefile instead. If the `'-'` argument is specified, standard input shall be used.

93886 The term *makefile* is used to refer to any rules provided by the user, whether in `./makefile` or its
 93887 variants, or specified by the `-f` option.

93888 The rules in makefiles shall consist of the following types of lines: target rules, including special
 93889 targets (see [Target Rules](#), on page 2835), inference rules (see [Inference Rules](#), on page 2838),
 93890 macro definitions (see [Macros](#), on page 2836), empty lines, and comments.

93891 When an escaped `<newline>` (one preceded by a backslash) is found anywhere in the makefile
 93892 except in a command line, an include line, or a line immediately preceding an include line, it
 93893 shall be replaced, along with any leading white space on the following line, with a single
 93894 `<space>`. When an escaped `<newline>` is found in a command line in a makefile, the command
 93895 line shall contain the backslash, the `<newline>`, and the next line, except that the first character
 93896 of the next line shall not be included if it is a `<tab>`. When an escaped `<newline>` is found in an
 93897 include line or in a line immediately preceding an include line, the behavior is unspecified. +

93898	Include Lines	+
93899	If the word include appears at the beginning of a line and is followed by one or more <blank>s,	+
93900	the string formed by the remainder of the line shall be processed as follows to produce a	+
93901	pathname:	+
93902	• The trailing <newline> and any comment shall be discarded. If the resulting string	+
93903	contains any double-quote characters (' " ') the behavior is unspecified.	+
93904	• The resulting string shall be processed for macro expansion (see Macros (on page 2836).	+
93905	• Any <blank>s that appear after the first non-<blank> shall be used as separators to divide	+
93906	the macro-expanded string into fields. It is unspecified whether any other white-space	+
93907	characters are also used as separators. It is unspecified whether pathname expansion (see	+
93908	Section 2.13 , on page 2278) is also performed.	+
93909	• If the processing of separators and optional pathname expansion results in either zero or	+
93910	two or more non-empty fields, the behavior is unspecified. If it results in one non-empty	+
93911	field, that field is taken as the pathname.	+
93912	If the pathname does not begin with a ' / ' it shall be treated as relative to the current working	+
93913	directory of the process, not relative to the directory containing the makefile. If the file does not	+
93914	exist in this location, it is unspecified whether additional directories are searched.	+
93915	The contents of the file specified by the pathname shall be read and processed as if they	+
93916	appeared in the makefile in place of the include line. If the file ends with an escaped <newline>	+
93917	the behavior is unspecified.	+
93918	The file may itself contain further include lines. Implementations shall support nesting of	+
93919	include files up to a depth of at least 16.	+
93920	Makefile Execution	
93921	Makefile command lines shall be processed one at a time by writing the makefile command line	
93922	to the standard output (unless one of the conditions listed under '@' suppresses the writing)	
93923	and executing the command(s) in the line. A <tab> may precede the command to standard	
93924	output. Command execution shall be as if the makefile command line were the argument to the	
93925	<i>system()</i> function. The environment for the command being executed shall contain all of the	
93926	variables in the environment of <i>make</i> .	
93927	By default, when <i>make</i> receives a non-zero status from the execution of a command, it shall	
93928	terminate with an error message to standard error.	
93929	Makefile command lines can have one or more of the following prefixes: a hyphen ('-'), an at	
93930	sign ('@'), or a plus sign ('+'). These shall modify the way in which <i>make</i> processes the	
93931	command. When a command is written to standard output, the prefix shall not be included in	
93932	the output.	
93933	- If the command prefix contains a hyphen, or the -i option is present, or the special target	
93934	.IGNORE has either the current target as a prerequisite or has no prerequisites, any error	
93935	found while executing the command shall be ignored.	
93936	@ If the command prefix contains an at sign and the <i>make</i> utility command line -n option is	
93937	not specified, or the -s option is present, or the special target .SILENT has either the current	
93938	target as a prerequisite or has no prerequisites, the command shall not be written to	
93939	standard output before it is executed.	
93940	+ If the command prefix contains a plus sign, this indicates a makefile command line that	
93941	shall be executed even if -n , -q , or -t is specified.	

93942 **Target Rules**

93943 Target rules are formatted as follows:

```
93944 target [target...]: [prerequisite...][;command]
93945 [command
93946 <tab>command
93947 ...]
```

93948 *line that does not begin with <tab>*

93949 Target entries are specified by a <blank>-separated, non-null list of targets, then a colon, then a
93950 <blank>-separated, possibly empty list of prerequisites. Text following a semicolon, if any, and
93951 all following lines that begin with a <tab>, are makefile command lines to be executed to update
93952 the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a new
93953 entry. An empty or blank line, or a line beginning with '#', may begin a new entry.

93954 Applications shall select target names from the set of characters consisting solely of periods,
93955 underscores, digits, and alphabets from the portable character set (see XBD Section 6.1, on
93956 page 111). Implementations may allow other characters in target names as extensions. The
93957 interpretation of targets containing the characters '%' and '"' is implementation-defined.

93958 A target that has prerequisites, but does not have any commands, can be used to add to the
93959 prerequisite list for that target. Only one target rule for any given target can contain commands.

93960 Lines that begin with one of the following are called *special targets* and control the operation of
93961 *make*:

93962 **.DEFAULT** If the makefile uses this special target, the application shall ensure that it is
93963 specified with commands, but without prerequisites. The commands shall be used
93964 by *make* if there are no other rules available to build a target.

93965 **.IGNORE** Prerequisites of this special target are targets themselves; this shall cause errors
93966 from commands associated with them to be ignored in the same manner as
93967 specified by the `-i` option. Subsequent occurrences of **.IGNORE** shall add to the
93968 list of targets ignoring command errors. If no prerequisites are specified, *make* shall
93969 behave as if the `-i` option had been specified and errors from all commands
93970 associated with all targets shall be ignored.

93971 **.POSIX** The application shall ensure that this special target is specified without
93972 prerequisites or commands. If it appears as the first non-comment line in the
93973 makefile, *make* shall process the makefile as specified by this section; otherwise, the
93974 behavior of *make* is unspecified.

93975 **.PRECIOUS** Prerequisites of this special target shall not be removed if *make* receives one of the
93976 asynchronous events explicitly described in the ASYNCHRONOUS EVENTS
93977 section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious
93978 files. If no prerequisites are specified, all targets in the makefile shall be treated as
93979 if specified with **.PRECIOUS**.

93980 XSI **.SCCS_GET** The application shall ensure that this special target is specified without
93981 prerequisites. If this special target is included in a makefile, the commands
93982 specified with this target shall replace the default commands associated with this
93983 special target (see [Default Rules](#), on page 2840). The commands specified with this
93984 target are used to get all SCCS files that are not found in the current directory.

93985 When source files are named in a dependency list, *make* shall treat them just like
93986 any other target. Because the source file is presumed to be present in the directory,
93987 there is no need to add an entry for it to the makefile. When a target has no
93988 dependencies, but is present in the directory, *make* shall assume that that file is up-
93989 to-date. If, however, an SCCS file named **SCCS/s.source_file** is found for a target

93990 *source_file*, *make* compares the timestamp of the target file with that of the
 93991 **SCCS/s.source_file** to ensure the target is up-to-date. If the target is missing, or if
 93992 the SCCS file is newer, *make* shall automatically issue the commands specified for
 93993 the **.SCCS_GET** special target to retrieve the most recent version. However, if the
 93994 target is writable by anyone, *make* shall not retrieve a new version.

93995 **.SILENT** Prerequisites of this special target are targets themselves; this shall cause
 93996 commands associated with them not to be written to the standard output before
 93997 they are executed. Subsequent occurrences of **.SILENT** shall add to the list of
 93998 targets with silent commands. If no prerequisites are specified, *make* shall behave
 93999 as if the **-s** option had been specified and no commands or touch messages
 94000 associated with any target shall be written to standard output.

94001 **.SUFFIXES** Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are
 94002 used in conjunction with the inference rules (see [Inference Rules](#), on page 2838). If
 94003 **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be
 94004 cleared.

94005 The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified
 94006 without commands.

94007 Targets with names consisting of a leading period followed by the uppercase letters "POSIX"
 94008 and then any other characters are reserved for future standardization. Targets with names
 94009 consisting of a leading period followed by one or more uppercase letters are reserved for
 94010 implementation extensions.

94011 **Macros**

94012 Macro definitions are in the form:

```
94013 string1 = [string2]
```

94014 The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all
 94015 characters, if any, after the equal sign, up to a comment character ('#') or an unescaped
 94016 <newline>. Any <blank>s immediately before or after the equal sign shall be ignored.

94017 Applications shall select macro names from the set of characters consisting solely of periods,
 94018 underscores, digits, and alphabets from the portable character set (see [XBD Section 6.1](#), on
 94019 page 111). A macro name shall not contain an equals sign. Implementations may allow other
 94020 characters in macro names as extensions.

94021 Macros can appear anywhere in the makefile. Macro expansions using the forms $\$(string1)$ or
 94022 $\${string1}$ shall be replaced by *string2*, as follows:

- 94023 • Macros in target lines shall be evaluated when the target line is read.
- 94024 • Macros in makefile command lines shall be evaluated when the command is executed.
- 94025 • Macros in the string before the equals sign in a macro definition shall be evaluated when
 94026 the macro assignment is made.
- 94027 • Macros after the equals sign in a macro definition shall not be evaluated until the defined
 94028 macro is used in a rule or command, or before the equals sign in a macro definition.

94029 The parentheses or braces are optional if *string1* is a single character. The macro $\$\$$ shall be
 94030 replaced by the single character '\$'. If *string1* in a macro expansion contains a macro
 94031 expansion, the results are unspecified.

94032 Macro expansions using the forms $\$(string1[:subst1=[subst2]])$ or $\${string1[:subst1=[subst2]]}$ can
 94033 be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is
 94034 performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word
 94035 in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of

94036 the line, a <blank>, or a <newline>). If *string1* in a macro expansion contains a macro expansion,
94037 the results are unspecified.

94038 Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro
94039 expansions in *string2* of macro definition lines shall be performed when the macro identified by
94040 *string1* is expanded in a rule or command.

94041 Macro definitions shall be taken from the following sources, in the following logical order,
94042 before the makefile(s) are read.

- 94043 1. Macros specified on the *make* utility command line, in the order specified on the
94044 command line. It is unspecified whether the internal macros defined in [Internal Macros](#)
94045 (on page 2839) are accepted from this source.
- 94046 2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the
94047 environment variable. It is unspecified whether the internal macros defined in [Internal](#)
94048 [Macros](#) (on page 2839) are accepted from this source.
- 94049 3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and
94050 including the variables with null values.
- 94051 4. Macros defined in the inference rules built into *make*.

94052 Macro definitions from these sources shall not override macro definitions from a lower-
94053 numbered source. Macro definitions from a single source (for example, the *make* utility
94054 command line, the *MAKEFLAGS* environment variable, or the other environment variables)
94055 shall override previous macro definitions from the same source.

94056 Macros defined in the makefile(s) shall override macro definitions that occur before them in the
94057 makefile(s) and macro definitions from source 4. If the *-e* option is not specified, macros defined
94058 in the makefile(s) shall override macro definitions from source 3. Macros defined in the
94059 makefile(s) shall not override macro definitions from source 1 or source 2.

94060 Before the makefile(s) are read, all of the *make* utility command line options (except *-f* and *-p*)
94061 and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not
94062 already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted
94063 in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance
94064 of the *make* command, the original macro's value is recovered. Other implementation-defined
94065 options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of
94066 the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the
94067 *MAKEFLAGS* environment variable shall be modified to match the new value of the
94068 *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

94069 Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the
94070 *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other
94071 implementation-defined variables may also be added to the environment of *make*.

94072 The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the
94073 pathname of the shell command language interpreter (see *sh*). The *SHELL* environment variable
94074 shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified
94075 on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect
94076 the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the
94077 command line are implementation-defined.

94078

Inference Rules

94079

Inference rules are formatted as follows:

94080

```
target:
```

94081

```
<tab>command
```

94082

```
[<tab>command]
```

94083

```
...
```

94084

```
line that does not begin with <tab> or #
```

94085

The application shall ensure that the *target* portion is a valid target name (see [Target Rules](#), on page 2835) of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any slashes or periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with two periods are double-suffix inference rules. Inference rules can have only one target before the colon.

94091

The application shall ensure that the makefile does not specify prerequisites for inference rules; no characters other than white space shall follow the colon in the first line, except when creating the *empty rule*, described below. Prerequisites are inferred, as described below.

94092

94093

94094

Inference rules can be redefined. A target that matches an existing inference rule shall overwrite the old inference rule. An empty rule can be created with a command consisting of simply a semicolon (that is, the rule still exists and is found during inference rule search, but since it is empty, execution has no effect). The empty rule can also be formatted as follows:

94095

94096

94097

94098

```
rule: ;
```

94099

where zero or more <blank>s separate the colon and semicolon.

94100

The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be made up-to-date. A list of inference rules defines the commands to be executed. By default, *make* contains a built-in set of inference rules. Additional rules can be specified in the makefile.

94101

94102

94103

The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by the inference rules. The order in which the suffixes are specified defines the order in which the inference rules for the suffixes are used. New suffixes shall be appended to the current list by specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is required to change the order of the suffixes.

94104

94105

94106

94107

94108

94109

Normally, the user would provide an inference rule for each suffix. The inference rule to update a target with a suffix *.s1* from a prerequisite with a suffix *.s2* is specified as a target *.s2.s1*. The internal macros provide the means to specify general inference rules (see [Internal Macros](#), on page 2839).

94110

94111

94112

94113

When no target rule is found to update a target, the inference rules shall be checked. The suffix of the target (*.s1*) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special targets. If the *.s1* suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order defined for the first *.s2.s1* rule whose prerequisite file (*\$.s2*) exists. If the target is out-of-date with respect to this prerequisite, the commands for that inference rule shall be executed.

94114

94115

94116

94117

94118

If the target to be built does not contain a suffix and there is no rule for the target, the single suffix inference rules shall be checked. The single-suffix inference rules define how to build a target if a file is found with a name that matches the target name with one of the single suffixes appended. A rule with one suffix *.s2* is the definition of how to build *target* from **target.s2**. The other suffix (*.s1*) is treated as null.

94119

94120

94121

94122

94123

XSI

A tilde ('~') in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the

94124

94125 SCCS files is a prefix, it is incompatible with *make's* suffix point of view. Hence, the '*~*' is a way
94126 of changing any file reference into an SCCS file reference.

94127 Libraries

94128 If a target or prerequisite contains parentheses, it shall be treated as a member of an archive
94129 library. For the *lib(member.o)* expression *lib* refers to the name of the archive library and *member.o*
94130 to the member name. The application shall ensure that the member is an object file with the *.o*
94131 suffix. The modification time of the expression is the modification time for the member as kept
94132 in the archive library; see *ar*. The *.a* suffix shall refer to an archive library. The *.s2.a* rule shall be
94133 used to update a member in the library from a file with a suffix *.s2*.

94134 Internal Macros

94135 The *make* utility shall maintain five internal macros that can be used in target and inference rules.
94136 In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,
94137 *inference rule*, *target*, and *prerequisite* is necessary.

94138 Target rules are specified by the user in a makefile for a particular target. Inference rules are
94139 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites
94140 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those
94141 prerequisites that are generated when inference rules are used. Inference rules are applied to
94142 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in
94143 the makefile. Target rules are applied to targets specified in the makefile.

94144 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)
94145 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon
94146 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be
94147 processed recursively until a target is found that has no prerequisites, at which point the
94148 recursion stops. The recursion shall then back up, updating each target as it goes.

94149 In the definitions that follow, the word *target* refers to one of:

- 94150 • A target specified in the makefile
- 94151 • An explicit prerequisite specified in the makefile that becomes the target when *make*
94152 processes it during recursion
- 94153 • An implicit prerequisite that becomes a target when *make* processes it during recursion

94154 In the definitions that follow, the word *prerequisite* refers to one of the following:

- 94155 • An explicit prerequisite specified in the makefile for a particular target
- 94156 • An implicit prerequisite generated as a result of locating an appropriate inference rule and
94157 corresponding file that matches the suffix of the target

94158 The five internal macros are:

94159 **\$@** The **\$@** shall evaluate to the full target name of the current target, or the archive
94160 filename part of a library archive target. It shall be evaluated for both target and
94161 inference rules.

94162 For example, in the *.c.a* inference rule, **\$@** represents the out-of-date *.a* file to be built.
94163 Similarly, in a makefile target rule to build **lib.a** from **file.c**, **\$@** represents the out-of-
94164 date **lib.a**.

94165 **\$\$** The **\$\$** macro shall be evaluated only when the current target is an archive library
94166 member of the form *libname(member.o)*. In these cases, **\$@** shall evaluate to *libname* and
94167 **\$\$** shall evaluate to *member.o*. The **\$\$** macro shall be evaluated for both target and
94168 inference rules.

- 94169 For example, in a makefile target rule to build **lib.a(file.o)**, **file.o**, as
 94170 opposed to **file.a**, which represents **lib.a**.
- 94171 **\$?** The **\$?** macro shall evaluate to the list of prerequisites that are newer than the current
 94172 target. It shall be evaluated for both target and inference rules.
- 94173 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
 94174 where *prog* is not out-of-date with respect to **file1.o**, but is out-of-date with respect to
 94175 **file2.o** and **file3.o**, **\$?** represents **file2.o** and **file3.o**.
- 94176 **\$<** In an inference rule, the **\$<** macro shall evaluate to the filename whose existence
 94177 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the **\$<**
 94178 macro shall evaluate to the current target name. The meaning of the **\$<** macro shall be
 94179 otherwise unspecified.
- 94180 For example, in the **.c.a** inference rule, **\$<** represents the prerequisite **.c** file.
- 94181 **\$*** The **\$*** macro shall evaluate to the current target name with its suffix deleted. It shall be
 94182 evaluated at least for inference rules.
- 94183 For example, in the **.c.a** inference rule, **\$*.o** represents the out-of-date **.o** file that
 94184 corresponds to the prerequisite **.c** file.

94185 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended
 94186 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part*
 94187 for 'F'. The directory part is the path prefix of the file without a trailing slash; for the current
 94188 directory, the directory part is **.** . When the **\$?** macro contains more than one prerequisite
 94189 filename, the **\$(?D)** and **\$(?F)** (or **\${?D}** and **\${?F}**) macros expand to a list of directory name parts
 94190 and filename parts respectively.

94191 For the target *lib(member.o)* and the **s2.a** rule, the internal macros shall be defined as:

94192 **\$<** *member.s2*
 94193 **\$*** *member*
 94194 **\$@** *lib*
 94195 **\$?** *member.s2*
 94196 **\$%** *member.o*

94197 Default Rules

94198 The default rules for *make* shall achieve results that are the same as if the following were used.
 94199 Implementations that do not support the C-Language Development Utilities option may omit
 94200 **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.
 94201 Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference
 94202 rules. Implementations may provide additional macros and rules.

94203 SPECIAL TARGETS

94204 XSI **.SCCS_GET: sccs \$(SCCSFLAGS) get \$(SCCSGETFLAGS) \$@**

94205 XSI **.SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~**

94206 MACROS

94207 **MAKE=make**
 94208 **AR=ar**
 94209 **ARFLAGS=-rv**
 94210 **YACC=yacc**
 94211 **YFLAGS=**


```

94212         LEX=lex
94213         LFLAGS=
94214         LDFLAGS=
94215         CC=c99
94216         CFLAGS=-O
94217         FC=fort77
94218         FFLAGS=-O 1
94219   XSI     GET=get
94220         GFLAGS=
94221         SCCSFLAGS=
94222         SCCSGETFLAGS=-s

```

94223 *SINGLE SUFFIX RULES*

```

94224         .c:
94225             $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
94226
94227         .f:
94228             $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<
94229
94230         .sh:
94231             cp $< $@
94232             chmod a+x $@
94233
94234   XSI     .c~:
94235             $(GET) $(GFLAGS) -p $< > $*.c
94236             $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
94237
94238         .f~:
94239             $(GET) $(GFLAGS) -p $< > $*.f
94240             $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
94241
94242         .sh~:
94243             $(GET) $(GFLAGS) -p $< > $*.sh
94244             cp $*.sh $@
94245             chmod a+x $@

```

94241 *DOUBLE SUFFIX RULES*

```

94242         .c.o:
94243             $(CC) $(CFLAGS) -c $<
94244
94245         .f.o:
94246             $(FC) $(FFLAGS) -c $<
94247
94248         .y.o:
94249             $(YACC) $(YFLAGS) $<
94250             $(CC) $(CFLAGS) -c y.tab.c
94251             rm -f y.tab.c
94252             mv y.tab.o $@
94253
94254         .l.o:
94255             $(LEX) $(LFLAGS) $<
94256             $(CC) $(CFLAGS) -c lex.yy.c
94257             rm -f lex.yy.c
94258             mv lex.yy.o $@
94259
94260         .y.c:
94261             $(YACC) $(YFLAGS) $<

```

```

94258         mv y.tab.c $@
94259     .l.c:
94260         $(LEX) $(LFLAGS) $<
94261         mv lex.yy.c $@
94262     XSI .c~.o:
94263         $(GET) $(GFLAGS) -p $< > $*.c
94264         $(CC) $(CFLAGS) -c $*.c
94265     .f~.o:
94266         $(GET) $(GFLAGS) -p $< > $*.f
94267         $(FC) $(FFLAGS) -c $*.f
94268     .y~.o:
94269         $(GET) $(GFLAGS) -p $< > $*.y
94270         $(YACC) $(YFLAGS) $*.y
94271         $(CC) $(CFLAGS) -c y.tab.c
94272         rm -f y.tab.c
94273         mv y.tab.o $@
94274     .l~.o:
94275         $(GET) $(GFLAGS) -p $< > $*.l
94276         $(LEX) $(LFLAGS) $*.l
94277         $(CC) $(CFLAGS) -c lex.yy.c
94278         rm -f lex.yy.c
94279         mv lex.yy.o $@
94280     .y~.c:
94281         $(GET) $(GFLAGS) -p $< > $*.y
94282         $(YACC) $(YFLAGS) $*.y
94283         mv y.tab.c $@
94284     .l~.c:
94285         $(GET) $(GFLAGS) -p $< > $*.l
94286         $(LEX) $(LFLAGS) $*.l
94287         mv lex.yy.c $@
94288     .c.a:
94289         $(CC) -c $(CFLAGS) $<
94290         $(AR) $(ARFLAGS) $@ $*.o
94291         rm -f $*.o
94292     .f.a:
94293         $(FC) -c $(FFLAGS) $<
94294         $(AR) $(ARFLAGS) $@ $*.o
94295         rm -f $*.o

```

EXIT STATUS

When the **-q** option is specified, the *make* utility shall exit with one of the following values:

- 0 Successful completion.
- 1 The target was not up-to-date.
- >1 An error occurred.

When the **-q** option is not specified, the *make* utility shall exit with one of the following values:

94302 0 Successful completion.

94303 >0 An error occurred.

94304 CONSEQUENCES OF ERRORS

94305 Default.

94306 APPLICATION USAGE

94307 If there is a source file (such as *./source.c*) and there are two SCCS files corresponding to it
 94308 (*./s.source.c* and *./SCCS/s.source.c*), on XSI-conformant systems *make* uses the SCCS file in the
 94309 current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*,
 94310 *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for
 94311 a given source file, future developers are very likely to be confused.

94312 It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to
 94313 guarantee that they are not affected by local extensions.

94314 The **-k** and **-S** options are both present so that the relationship between the command line, the
 94315 **MAKEFLAGS** variable, and the makefile can be controlled precisely. If the **k** flag is passed in
 94316 **MAKEFLAGS** and a command is of the form:

```
94317 $(MAKE) -S foo
```

94318 then the default behavior is restored for the child *make*.

94319 When the **-n** option is specified, it is always added to **MAKEFLAGS**. This allows a recursive
 94320 *make -n target* to be used to see all of the action that would be taken to update *target*.

94321 Because of widespread historical practice, interpreting a '#' number sign inside a variable as
 94322 the start of a comment has the unfortunate side effect of making it impossible to place a number
 94323 sign in a variable, thus forbidding something like:

```
94324 CFLAGS = "-D COMMENT_CHAR='#'"
```

94325 Many historical *make* utilities stop chaining together inference rules when an intermediate target
 94326 is nonexistent. For example, it might be possible for a *make* to determine that both **.y.c** and **.c.o**
 94327 could be used to convert a **.y** to a **.o**. Instead, in this case, *make* requires the use of a **.y.o** rule.

94328 The best way to provide portable makefiles is to include all of the rules needed in the makefile
 94329 itself. The rules provided use only features provided by other parts of this volume of
 94330 POSIX.1-200x. The default rules include rules for optional commands in this volume of
 94331 POSIX.1-200x. Only rules pertaining to commands that are provided are needed in an
 94332 implementation's default set.

94333 Macros used within other macros are evaluated when the new macro is used rather than when
 94334 the new macro is defined. Therefore:

```
94335 MACRO = value1
94336 NEW   = $(MACRO)
94337 MACRO = value2
```

```
94338 target:
94339     echo $(NEW)
```

94340 would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the
 94341 *echo* command line.

94342 Some historical applications have been known to intermix *target_name* and *macro=name* operands
 94343 on the command line, expecting that all of the macros are processed before any of the targets are
 94344 dealt with. Conforming applications do not do this, although some backwards-compatibility
 94345 support may be included in some implementations.

94346 The following characters in filenames may give trouble: '=', ':', '\', '|', and '@'. In |

94347 include filenames, pattern matching characters and ' ' should also be avoided, as they may be |
 94348 treated as special by some implementations. |

94349 For inference rules, the description of \$< and \$? seem similar. However, an example shows the |
 94350 minor difference. In a makefile containing:

```
94351 foo.o: foo.h
```

94352 if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from
 94353 **foo.c** is used, with \$< equal to **foo.c** and \$? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, \$< is
 94354 equal to **foo.c** and \$? is equal to **foo.h** **foo.c**.

94355 EXAMPLES

- 94356 1. The following command:

```
94357 make
```

94358 makes the first target found in the makefile.

- 94359 2. The following command:

```
94360 make junk
```

94361 makes the target **junk**.

- 94362 3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in
 94363 turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

```
94364 pgm: a.o b.o
94365      c99 a.o b.o -o pgm
94366 a.o: incl.h a.c
94367      c99 -c a.c
94368 b.o: incl.h b.c
94369      c99 -c b.c
```

- 94370 4. An example for making optimized **.o** files from **.c** files is:

```
94371 .c.o:
94372      c99 -c -O $*.c
```

94373 or:

```
94374 .c.o:
94375      c99 -c -O $<
```

- 94376 5. The most common use of the archive interface follows. Here, it is assumed that the source
 94377 files are all C-language source:

```
94378 lib: lib(file1.o) lib(file2.o) lib(file3.o)
94379      @echo lib is now up-to-date
```

94380 The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.

94381 The treatment of escaped <newline>s throughout the makefile is historical practice. For
 94382 example, the inference rule:

```
94383 .c.o\  

  94384 :
```

94385 works, and the macro:

```
94386 f= bar baz\  

  94387    biz
94388 a:
94389    echo ==$f==
```

94390 echoes "=="bar baz biz==".

94391 If \$? were:

94392 /usr/include/stdio.h /usr/include/unistd.h foo.h

94393 then \$(?D) would be:

94394 /usr/include /usr/include .

94395 and \$(?F) would be:

94396 stdio.h unistd.h foo.h

94397 6. The contents of the built-in rules can be viewed by running:

94398 make -p -f /dev/null 2>/dev/null

RATIONALE

94399 The *make* utility described in this volume of POSIX.1-200x is intended to provide the means for
 94400 changing portable source code into executables that can be run on an POSIX.1-200x-conforming
 94401 system. It reflects the most common features present in System V and BSD *makes*.
 94402

94403 Historically, the *make* utility has been an especially fertile ground for vendor and research
 94404 organization-specific syntax modifications and extensions. Examples include:

- 94405 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,
 94406 and others)
- 94407 • Additional “operators” separating targets and their prerequisites (System V, BSD, and
 94408 others)
- 94409 • Specifying that command lines containing the strings “\${MAKE}” and “\$(MAKE)” are
 94410 executed when the `-n` option is specified (GNU and System V)
- 94411 • Modifications of the meaning of internal macros when referencing libraries (BSD and
 94412 others)
- 94413 • Using a single instance of the shell for all of the command lines of the target (BSD and
 94414 others)
- 94415 • Allowing spaces as well as tabs to delimit command lines (BSD)
- 94416 • Adding C preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and
 94417 others)
- 94418 • Remote execution of command lines (Sprite and others)
- 94419 • Specifying additional special targets (BSD, System V, and most others)

94420 Additionally, many vendors and research organizations have rethought the basic concepts of
 94421 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of
 94422 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of
 94423 POSIX.1-200x to require behavior that would be incompatible (and probably inferior) to
 94424 historical practice for such a community.

94425 In similar circumstances, when the industry has enough sufficiently incompatible formats as to
 94426 make them irreconcilable, this volume of POSIX.1-200x has followed one or both of two courses
 94427 of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line options
 94428 have been provided to select the desired behavior (*grep*, *od*, and *pax*).

94429 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes
 94430 accepted by almost all versions of *make*, it was decided that it would be counter-productive to
 94431 change the name. And since the makefile itself is a basic unit of portability, it would not be
 94432 completely effective to reserve a new option letter, such as *make -P*, to achieve the portable

94433 behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to
 94434 specify “standard” behavior. This special target does not preclude extensions in the *make* utility,
 94435 nor does it preclude such extensions being used by the makefile specifying the target; it does,
 94436 however, preclude any extensions from being applied that could alter the behavior of previously
 94437 valid syntax; such extensions must be controlled via command line options or new special
 94438 targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to
 94439 guarantee that they are not affected by local extensions.

94440 The portable version of *make* described in this reference page is not intended to be the state-of-
 94441 the-art software generation tool and, as such, some newer and more leading-edge features have
 94442 not been included. An attempt has been made to describe the portable makefile in a manner that
 94443 does not preclude such extensions as long as they do not disturb the portable behavior described
 94444 here.

94445 When the **-n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive
 94446 *make -n target* to be used to see all of the action that would be taken to update *target*.

94447 The definition of *MAKEFLAGS* allows both the System V letter string and the BSD command
 94448 line formats. The two formats are sufficiently different to allow implementations to support both
 94449 without ambiguity.

94450 Early proposals stated that an “unquoted” number sign was treated as the start of a comment.
 94451 The *make* utility does not pay any attention to quotes. A number sign starts a comment
 94452 regardless of its surroundings.

94453 The text about “other implementation-defined pathnames may also be tried” in addition to
 94454 *./makefile* and *./Makefile* is to allow such extensions as **SCCS/s.Makefile** and other variations.
 94455 It was made an implementation-defined requirement (as opposed to unspecified behavior) to
 94456 highlight surprising implementations that might select something unexpected like
 94457 */etc/Makefile*. XSI-conformant systems also try *./s.makefile*, **SCCS/s.makefile**, *./s.Makefile*,
 94458 and **SCCS/s.Makefile**.

94459 Early proposals contained the macro **NPROC** as a means of specifying that *make* should use *n*
 94460 processes to do the work required. While this feature is a valuable extension for many systems, it
 94461 is not common usage and could require other non-trivial extensions to makefile syntax. This
 94462 extension is not required by this volume of POSIX.1-200x, but could be provided as a compatible
 94463 extension. The macro **PARALLEL** is used by some historical systems with essentially the same
 94464 meaning (but without using a name that is a common system limit value). It is suggested that
 94465 implementors recognize the existing use of **NPROC** and/or **PARALLEL** as extensions to *make*.

94466 The default rules are based on System V. The default **CC=** value is *c99* instead of *cc* because this
 94467 volume of POSIX.1-200x does not standardize the utility named *cc*. Thus, every conforming
 94468 application would be required to define **CC=c99** to expect to run. There is no advantage
 94469 conferred by the hope that the makefile might hit the “preferred” compiler because this cannot
 94470 be guaranteed to work. Also, since the portable makescript can only use the *c99* options, no
 94471 advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue
 94472 as to whether *c99* is as valuable as *cc*.

94473 The **-d** option to *make* is frequently used to produce debugging information, but is too
 94474 implementation-defined to add to this volume of POSIX.1-200x.

94475 The **-p** option is not passed in *MAKEFLAGS* on most historical implementations and to change
 94476 this would cause many implementations to break without sufficiently increased portability.

94477 Commands that begin with a plus sign (‘+’) are executed even if the **-n** option is present. Based
 94478 on the GNU version of *make*, the behavior of **-n** when the plus-sign prefix is encountered has
 94479 been extended to apply to **-q** and **-t** as well. However, the System V convention of forcing
 94480 command execution with **-n** when the command line of a target contains either of the strings
 94481 “\$(MAKE)” or “\${MAKE}” has not been adopted. This functionality appeared in early

94482 proposals, but the danger of this approach was pointed out with the following example of a
94483 portion of a makefile:

```
94484 subdir:
94485     cd subdir; rm all_the_files; $(MAKE)
```

94486 The loss of the System V behavior in this case is well-balanced by the safety afforded to other
94487 makefiles that were not aware of this situation. In any event, the command line plus-sign prefix
94488 can provide the desired functionality.

94489 The double colon in the target rule format is supported in BSD systems to allow more than one
94490 target line containing the same target name to have commands associated with it. Since this is
94491 not functionality described in the SVID or XPG3 it has been allowed as an extension, but not
94492 mandated.

94493 The default rules are provided with text specifying that the built-in rules shall be the same as if
94494 the listed set were used. The intent is that implementations should be able to use the rules
94495 without change, but will be allowed to alter them in ways that do not affect the primary
94496 behavior.

94497 The best way to provide portable makefiles is to include all of the rules needed in the makefile
94498 itself. The rules provided use only features provided by other portions of this volume of
94499 POSIX.1-200x. The default rules include rules for optional commands in this volume of
94500 POSIX.1-200x. Only rules pertaining to commands that are provided are needed in the default
94501 set of an implementation.

94502 One point of discussion was whether to drop the default rules list from this volume of
94503 POSIX.1-200x. They provide convenience, but do not enhance portability of applications. The
94504 prime benefit is in portability of users who wish to type *make command* and have the command
94505 build from a **command.c** file.

94506 The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user
94507 override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the
94508 shell should be chosen by the makefile writer or specified on the *make* command line and not by
94509 a user running *make*.

94510 The *make* utilities in most historical implementations process the prerequisites of a target in left-
94511 to-right order, and the makefile format requires this. It supports the standard idiom used in
94512 many makefiles that produce *yacc* programs; for example:

```
94513 foo: y.tab.o lex.o main.o
94514     $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

94515 In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct
94516 **y.tab.h**. Although there may be better ways to express this relationship, it is widely used
94517 historically. Implementations that desire to update prerequisites in parallel should require an
94518 explicit extension to *make* or the makefile format to accomplish it, as described previously.

94519 The algorithm for determining a new entry for target rules is partially unspecified. Some
94520 historical *makes* allow blank, empty, or comment lines within the collection of commands
94521 marked by leading <tab>s. A conforming makefile must ensure that each command starts with
94522 a <tab>, but implementations are free to ignore blank, empty, and comment lines without
94523 triggering the start of a new entry.

94524 The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with
94525 the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do
94526 so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned
94527 up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it
94528 is required to resend itself the signal it received so that it exits with a status that reflects the
94529 signal. The results from SIGQUIT are partially unspecified because, on systems that create **core**

94530 files upon receipt of SIGQUIT, the **core** from *make* would conflict with a **core** file from the
 94531 command that was running when the SIGQUIT arrived. The main concern was to prevent
 94532 damaged files from appearing up-to-date when *make* is rerun.

94533 The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no
 94534 prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites;
 94535 it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of
 94536 targets than for the entire makefile. These extensions to *make* in System V were made to match
 94537 historical practice from the BSD *make*.

94538 Macros are not exported to the environment of commands to be run. This was never the case in
 94539 any historical *make* and would have serious consequences. The environment is the same as the
 94540 environment to *make* except that **MAKEFLAGS** and macros defined on the *make* command line
 94541 are added.

94542 Some implementations do not use *system()* for all command lines, as required by the portable
 94543 makefile format; as a performance enhancement, they select lines without shell metacharacters
 94544 for direct execution by *execve()*. There is no requirement that *system()* be used specifically, but
 94545 merely that the same results be achieved. The metacharacters typically used to bypass the direct
 94546 *execve()* execution have been any of:

94547 = | ^ () ; & < > * ? [] : \$ \ ' " \ \n

94548 The default in some advanced versions of *make* is to group all the command lines for a target and
 94549 execute them using a single shell invocation; the System V method is to pass each line
 94550 individually to a separate shell. The single-shell method has the advantages in performance and
 94551 the lack of a requirement for many continued lines. However, converting to this newer method
 94552 has caused portability problems with many historical makefiles, so the behavior with the POSIX
 94553 makefile is specified to be the same as that of System V. It is suggested that the special target
 94554 **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a
 94555 target or group of targets.

94556 Novice users of *make* have had difficulty with the historical need to start commands with a
 94557 <tab>. Since it is often difficult to discern differences between <tab>s and <space>s on terminals
 94558 or printed listings, confusing bugs can arise. In early proposals, an attempt was made to correct
 94559 this problem by allowing leading <blank>s instead of <tab>s. However, implementors reported
 94560 many makefiles that failed in subtle ways following this change, and it is difficult to implement
 94561 a *make* that unambiguously can differentiate between macro and command lines. There is
 94562 extensive historical practice of allowing leading spaces before macro definitions. Forcing macro
 94563 lines into column 1 would be a significant backwards-compatibility problem for some makefiles.
 94564 Therefore, historical practice was restored.

94565 There is substantial variation in the handling of include lines by different implementations. |
 94566 However, there is enough commonality for the standard to be able to specify a minimum set of |
 94567 requirements that allow the feature to be used portably. Known variations have been explicitly |
 94568 called out as unspecified behavior in the description.

94569 The System V dynamic dependency feature was not included. It would support:

94570 cat: \$\$@.c

94571 that would expand to;

94572 cat: cat.c

94573 This feature exists only in the new version of System V *make* and, while useful, is not in wide
 94574 usage. This means that macros are expanded twice for prerequisites: once at makefile parse time
 94575 and once at target update time.

94576 Consideration was given to adding metarules to the POSIX *make*. This would make **%.o: %.c** the
 94577 same as **.c.o:**. This is quite useful and available from some vendors, but it would cause too many

94578 changes to this *make* to support. It would have introduced rule chaining and new substitution
 94579 rules. However, the rules for target names have been set to reserve the ‘%’ and ‘”’ characters.
 94580 These are traditionally used to implement metarules and quoting of target names, respectively.
 94581 Implementors are strongly encouraged to use these characters only for these purposes.

94582 A request was made to extend the suffix delimiter character from a period to any character. The
 94583 metarules feature in newer *makes* solves this problem in a more general way. This volume of
 94584 POSIX.1-200x is staying with the more conservative historical definition.

94585 The standard output format for the `-p` option is not described because it is primarily a
 94586 debugging option and because the format is not generally useful to programs. In historical
 94587 implementations the output is not suitable for use in generating makefiles. The `-p` format has
 94588 been variable across historical implementations. Therefore, the definition of `-p` was only to
 94589 provide a consistently named option for obtaining *make* script debugging information.

94590 Some historical implementations have not cleared the suffix list with `-r`.

94591 Implementations should be aware that some historical applications have intermixed *target_name*
 94592 and *macro=value* operands on the command line, expecting that all of the macros are processed
 94593 before any of the targets are dealt with. Conforming applications do not do this, but some
 94594 backwards-compatibility support may be warranted.

94595 Empty inference rules are specified with a semicolon command rather than omitting all
 94596 commands, as described in an early proposal. The latter case has no traditional meaning and is
 94597 reserved for implementation extensions, such as in GNU *make*.

94598 FUTURE DIRECTIONS

94599 None.

94600 SEE ALSO

94601 [Chapter 2](#) (on page 2245), *ar*, *c99*, *get*, *lex*, *sccs*, *sh*, *yacc*

94602 XBD [Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

94603 XSH *exec*, *system()*

94604 CHANGE HISTORY

94605 First released in Issue 2.

94606 Issue 5

94607 The FUTURE DIRECTIONS section is added.

94608 Issue 6

94609 This utility is marked as part of the Software Development Utilities option.

94610 The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the
 94611 SPECIAL TARGETS section.

94612 In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from
 94613 “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of
 94614 *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

94615 It is specified whether the command line is related to the makefile or to the *make* command, and
 94616 the macro processing rules are updated to align with the IEEE P1003.2b draft standard.

94617 The normative text is reworded to avoid use of the term “must” for application requirements.

94618 PASC Interpretation 1003.2 #193 is applied.

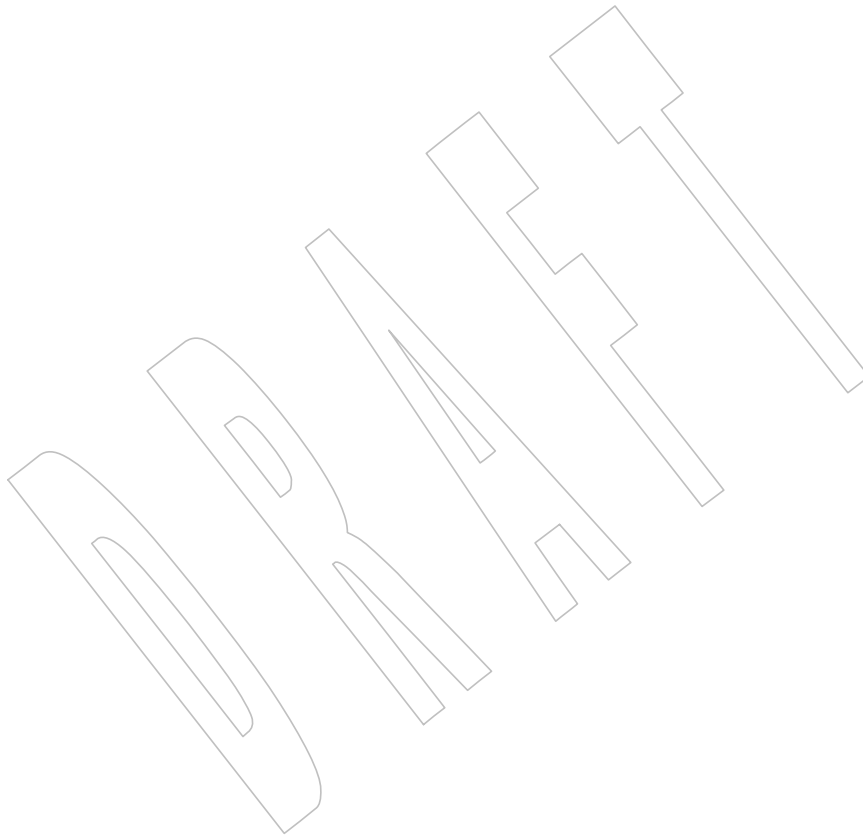
94619
94620
94621
94622
94623**Issue 7**

SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not apply.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

Include lines in makefiles are introduced.

+



94624 **NAME**
 94625 man — display system documentation

94626 **SYNOPSIS**
 94627 man [-k] *name* . . .

94628 **DESCRIPTION**
 94629 The *man* utility shall write information about each of the *name* operands. If *name* is the name of a
 94630 standard utility, *man* at a minimum shall write a message describing the syntax used by the
 94631 standard utility, its options, and operands. If more information is available, the *man* utility shall
 94632 provide it in an implementation-defined manner.

94633 An implementation may provide information for values of *name* other than the standard utilities.
 94634 Standard utilities that are listed as optional and that are not supported by the implementation
 94635 either shall cause a brief message indicating that fact to be displayed or shall cause a full display
 94636 of information as described previously.

94637 **OPTIONS**
 94638 The *man* utility shall conform to XBD [Section 12.2](#) (on page 201).

94639 The following option shall be supported:

94640 **-k** Interpret *name* operands as keywords to be used in searching a utilities summary
 94641 database that contains a brief purpose entry for each standard utility and write lines
 94642 from the summary database that match any of the keywords. The keyword search shall
 94643 produce results that are the equivalent of the output of the following command:

```
94644 grep -Ei '  
94645 name  
94646 name  
94647 ...  
94648 ' summary-database
```

94649 This assumes that the *summary-database* is a text file with a single entry per line; this
 94650 organization is not required and the example using *grep -Ei* is merely illustrative of the
 94651 type of search intended. The purpose entry to be included in the database shall consist
 94652 of a terse description of the purpose of the utility.

94653 **OPERANDS**
 94654 The following operand shall be supported:

94655 *name* A keyword or the name of a standard utility. When **-k** is not specified and *name*
 94656 does not represent one of the standard utilities, the results are unspecified.

94657 **STDIN**
 94658 Not used.

94659 **INPUT FILES**
 94660 None.

94661 **ENVIRONMENT VARIABLES**
 94662 The following environment variables shall affect the execution of *man*:

94663 **LANG** Provide a default value for the internationalization variables that are unset or null.
 94664 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 94665 variables used to determine the values of locale categories.)

94666	<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
94667		
94668	<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and in the summary database). The value of <i>LC_CTYPE</i> need not affect the format of the information written about the <i>name</i> operands.
94669		
94670		
94671		
94672	<i>LC_MESSAGES</i>	
94673		Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
94674		
94675		
94676	XSI <i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
94677	<i>PAGER</i>	Determine an output filtering command for writing the output to a terminal. Any string acceptable as a <i>command_string</i> operand to the <i>sh -c</i> command shall be valid. When standard output is a terminal device, the reference page output shall be piped through the command. If the <i>PAGER</i> variable is null or not set, the command shall be either <i>more</i> or another paginator utility documented in the system documentation.
94678		
94679		
94680		
94681		
94682		
94683	ASYNCHRONOUS EVENTS	
94684		Default.
94685	STDOUT	
94686		The <i>man</i> utility shall write text describing the syntax of the utility <i>name</i> , its options and its operands, or, when <i>-k</i> is specified, lines from the summary database. The format of this text is implementation-defined.
94687		
94688		
94689	STDERR	
94690		The standard error shall be used for diagnostic messages, and may also be used for informational messages of unspecified format.
94691		
94692	OUTPUT FILES	
94693		None.
94694	EXTENDED DESCRIPTION	
94695		None.
94696	EXIT STATUS	
94697		The following exit values shall be returned:
94698		0 Successful completion.
94699		>0 An error occurred.
94700	CONSEQUENCES OF ERRORS	
94701		Default.
94702	APPLICATION USAGE	
94703		None.
94704	EXAMPLES	
94705		None.
94706	RATIONALE	
94707		It is recognized that the <i>man</i> utility is only of minimal usefulness as specified. The opinion of the standard developers was strongly divided as to how much or how little information <i>man</i> should be required to provide. They considered, however, that the provision of some portable way of accessing documentation would aid user portability. The arguments against a fuller specification were:
94708		
94709		
94710		
94711		

- 94712 • Large quantities of documentation should not be required on a system that does not have
- 94713 excess disk space.
- 94714 • The current manual system does not present information in a manner that greatly aids user
- 94715 portability.
- 94716 • A “better help system” is currently an area in which vendors feel that they can add value
- 94717 to their POSIX implementations.

94718 The `-f` option was considered, but due to implementation differences, it was not included in this
 94719 volume of POSIX.1-200x.

94720 The description was changed to be more specific about what has to be displayed for a utility. The
 94721 standard developers considered it insufficient to allow a display of only the synopsis without
 94722 giving a short description of what each option and operand does.

94723 The “purpose” entry to be included in the database can be similar to the section title (less the
 94724 numeric prefix) from this volume of POSIX.1-200x for each utility. These titles are similar to
 94725 those used in historical systems for this purpose.

94726 See *mailx* for rationale concerning the default paginator.

94727 The caveat in the *LC_CTYPE* description was added because it is not a requirement that an
 94728 implementation provide reference pages for all of its supported locales on each system;
 94729 changing *LC_CTYPE* does not necessarily translate the reference page into another language.
 94730 This is equivalent to the current state of *LC_MESSAGES* in POSIX.1-200x—locale-specific
 94731 messages are not yet a requirement.

94732 The historical *MANPATH* variable is not included in POSIX because no attempt is made to
 94733 specify naming conventions for reference page files, nor even to mandate that they are files at
 94734 all. On some implementations they could be a true database, a hypertext file, or even fixed
 94735 strings within the *man* executable. The standard developers considered the portability of
 94736 reference pages to be outside their scope of work. However, users should be aware that
 94737 *MANPATH* is implemented on a number of historical systems and that it can be used to tailor
 94738 the search pattern for reference pages from the various categories (utilities, functions, file
 94739 formats, and so on) when the system administrator reveals the location and conventions for
 94740 reference pages on the system.

94741 The keyword search can rely on at least the text of the section titles from these utility
 94742 descriptions, and the implementation may add more keywords. The term “section titles” refers
 94743 to the strings such as:

```
94744 man - Display system documentation
94745 ps - Report process status
```

94746 FUTURE DIRECTIONS

94747 None.

94748 SEE ALSO

94749 *more*

94750 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

94751 CHANGE HISTORY

94752 First released in Issue 4.

94753 Issue 5

94754 The FUTURE DIRECTIONS section is added.

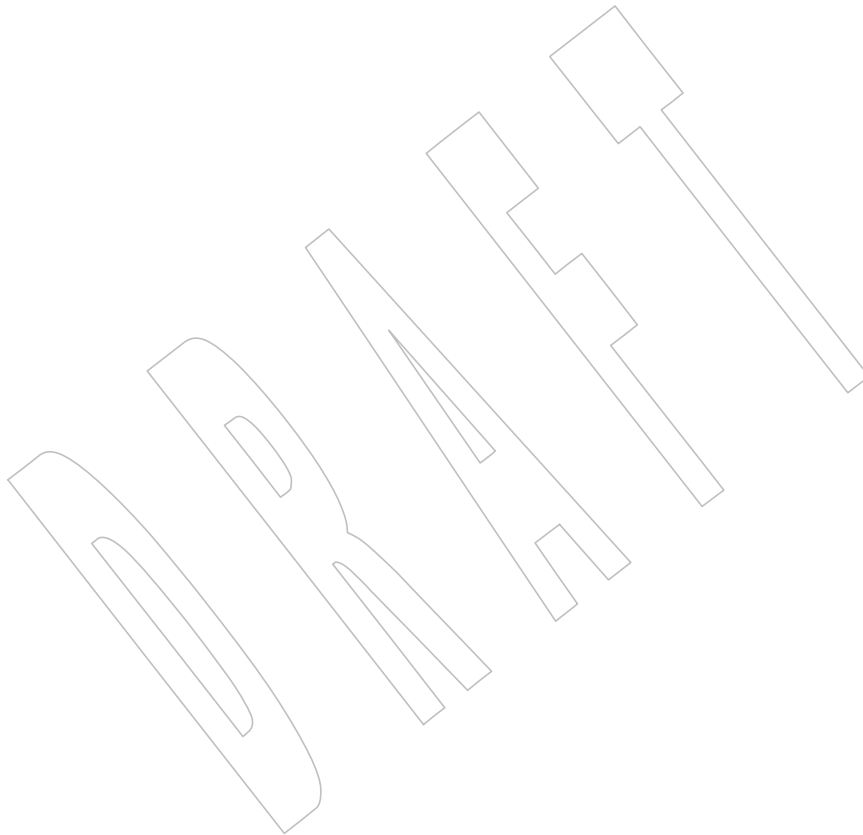
94755

Issue 7

94756

94757

Austin Group Interpretation 1003.1-2001 #108 is applied, clarifying that informational messages may appear on standard error.



94758

NAME

94759

mesg — permit or deny messages

94760

SYNOPSIS

94761

mesg [y|n]

94762

DESCRIPTION

94763

94764

94765

94766

94767

94768

The *mesg* utility shall control whether other users are allowed to send messages via *write*, *talk*, or other utilities to a terminal device. The terminal device affected shall be determined by searching for the first terminal in the sequence of devices associated with standard input, standard output, and standard error, respectively. With no arguments, *mesg* shall report the current state without changing it. Processes with appropriate privileges may be able to send messages to the terminal independent of the current state.

94769

OPTIONS

94770

None.

94771

OPERANDS

94772

The following operands shall be supported in the POSIX locale:

94773

y

Grant permission to other users to send messages to the terminal device.

94774

n

Deny permission to other users to send messages to the terminal device.

94775

STDIN

94776

Not used.

94777

INPUT FILES

94778

None.

94779

ENVIRONMENT VARIABLES

94780

The following environment variables shall affect the execution of *mesg*:

94781

LANG

Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)

94783

94784

LC_ALL

If set to a non-empty string value, override the values of all the other internationalization variables.

94785

94786

LC_CTYPE

Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

94787

94788

94789

LC_MESSAGES

Determine the locale that should be used to affect the format and contents of diagnostic messages written (by *mesg*) to standard error.

94790

94791

94792

XSI

*NLSPATH*Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94793

ASYNCHRONOUS EVENTS

94794

Default.

94795

STDOUT

94796

If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

94797 **STDERR**
 94798 The standard error shall be used only for diagnostic messages.

94799 **OUTPUT FILES**
 94800 None.

94801 **EXTENDED DESCRIPTION**
 94802 None.

94803 **EXIT STATUS**
 94804 The following exit values shall be returned:

- 94805 0 Receiving messages is allowed.
- 94806 1 Receiving messages is not allowed.
- 94807 >1 An error occurred.

94808 **CONSEQUENCES OF ERRORS**
 94809 Default.

94810 **APPLICATION USAGE**
 94811 The mechanism by which the message status of the terminal is changed is unspecified.
 94812 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has
 94813 successfully completed. These actions may include, but are not limited to: another invocation of
 94814 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or
 94815 *chmod()* function, and so on.

94816 **EXAMPLES**
 94817 None.

94818 **RATIONALE**
 94819 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather
 94820 than the controlling terminal for the session. This is because users logged in more than once
 94821 should be able to change any of their login terminals without having to stop the job running in
 94822 those sessions. This is not a security problem involving the terminals of other users because
 94823 appropriate privileges would be required to affect the terminal of another user.

94824 The method of checking each of the first three file descriptors in sequence until a terminal is
 94825 found was adopted from System V.

94826 The file */dev/tty* is not specified for the terminal device because it was thought to be too
 94827 restrictive. Typical environment changes for the *n* operand are that write permissions are
 94828 removed for *others* and *group* from the appropriate device. It was decided to leave the actual
 94829 description of what is done as unspecified because of potential differences between
 94830 implementations.

94831 The format for standard output is unspecified because of differences between historical
 94832 implementations. This output is generally not useful to shell scripts (they can use the exit status),
 94833 so exact parsing of the output is unnecessary.

94834 **FUTURE DIRECTIONS**
 94835 None.

94836 **SEE ALSO**
 94837 *talk*, *write*

94838 XBD Chapter 8 (on page 159)

+

94839

CHANGE HISTORY

94840

First released in Issue 2.

94841

Issue 6

94842

This utility is marked as part of the User Portability Utilities option.

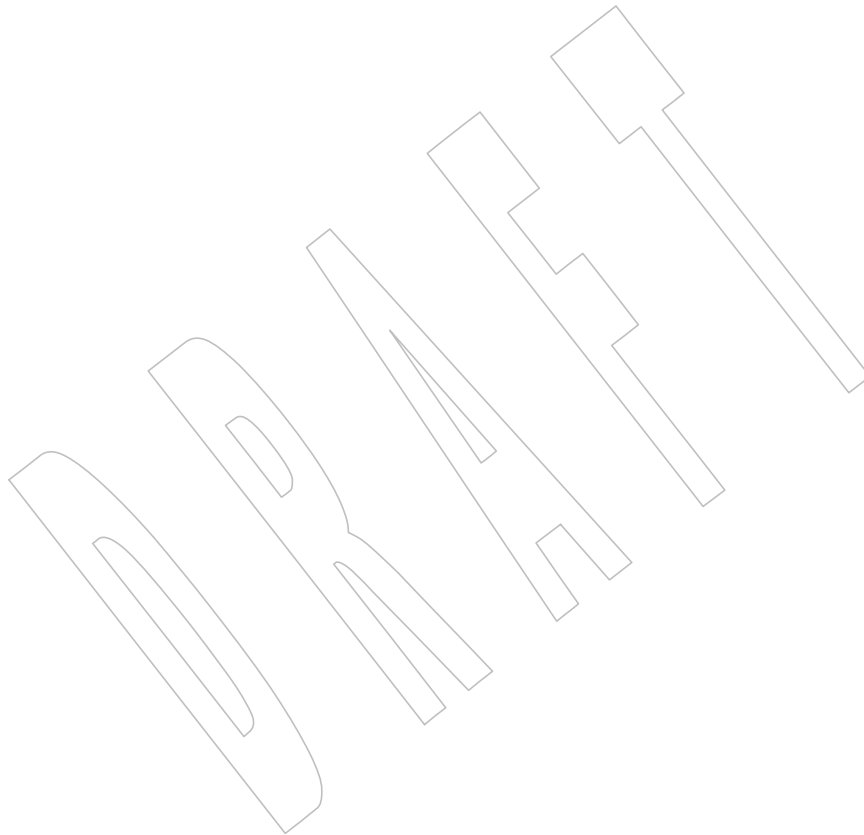
94843

Issue 7

94844

The *mesg* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

94845



94846 **NAME**

94847 mkdir — make directories

94848 **SYNOPSIS**94849 mkdir [-p] [-m *mode*] *dir*...94850 **DESCRIPTION**94851 The *mkdir* utility shall create the directories specified by the operands, in the order specified.94852 For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir()* function
94853 defined in the System Interfaces volume of POSIX.1-200x, called with the following arguments:

- 94854 1. The
- dir*
- operand is used as the
- path*
- argument.
-
- 94855 2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as
-
- 94856 the
- mode*
- argument. (If the
- m**
- option is specified, the value of the
- mkdir()*
- mode*
- argument
-
- 94857 is unspecified, but the directory shall at no time have permissions less restrictive than the
-
- 94858
- m mode**
- option-argument.)

94859 **OPTIONS**94860 The *mkdir* utility shall conform to XBD [Section 12.2](#) (on page 201).

94861 The following options shall be supported:

94862 **-m mode** Set the file permission bits of the newly-created directory to the specified *mode*
94863 value. The *mode* option-argument shall be the same as the *mode* operand defined
94864 for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-'
94865 shall be interpreted relative to an assumed initial mode of *a=rwx*; '+' shall add
94866 permissions to the default mode, '-' shall delete permissions from the default
94867 mode.94868 **-p** Create any missing intermediate pathname components.94869 For each *dir* operand that does not name an existing directory, effects equivalent to
94870 those caused by the following command shall occur:94871

```
mkdir -p -m $(umask -S),u+wx $(dirname dir) &&  
mkdir [-m mode] dir
```

94873 where the **-m mode** option represents that option supplied to the original
94874 invocation of *mkdir*, if any.94875 Each *dir* operand that names an existing directory shall be ignored without error.94876 **OPERANDS**

94877 The following operand shall be supported:

94878 *dir* A pathname of a directory to be created.94879 **STDIN**

94880 Not used.

94881 **INPUT FILES**

94882 None.

94883 **ENVIRONMENT VARIABLES**94884 The following environment variables shall affect the execution of *mkdir*:94885 **LANG** Provide a default value for the internationalization variables that are unset or null. |
94886 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
94887 variables used to determine the values of locale categories.)

- 94888 *LC_ALL* If set to a non-empty string value, override the values of all the other
94889 internationalization variables.
- 94890 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
94891 characters (for example, single-byte as opposed to multi-byte characters in
94892 arguments).
- 94893 *LC_MESSAGES*
94894 Determine the locale that should be used to affect the format and contents of
94895 diagnostic messages written to standard error.
- 94896 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 94897 **ASYNCHRONOUS EVENTS**
- 94898 Default.
- 94899 **STDOUT**
- 94900 Not used.
- 94901 **STDERR**
- 94902 The standard error shall be used only for diagnostic messages.
- 94903 **OUTPUT FILES**
- 94904 None.
- 94905 **EXTENDED DESCRIPTION**
- 94906 None.
- 94907 **EXIT STATUS**
- 94908 The following exit values shall be returned:
- 94909 0 All the specified directories were created successfully or the **-p** option was specified and all
94910 the specified directories now exist.
- 94911 >0 An error occurred.
- 94912 **CONSEQUENCES OF ERRORS**
- 94913 Default.
- 94914 **APPLICATION USAGE**
- 94915 The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions
94916 removed in accordance with the file mode creation mask. For intermediate pathname
94917 components created by *mkdir*, the mode is the default modified by *u+wx* so that the
94918 subdirectories can always be created regardless of the file mode creation mask; if different
94919 ultimate permissions are desired for the intermediate directories, they can be changed
94920 afterwards with *chmod*.
- 94921 Note that some of the requested directories may have been created even if an error occurs.
- 94922 **EXAMPLES**
- 94923 None.
- 94924 **RATIONALE**
- 94925 The System V **-m** option was included to control the file mode.
- 94926 The System V **-p** option was included to create any needed intermediate directories and to
94927 complement the functionality provided by *rmdir* for removing directories in the path prefix as
94928 they become empty. Because no error is produced if any path component already exists, the **-p**
94929 option is also useful to ensure that a particular directory exists.
- 94930 The functionality of *mkdir* is described substantially through a reference to the *mkdir()* function
94931 in the System Interfaces volume of POSIX.1-200x. For example, by default, the mode of the
94932 directory is affected by the file mode creation mask in accordance with the specified behavior of

94933 the *mkdir()* function. In this way, there is less duplication of effort required for describing details
94934 of the directory creation.

FUTURE DIRECTIONS

94935 None.
94936

SEE ALSO

94937 *chmod*, *rm*, *rmdir*, *umask*
94938

94939 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) |

94940 XSH *mkdir*

CHANGE HISTORY

94941 First released in Issue 2.
94942

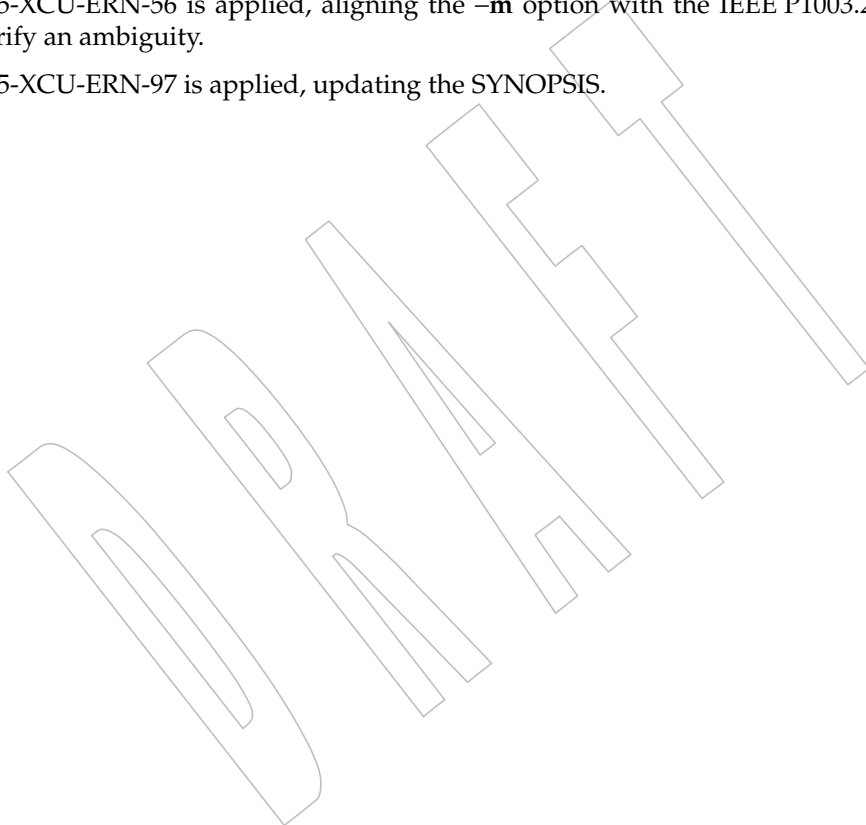
Issue 5

94943 The FUTURE DIRECTIONS section is added.
94944

Issue 7

94945 SD5-XCU-ERN-56 is applied, aligning the **-m** option with the IEEE P1003.2b draft standard to
94946 clarify an ambiguity.
94947

94948 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



94949 **NAME**

94950 mkfifo — make FIFO special files

94951 **SYNOPSIS**94952 `mkfifo [-m mode] file...`94953 **DESCRIPTION**94954 The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order
94955 specified.94956 For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo()* function
94957 defined in the System Interfaces volume of POSIX.1-200x, called with the following arguments:

- 94958
1. The *file* operand is used as the *path* argument.
 - 94959 2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP,
94960 S_IROTH, and S_IWOTH is used as the *mode* argument. (If the `-m` option is specified, the
94961 value of the *mkfifo()* *mode* argument is unspecified, but the FIFO shall at no time have
94962 permissions less restrictive than the `-m mode` option-argument.)

94963 **OPTIONS**94964 The *mkfifo* utility shall conform to XBD [Section 12.2](#) (on page 201).

94965 The following option shall be supported:

94966 `-m mode` Set the file permission bits of the newly-created FIFO to the specified *mode* value.
94967 The *mode* option-argument shall be the same as the *mode* operand defined for the
94968 *chmod* utility. In the *symbolic_mode* strings, the *op* characters '+' and '-' shall be
94969 interpreted relative to an assumed initial mode of *a=rw*.94970 **OPERANDS**

94971 The following operand shall be supported:

94972 *file* A pathname of the FIFO special file to be created.94973 **STDIN**

94974 Not used.

94975 **INPUT FILES**

94976 None.

94977 **ENVIRONMENT VARIABLES**94978 The following environment variables shall affect the execution of *mkfifo*:94979 *LANG* Provide a default value for the internationalization variables that are unset or null.
94980 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
94981 variables used to determine the values of locale categories.)94982 *LC_ALL* If set to a non-empty string value, override the values of all the other
94983 internationalization variables.94984 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
94985 characters (for example, single-byte as opposed to multi-byte characters in
94986 arguments).94987 *LC_MESSAGES*94988 Determine the locale that should be used to affect the format and contents of
94989 diagnostic messages written to standard error.

94990 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

94991 **ASYNCHRONOUS EVENTS**

94992 Default.

94993 **STDOUT**

94994 Not used.

94995 **STDERR**

94996 The standard error shall be used only for diagnostic messages.

94997 **OUTPUT FILES**

94998 None.

94999 **EXTENDED DESCRIPTION**

95000 None.

95001 **EXIT STATUS**

95002 The following exit values shall be returned:

95003 0 All the specified FIFO special files were created successfully.

95004 >0 An error occurred.

95005 **CONSEQUENCES OF ERRORS**

95006 Default.

95007 **APPLICATION USAGE**

95008 None.

95009 **EXAMPLES**

95010 None.

95011 **RATIONALE**

95012 This utility was added to permit shell applications to create FIFO special files.

95013 The **-m** option was added to control the file mode, for consistency with the similar functionality
95014 provided by the *mkdir* utility.

95015 Early proposals included a **-p** option similar to the *mkdir -p* option that created intermediate
95016 directories leading up to the FIFO specified by the final component. This was removed because
95017 it is not commonly needed and is not common practice with similar utilities.

95018 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo()* function
95019 in the System Interfaces volume of POSIX.1-200x. For example, by default, the mode of the FIFO
95020 file is affected by the file mode creation mask in accordance with the specified behavior of the
95021 *mkfifo()* function. In this way, there is less duplication of effort required for describing details of
95022 the file creation.

95023 **FUTURE DIRECTIONS**

95024 None.

95025 **SEE ALSO**

95026 *chmod*, *umask*

95027 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

95028 XSH *mkfifo*

95029 **CHANGE HISTORY**

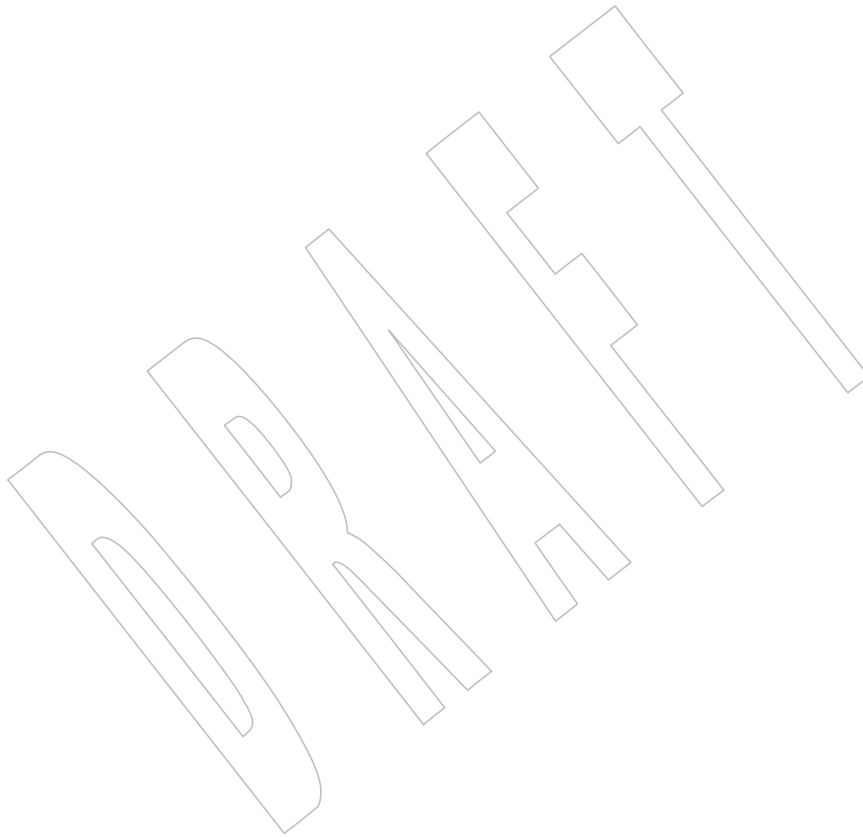
95030 First released in Issue 3.

95031

95032

Issue 6

The `-m` option is aligned with the IEEE P1003.2b draft standard to clarify an ambiguity.



95033

NAME

95034

more — display files on a page-by-page basis

95035

SYNOPSIS

95036

UP `more [-ceisu] [-n number] [-p command] [-t tagstring] [file...]`

95037

DESCRIPTION

95038

95039

95040

95041

95042

95043

The *more* utility shall read files and either write them to the terminal on a page-by-page basis or filter them to standard output. If standard output is not a terminal device, all input files shall be copied to standard output in their entirety, without modification, except as specified for the `-s` option. If standard output is a terminal device, the files shall be written a number of lines (one screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION section.

95044

95045

95046

95047

Certain block-mode terminals do not have all the capabilities necessary to support the complete *more* definition; they are incapable of accepting commands that are not terminated with a <newline>. Implementations that support such terminals shall provide an operating mode to *more* in which all commands can be terminated with a <newline> on those terminals. This mode:

95048

- Shall be documented in the system documentation

95049

95050

95051

- Shall, at invocation, inform the user of the terminal deficiency that requires the <newline> usage and provide instructions on how this warning can be suppressed in future invocations

95052

- Shall not be required for implementations supporting only fully capable terminals

95053

- Shall not affect commands already requiring <newline>s

95054

95055

- Shall not affect users on the capable terminals from using *more* as described in this volume of POSIX.1-200x

95056

OPTIONS

95057

95058

The *more* utility shall conform to XBD [Section 12.2](#) (on page 201), except that '+' may be recognized as an option delimiter as well as '-'.

95059

The following options shall be supported:

95060

95061

95062

95063

95064

-c If a screen is to be written that has no lines in common with the current screen, or *more* is writing its first screen, *more* shall not scroll the screen, but instead shall redraw each line of the screen in turn, from the top of the screen to the bottom. In addition, if *more* is writing its first screen, the screen shall be cleared. This option may be silently ignored on devices with insufficient terminal capabilities.

95065

95066

-e By default, *more* shall exit immediately after writing the last line of the last file in the argument list. If the `-e` option is specified:

95067

95068

95069

1. If there is only a single file in the argument list and that file was completely displayed on a single screen, *more* shall exit immediately after writing the last line of that file.

95070

95071

95072

2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the argument list twice without an intervening operation. See the EXTENDED DESCRIPTION section.

95073

95074

-i Perform pattern matching in searches without regard to case; see XBD [Section 9.2](#) (on page 168).

- 95075 **-n number** Specify the number of lines per screenful. The *number* argument is a positive
95076 decimal integer. The **-n** option shall override any values obtained from any other
95077 source.
- 95078 **-p command** Each time a screen from a new file is displayed or redisplayed (including as a
95079 result of *more* commands; for example, **:p**), execute the *more* command(s) in the
95080 command arguments in the order specified, as if entered by the user after the first
95081 screen has been displayed. No intermediate results shall be displayed (that is, if the
95082 command is a movement to a screen different from the normal first screen, only the
95083 screen resulting from the command shall be displayed.) If any of the commands
95084 fail for any reason, an informational message to this effect shall be written, and no
95085 further commands specified using the **-p** option shall be executed for this file.
- 95086 **-s** Behave as if consecutive empty lines were a single empty line.
- 95087 **-t tagstring** Write the screenful of the file containing the tag named by the *tagstring* argument.
95088 See the *ctags* utility. The tags feature represented by **-t tagstring** and the **:t**
95089 command is optional. It shall be provided on any system that also provides a
95090 conforming implementation of *ctags*; otherwise, the use of **-t** produces undefined
95091 results.
- 95092 The filename resulting from the **-t** option shall be logically added as a prefix to the
95093 list of command line files, as if specified by the user. If the tag named by the
95094 *tagstring* argument is not found, it shall be an error, and *more* shall take no further
95095 action.
- 95096 If the tag specifies a line number, the first line of the display shall contain the
95097 beginning of that line. If the tag specifies a pattern, the first line of the display shall
95098 contain the beginning of the matching text from the first line of the file that
95099 contains that pattern. If the line does not exist in the file or matching text is not
95100 found, an informational message to this effect shall be displayed, and *more* shall
95101 display the default screen as if **-t** had not been specified.
- 95102 If both the **-t tagstring** and **-p command** options are given, the **-t tagstring** shall be
95103 processed first; that is, the file and starting line for the display shall be as specified
95104 by **-t**, and then the **-p more** command shall be executed. If the line (matching text)
95105 specified by the **-t** command does not exist (is not found), no **-p more** command
95106 shall be executed for this file at any time.
- 95107 **-u** Treat a <backspace> as a printable control character, displayed as an
95108 implementation-defined character sequence (see the EXTENDED DESCRIPTION
95109 section), suppressing backspacing and the special handling that produces
95110 underlined or standout mode text on some terminal types. Also, do not ignore a
95111 <carriage-return> at the end of a line.

OPERANDS

The following operand shall be supported:

- 95114 *file* A pathname of an input file. If no *file* operands are specified, the standard input
95115 shall be used. If a *file* is '-', the standard input shall be read at that point in the
95116 sequence.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.

INPUT FILES

The input files being examined shall be text files. If standard output is a terminal, standard error shall be used to read commands from the user. If standard output is a terminal, standard error is not readable, and command input is needed, *more* may attempt to obtain user commands from the controlling terminal (for example, */dev/tty*); otherwise, *more* shall terminate with an error

more*Utilities*

95124 indicating that it was unable to read user commands. If standard output is not a terminal, no
 95125 error shall result if standard error cannot be opened for reading.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *more*:

95126 *COLUMNS* Override the system-selected horizontal display line size. See XBD [Chapter 8](#) (on
 95129 page 159) for valid values and results when it is unset or null.

95130 *EDITOR* Used by the *v* command to select an editor. See the EXTENDED DESCRIPTION
 95131 section.

95132 *LANG* Provide a default value for the internationalization variables that are unset or null.
 95133 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 95134 variables used to determine the values of locale categories.)

95135 *LC_ALL* If set to a non-empty string value, override the values of all the other
 95136 internationalization variables.

95137 *LC_COLLATE*
 95138 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 95139 character collating elements within regular expressions.

95140 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 95141 characters (for example, single-byte as opposed to multi-byte characters in
 95142 arguments and input files) and the behavior of character classes within regular
 95143 expressions.

95144 *LC_MESSAGES*
 95145 Determine the locale that should be used to affect the format and contents of
 95146 diagnostic messages written to standard error and informative messages written to
 95147 standard output.

95148 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

95149 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
 95150 screenful. See XBD [Chapter 8](#) (on page 159) for valid values and results when it is
 95151 unset or null. The *-n* option shall take precedence over the *LINES* variable for
 95152 determining the number of lines in a screenful.

95153 *MORE* Determine a string containing options described in the OPTIONS section preceded
 95154 with hyphens and <blank>-separated as on the command line. Any command line
 95155 options shall be processed after those in the *MORE* variable, as if the command
 95156 line were:

95157 `more $MORE options operands`

95158 The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for
 95159 determining the number of lines in a screenful.

95160 *TERM* Determine the name of the terminal type. If this variable is unset or null, an
 95161 unspecified default terminal type is used.

ASYNCHRONOUS EVENTS

95162 Default.

STDOUT

95164 The standard output shall be used to write the contents of the input files.
 95165

95166 **STDERR**

95167 The standard error shall be used for diagnostic messages and user commands (see the INPUT
 95168 FILES section), and, if standard output is a terminal device, to write a prompting string. The
 95169 prompting string shall appear on the screen line below the last line of the file displayed in the
 95170 current screenful. The prompt shall contain the name of the file currently being examined and
 95171 shall contain an end-of-file indication and the name of the next file, if any, when prompting at
 95172 the end-of-file. If an error or informational message is displayed, it is unspecified whether it is
 95173 contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the
 95174 user shall be prompted for a continuation character, at which point another message or the user
 95175 prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether
 95176 informational messages are written for other user commands.

95177 **OUTPUT FILES**

95178 None.

95179 **EXTENDED DESCRIPTION**

95180 The following section describes the behavior of *more* when the standard output is a terminal
 95181 device. If the standard output is not a terminal device, no options other than `-s` shall have any
 95182 effect, and all input files shall be copied to standard output otherwise unmodified, at which time
 95183 *more* shall exit without further action.

95184 The number of lines available per screen shall be determined by the `-n` option, if present, or by
 95185 examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither
 95186 method yields a number, an unspecified number of lines shall be used.

95187 The maximum number of lines written shall be one less than this number, because the screen
 95188 line after the last line written shall be used to write a user prompt and user input. If the number
 95189 of lines in the screen is less than two, the results are undefined. It is unspecified whether user
 95190 input is permitted to be longer than the remainder of the single line where the prompt has been
 95191 written.

95192 The number of columns available per line shall be determined by examining values in the
 95193 environment (see the ENVIRONMENT VARIABLES section), with a default value as described
 95194 in XBD [Chapter 8](#) (on page 159).

95195 Lines that are longer than the display shall be folded; the length at which folding occurs is
 95196 unspecified, but should be appropriate for the output device. Folding may occur between glyphs
 95197 of single characters that take up multiple display columns.

95198 When standard output is a terminal and `-u` is not specified, *more* shall treat `<backspace>`s and
 95199 `<carriage-return>`s specially:

- 95200 • A character, followed first by a sequence of n `<backspace>`s (where n is the same as the
 95201 number of column positions that the character occupies), then by n underscore characters
 95202 (`'_'`), shall cause that character to be written as underlined text, if the terminal type
 95203 supports that. The n underscore characters, followed first by n `<backspace>`s, then any
 95204 character with n column positions, shall also cause that character to be written as
 95205 underlined text, if the terminal type supports that.
- 95206 • A sequence of n `<backspace>`s (where n is the same as the number of column positions that
 95207 the previous character occupies) that appears between two identical printable characters
 95208 shall cause the first of those two characters to be written as emboldened text (that is,
 95209 visually brighter, standout mode, or inverse-video mode), if the terminal type supports
 95210 that, and the second to be discarded. Immediately subsequent occurrences of
 95211 `<backspace>/character` pairs for that same character shall also be discarded. (For example,
 95212 the sequence `"a\b a\b a\b a"` is interpreted as a single emboldened `'a'`.)

- 95213 • The *more* utility shall logically discard all other <backspace>s from the line as well as the
- 95214 character which precedes them, if any.
- 95215 • A <carriage-return> at the end of a line shall be ignored, rather than being written as a
- 95216 non-printable character, as described in the next paragraph.

95217 It is implementation-defined how other non-printable characters are written. Implementations
 95218 should use the same format that they use for the *ex print* command; see the OPTIONS section
 95219 within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it
 95220 crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the
 95221 number of columns on the display is less than the number of columns any single character in the
 95222 line being displayed would occupy.

95223 When each new file is displayed (or redisplayed), *more* shall write the first screen of the file.
 95224 Once the initial screen has been written, *more* shall prompt for a user command. If the execution
 95225 of the user command results in a screen that has lines in common with the current screen, and
 95226 the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is
 95227 unspecified whether the screen is scrolled or redrawn.

95228 For all files but the last (including standard input if no file was specified, and for the last file as
 95229 well, if the *-e* option was not specified), when *more* has written the last line in the file, *more* shall
 95230 prompt for a user command. This prompt shall contain the name of the next file as well as an
 95231 indication that *more* has reached end-of-file. If the user command is *f*, <control>-F, <space>, *j*,
 95232 <newline>, *d*, <control>-D, or *s*, *more* shall display the next file. Otherwise, if displaying the last
 95233 file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

95234 Several of the commands described in this section display a previous screen from the input
 95235 stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is
 95236 implementation-defined how much backwards motion is supported. If a command cannot be
 95237 executed because of a limitation on backwards motion, an error message to this effect shall be
 95238 displayed, the current screen shall not change, and the user shall be prompted for another
 95239 command.

95240 If a command cannot be performed because there are insufficient lines to display, *more* shall alert
 95241 the terminal. If a command cannot be performed because there are insufficient lines to display or
 95242 a */* command fails: if the input is the standard input, the last screen in the file may be displayed;
 95243 otherwise, the current file and screen shall not change, and the user shall be prompted for
 95244 another command.

95245 The interactive commands in the following sections shall be supported. Some commands can be
 95246 preceded by a decimal integer, called *count* in the following descriptions. If not specified with
 95247 the command, *count* shall default to 1. In the following descriptions, *pattern* is a basic regular
 95248 expression, as described in XBD Section 9.3 (on page 169). The term “examine” is historical
 95249 usage meaning “open the file for viewing”; for example, *more foo* would be expressed as
 95250 examining file *foo*.

95251 In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a
 95252 line from the file being examined.

95253 In the following descriptions, the *current position* refers to two things:

- 95254 1. The position of the current line on the screen
- 95255 2. The line number (in the file) of the current line on the screen

95256 Usually, the line on the screen corresponding to the current position is the third line on the
 95257 screen. If this is not possible (there are fewer than three lines to display or this is the first page of
 95258 the file, or it is the last page of the file), then the current position is either the first or last line on
 95259 the screen as described later.

95260

Help

95261

Synopsis: h

95262

Write a summary of these commands and other implementation-defined commands. The behavior shall be as if the *more* utility were executed with the *-e* option on a file that contained the summary information. The user shall be prompted as described earlier in this section when end-of-file is reached. If the user command is one of those specified to continue to the next file, *more* shall return to the file and screen state from which the *h* command was executed.

95263

95264

95265

95266

95267

Scroll Forward One Screenful

95268

Synopsis: [*count*]f

95269

[*count*]<control>-F

95270

Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size, only the final screenful shall be written.

95271

95272

Scroll Backward One Screenful

95273

Synopsis: [*count*]b

95274

[*count*]<control>-B

95275

Scroll backward *count* lines, with a default of one screenful (see the *-n* option). If *count* is more than the screen size, only the final screenful shall be written.

95276

95277

Scroll Forward One Line

95278

Synopsis: [*count*]<space>

95279

[*count*]j

95280

[*count*]<newline>

95281

Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for *j* and <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen size.

95282

95283

95284

Scroll Backward One Line

95285

Synopsis: [*count*]k

95286

Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the screen size.

95287

95288

Scroll Forward One Half Screenful

95289

Synopsis: [*count*]d

95290

[*count*]<control>-D

95291

Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent *d*, <control>-D, and *u* commands.

95292

95293

Skip Forward One Line

95294

Synopsis: [*count*]s

95295

Display the screenful beginning with the line *count* lines after the last line on the current screen. If *count* would cause the current position to be such that less than one screenful would be written, the last screenful in the file shall be written.

95296

95297

95298 **Scroll Backward One Half Screenful**

95299 *Synopsis:* [count]u
 95300 [count]<control>-U

95301 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it
 95302 shall become the new default for subsequent **d**, <control>-D, **u**, and <control>-U commands.
 95303 The entire *count* lines shall be written, even if *count* is more than the screen size.

95304 **Go to Beginning of File**

95305 *Synopsis:* [count]g

95306 Display the screenful beginning with line *count*.

95307 **Go to End-of-File**

95308 *Synopsis:* [count]G

95309 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the
 95310 last screenful of the file.

95311 **Refresh the Screen**

95312 *Synopsis:* r
 95313 <control>-L

95314 Refresh the screen.

95315 **Discard and Refresh**

95316 *Synopsis:* R

95317 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered
 95318 input shall not be discarded and the **R** command shall be equivalent to the **r** command.

95319 **Mark Position**

95320 *Synopsis:* m*letter*

95321 Mark the current position with the letter named by *letter*, where *letter* represents the name of one
 95322 of the lowercase letters of the portable character set. When a new file is examined, all marks may
 95323 be lost.

95324 **Return to Mark**

95325 *Synopsis:* ' *letter*

95326 Return to the position that was previously marked with the letter named by *letter*, making that
 95327 line the current position.

95328 **Return to Previous Position**

95329 *Synopsis:* ''

95330 Return to the position from which the last large movement command was executed (where a
 95331 "large movement" is defined as any movement of more than a screenful of lines). If no such
 95332 movements have been made, return to the beginning of the file.

95333

Search Forward for Pattern

95334

Synopsis: [*count*]/[!]*pattern*<newline>

95335

Display the screenful beginning with the *count*th line containing the pattern. The search shall start after the first line currently displayed. The null regular expression ('/' followed by a <newline>) shall repeat the search using the previous regular expression, with a default *count*. If the character '!' is included, the matching lines shall be those that do not contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be displayed.

95336

95337

95338

95339

95340

Search Backward for Pattern

95341

Synopsis: [*count*?][!]*pattern*<newline>

95342

Display the screenful beginning with the *count*th previous line containing the pattern. The search shall start on the last line before the first line currently displayed. The null regular expression ('?' followed by a <newline>) shall repeat the search using the previous regular expression, with a default *count*. If the character '!' is included, matching lines shall be those that do not contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be displayed.

95343

95344

95345

95346

95347

95348

Repeat Search

95349

Synopsis: [*count*]n

95350

Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was " /!" or "?!").

95351

95352

Repeat Search in Reverse

95353

Synopsis: [*count*]N

95354

Repeat the search in the opposite direction of the previous search for the *count*th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was " /!" or "?!").

95355

95356

Examine New File

95357

Synopsis: :e [*filename*]<newline>

95358

Examine a new file. If the *filename* argument is not specified, the current file (see the :n and :p commands below) shall be re-examined. The *filename* shall be subjected to the process of shell word expansions (see [Section 2.6](#), on page 2253); if more than a single pathname results, the effects are unspecified. If *filename* is a number sign ('#'), the previously examined file shall be re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file), an error message to this effect shall be displayed and the current file and screen shall not change.

95359

95360

95361

95362

95363

95364

Examine Next File

95365

Synopsis: [*count*]:n

95366

Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If *filename* refers to a non-seekable file, the results are unspecified.

95367

95368

Examine Previous File

95369

Synopsis: [*count*]:p

95370

Examine the previous file. If a number *count* is specified, the *count*th previous file shall be examined. If *filename* refers to a non-seekable file, the results are unspecified.

95371

95372

Go to Tag

95373

Synopsis: :t *tagstring*<newline>

95374

If the file containing the tag named by the *tagstring* argument is not the current file, examine the file, as if the :e command was executed with that file as the argument. Otherwise, or in addition, display the screenful beginning with the tag, as described for the -t option (see the OPTIONS section). If the *ctags* utility is not supported by the system, the use of :t produces undefined results.

95375

95376

95377

95378

95379

Invoke Editor

95380

Synopsis: v

95381

Invoke an editor to edit the current file being examined. If standard input is being examined, the results are unspecified. The name of the editor shall be taken from the environment variable *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the editor shall be invoked with a -c *linenumber* command line argument, where *linenumber* is the line number of the file line containing the display line currently displayed as the first line of the screen. It is implementation-defined whether line-setting options are passed to editors other than *vi* and *ex*.

95382

95383

95384

95385

95386

95387

95388

When the editor exits, *more* shall resume with the same file and screen as when the editor was invoked.

95389

95390

Display Position

95391

Synopsis: =
<control>-G

95392

95393

Write a message for which the information references the first byte of the line after the last line of the file on the screen. This message shall include the name of the file currently being examined, its number relative to the total number of files there are to examine, the line number in the file, the byte number and the total bytes in the file, and what percentage of the file precedes the current position. If *more* is reading from standard input, or the file is shorter than a single screen, the line number, the byte number, the total bytes, and the percentage need not be written.

95394

95395

95396

95397

95398

95399

Quit

95400

Synopsis: q
 :q
 ZZ

95401

95402

95403

Exit *more*.

95404

EXIT STATUS

95405

The following exit values shall be returned:

95406

0 Successful completion.

95407

>0 An error occurred.

95408
95409
95410
95411
95412
95413
95414**CONSEQUENCES OF ERRORS**

If an error is encountered accessing a file when using the **:n** command, *more* shall attempt to examine the next file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the **:p** command, *more* shall attempt to examine the previous file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the **:e** command, *more* shall remain in the current file and the final exit status shall not be affected.

95415
95416
95417
95418
95419**APPLICATION USAGE**

When the standard output is not a terminal, only the **-s** filter-modification option is effective. This is based on historical practice. For example, a typical implementation of *man* pipes its output through *more -s* to squeeze excess white space for terminal users. When *man* is piped to *lp*, however, it is undesirable for this squeezing to happen.

95420
95421**EXAMPLES**

The **-p** allows arbitrary commands to be executed at the start of each file. Examples are:

95422
95423

```
more -p G file1 file2
```

Examine each file starting with its last screenful.

95424
95425
95426

```
more -p 100 file1 file2
```

Examine each file starting with line 100 in the current position (usually the third line, so line 98 would be the first line written).

95427
95428
95429

```
more -p /100 file1 file2
```

Examine each file starting with the first line containing the string "100" in the current position

95430

RATIONALE95431
95432
95433
95434
95435
95436
95437

The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX file display program since it is more widely available than either the public-domain program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the features selected; it is almost fully upwards-compatible from the 4.3 BSD version in wide use and has become more amenable for *vi* users. Several features originally derived from various file editors, found in both *less* and *pg*, have been added to this volume of POSIX.1-200x as they have proved extremely popular with users.

95438
95439
95440
95441
95442

There are inconsistencies between *more* and *vi* that result from historical practice. For example, the single-character commands **h**, **f**, **b**, and **<space>** are screen movers in *more*, but cursor movers in *vi*. These inconsistencies were maintained because the cursor movements are not applicable to *more* and the powerful functionality achieved without the use of the control key justifies the differences.

95443
95444
95445

The tags interface has been included in a program that is not a text editor because it promotes another degree of consistent operation with *vi*. It is conceivable that the paging environment of *more* would be superior for browsing source code files in some circumstances.

95446
95447
95448
95449
95450
95451
95452
95453

The operating mode referred to for block-mode terminals effectively adds a **<newline>** to each Synopsis line that currently has none. So, for example, **d<newline>** would page one screenful. The mode could be triggered by a command line option, environment variable, or some other method. The details are not imposed by this volume of POSIX.1-200x because there are so few systems known to support such terminals. Nevertheless, it was considered that all systems should be able to support *more* given the exception cited for this small community of terminals because, in comparison to *vi*, the cursor movements are few and the command set relatively amenable to the optional **<newline>s**.

95454
95455
95456

Some versions of *more* provide a shell escaping mechanism similar to the *ex !* command. The standard developers did not consider that this was necessary in a paginator, particularly given the wide acceptance of multiple window terminals and job control features. (They chose to

95457 retain such features in the editors and *mailx* because the shell interaction also gives an
95458 opportunity to modify the editing buffer, which is not applicable to *more*.)

95459 The **-p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. The
95460 **+command** option is no longer specified by POSIX.1-200x but may be present in some
95461 implementations. In early proposals, it took a *pattern* argument, but historical *less* provided
95462 the *more* general facility of a command. It would have been desirable to use the same **-c** as *ex* and *vi*,
95463 but the letter was already in use.

95464 The text stating “from a non-rewindable stream ... implementations may limit the amount of
95465 backwards motion supported” would allow an implementation that permitted no backwards
95466 motion beyond text already on the screen. It was not possible to require a minimum amount of
95467 backwards motion that would be effective for all conceivable device types. The implementation
95468 should allow the user to back up as far as possible, within device and reasonable memory
95469 allocation constraints.

95470 Historically, non-printable characters were displayed using the ARPA standard mappings,
95471 which are as follows:

- 95472 1. Printable characters are left alone.
- 95473 2. Control characters less than `\177` are represented as followed by the character offset from
95474 the '@' character in the ASCII map; for example, `\007` is represented as 'G'.
- 95475 3. `\177` is represented as followed by ' ? '.

95476 The display of characters having their eighth bit set was less standard. Existing implementations
95477 use hex (`0x00`), octal (`\000`), and a meta-bit display. (The latter displayed characters with their
95478 eighth bit set as the two characters "M-", followed by the seven-bit display as described
95479 previously.) The latter probably has the best claim to historical practice because it was used with
95480 the **-v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

95481 No specific display format is required by POSIX.1-200x. Implementations are encouraged to
95482 conform to historic practice in the absence of any strong reason to diverge.

95483 FUTURE DIRECTIONS

95484 None.

95485 SEE ALSO

95486 [Chapter 2](#) (on page 2245), *ctags*, *ed*, *ex*, *vi*

95487 XBD [Chapter 8](#) (on page 159), [Section 9.2](#) (on page 168), [Section 9.3](#) (on page 169), [Section 12.2](#) +
95488 (on page 201)

95489 CHANGE HISTORY

95490 First released in Issue 4.

95491 Issue 5

95492 The FUTURE DIRECTIONS section is added.

95493 Issue 6

95494 This utility is marked as part of the User Portability Utilities option.

95495 The obsolescent SYNOPSIS is removed.

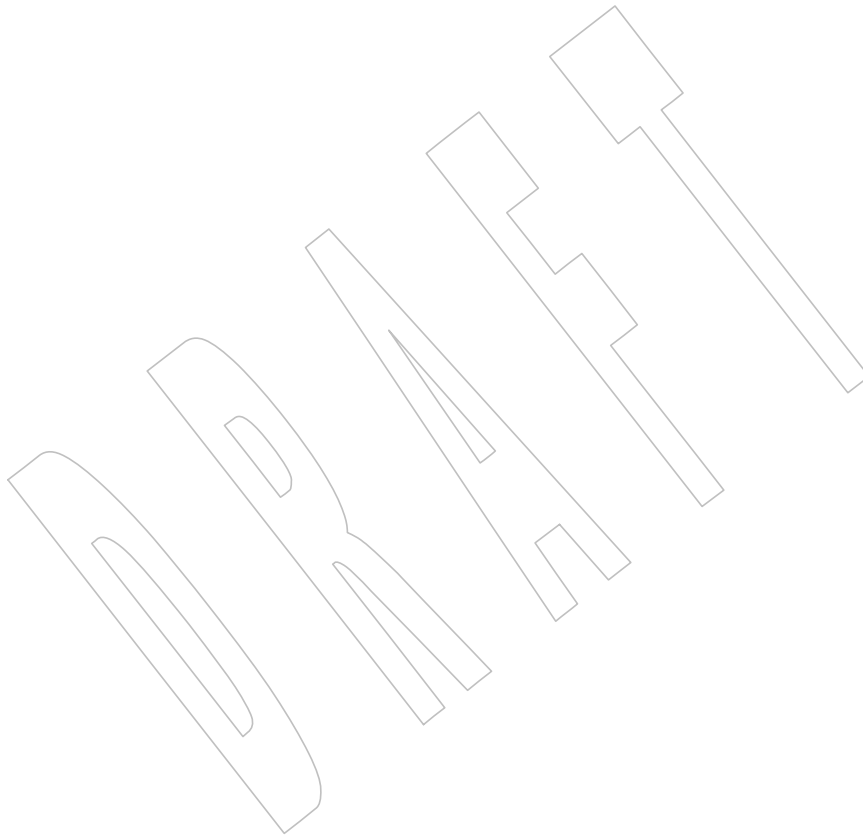
95496 The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:

- 95497 • Changes have been made as a result of IEEE PASC Interpretations 1003.2 #37 and #109.
- 95498 • The *more* utility should be able to handle underlined and emboldened displays of
95499 characters that are wider than a single column position.

95500
95501
95502
95503**Issue 7**

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized as an option delimiter in the OPTIONS section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



95504 **NAME**
 95505 mv — move files

95506 **SYNOPSIS**
 95507 mv [-if] *source_file target_file*
 95508 mv [-if] *source_file... target_dir*

95509 **DESCRIPTION**

95510 In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to
 95511 the destination specified by the *target_file*. This first synopsis form is assumed when the final
 95512 operand does not name an existing directory and is not a symbolic link referring to an existing
 95513 directory.

95514 In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a
 95515 destination file in the existing directory named by the *target_dir* operand, or referenced if
 95516 *target_dir* is a symbolic link referring to an existing directory. The destination path for each
 95517 *source_file* shall be the concatenation of the target directory, a single slash character, and the last
 95518 pathname component of the *source_file*. This second form is assumed when the final operand
 95519 names an existing directory.

95520 If any operand specifies an existing file of a type not specified by the System Interfaces volume
 95521 of POSIX.1-200x, the behavior is implementation-defined.

95522 For each *source_file* the following steps shall be taken:

- 95523 1. If the destination path exists, the *-f* option is not specified, and either of the following
 95524 conditions is true:
- 95525 a. The permissions of the destination path do not permit writing and the standard
 95526 input is a terminal.
 - 95527 b. The *-i* option is specified.

95528 the *mv* utility shall write a prompt to standard error and read a line from standard input.
 95529 If the response is not affirmative, *mv* shall do nothing more with the current *source_file*
 95530 and go on to any remaining *source_files*.

- 95531 2. The *mv* utility shall perform actions equivalent to the *rename()* function defined in the
 95532 System Interfaces volume of POSIX.1-200x, called with the following arguments:
- 95533 a. The *source_file* operand is used as the *old* argument.
 - 95534 b. The destination path is used as the *new* argument.

95535 If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any
 95536 remaining *source_files*. If this fails for any reasons other than those described for the *errno*
 95537 [EXDEV] in the System Interfaces volume of POSIX.1-200x, *mv* shall write a diagnostic
 95538 message to standard error, do nothing more with the current *source_file*, and go on to any
 95539 remaining *source_files*.

- 95540 3. If the destination path exists, and it is a file of type directory and *source_file* is not a file of
 95541 type directory, or it is a file not of type directory and *source_file* is a file of type directory,
 95542 *mv* shall write a diagnostic message to standard error, do nothing more with the current
 95543 *source_file*, and go on to any remaining *source_files*.

- 95544 4. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*
 95545 shall write a diagnostic message to standard error, do nothing more with the current
 95546 *source_file*, and go on to any remaining *source_files*.

95547 5. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the
 95548 destination path. If *source_file* or any of the files below it in the hierarchy are symbolic
 95549 links, the links themselves shall be duplicated, including their contents, rather than any
 95550 files to which they refer. The following characteristics of each file in the file hierarchy
 95551 shall be duplicated:

- 95552 • The time of last data modification and time of last access
- 95553 • The user ID and group ID
- 95554 • The file mode

95555 If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode
 95556 bits *S_ISUID* and *S_ISGID* shall not be duplicated.

95557 When files are duplicated to another file system, the implementation may require that the
 95558 process invoking *mv* has read access to each file being duplicated.

95559 If files being duplicated to another file system have hard links to other files, it is
 95560 unspecified whether the files copied to the new file system have the hard links preserved
 95561 or separate copies are created for the linked files.

95562 If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic
 95563 message to standard error, do nothing more with the current *source_file*, and go on to any
 95564 remaining *source_files*.

95565 If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic
 95566 message to standard error, but this failure shall not cause *mv* to modify its exit status.

95567 6. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv*
 95568 shall write a diagnostic message to the standard error, do nothing more with the current
 95569 *source_file*, and go on to any remaining *source_files*.

95570 OPTIONS

95571 The *mv* utility shall conform to XBD [Section 12.2](#) (on page 201).

95572 The following options shall be supported:

- 95573 **-f** Do not prompt for confirmation if the destination path exists. Any previous
 95574 occurrence of the **-i** option is ignored.
- 95575 **-i** Prompt for confirmation if the destination path exists. Any previous occurrence of
 95576 the **-f** option is ignored.

95577 Specifying more than one of the **-f** or **-i** options shall not be considered an error. The last option
 95578 specified shall determine the behavior of *mv*.

95579 OPERANDS

95580 The following operands shall be supported:

- 95581 *source_file* A pathname of a file or directory to be moved.
- 95582 *target_file* A new pathname for the file or directory being moved.
- 95583 *target_dir* A pathname of an existing directory into which to move the input files.

95584 STDIN

95585 The standard input shall be used to read an input line in response to each prompt specified in
 95586 the *STDERR* section. Otherwise, the standard input shall not be used.

95587 **INPUT FILES**95588 The input files specified by each *source_file* operand can be of any file type.95589 **ENVIRONMENT VARIABLES**95590 The following environment variables shall affect the execution of *mv*:

95591 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 95592 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 95593 variables used to determine the values of locale categories.)

95594 **LC_ALL** If set to a non-empty string value, override the values of all the other
 95595 internationalization variables.

95596 **LC_COLLATE**

95597 Determine the locale for the behavior of ranges, equivalence classes, and multi-
 95598 character collating elements used in the extended regular expression defined for
 95599 the **yesexpr** locale keyword in the **LC_MESSAGES** category.

95600 **LC_CTYPE**

95601 Determine the locale for the interpretation of sequences of bytes of text data as
 95602 characters (for example, single-byte as opposed to multi-byte characters in
 95603 arguments and input files), the behavior of character classes used in the extended
 95604 regular expression defined for the **yesexpr** locale keyword in the **LC_MESSAGES**
 category.

95605 **LC_MESSAGES**

95606 Determine the locale for the processing of affirmative responses that should be
 95607 used to affect the format and contents of diagnostic messages written to standard
 95608 error.

95609 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

95610 **ASYNCHRONOUS EVENTS**

95611 Default.

95612 **STDOUT**

95613 Not used.

95614 **STDERR**

95615 Prompts shall be written to the standard error under the conditions specified in the
 95616 DESCRIPTION section. The prompts shall contain the destination pathname, but their format is
 95617 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic
 95618 messages.

95619 **OUTPUT FILES**

95620 The output files may be of any file type.

95621 **EXTENDED DESCRIPTION**

95622 None.

95623 **EXIT STATUS**

95624 The following exit values shall be returned:

95625 0 All input files were moved successfully.

95626 >0 An error occurred.

95627 **CONSEQUENCES OF ERRORS**

95628 If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may
 95629 leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify
 95630 both *source_file* and the destination path simultaneously; termination at any point shall leave
 95631 either *source_file* or the destination path complete.

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

EXAMPLES

If the current directory contains only files **a** (of any type defined by the System Interfaces volume of POSIX.1-200x), **b** (also of any type), and a directory **c**:

```
mv a b c
mv c d
```

results with the original files **a** and **b** residing in the directory **d** in the current directory.

RATIONALE

Early proposals diverged from the SVID and BSD historical practice in that they required that when the destination path exists, the `-f` option is not specified, and input is not a terminal, `mv` fails. This was done for compatibility with `cp`. The current text returns to historical practice. It should be noted that this is consistent with the `rename()` function defined in the System Interfaces volume of POSIX.1-200x, which does not require write permission on the target.

For absolute clarity, paragraph (1), describing the behavior of `mv` when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD `mv` deletes all existing destination paths without prompting, even when `-i` is specified; this is inconsistent with the behavior of the 4.3 BSD `cp` utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The standard developers decided that use of `-i` is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds to standard input.

The `rename()` function is able to move directories within the same file system. Some historical versions of `mv` have been able to move directories, but not to a different file system. The standard developers considered that this was an annoying inconsistency, so this volume of POSIX.1-200x requires directories to be able to be moved even across file systems. There is no `-R` option to confirm that moving a directory is actually intended, since such an option was not required for moving directories in historical practice. Requiring the application to specify it sometimes, depending on the destination, seemed just as inconsistent. The semantics of the `rename()` function were preserved as much as possible. For example, `mv` is not permitted to “rename” files to or from directories, even though they might be empty and removable.

Historic implementations of `mv` did not exit with a non-zero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the `-f` option or using the `-i` option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

When `mv` is dealing with a single file system and `source_file` is a symbolic link, the link itself is moved as a consequence of the dependence on the `rename()` functionality, per the

95681 DESCRIPTION. Across file systems, this has to be made explicit.

95682 **FUTURE DIRECTIONS**

95683 None.

95684 **SEE ALSO**

95685 *cp, ln*

95686 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) |

95687 XSH *rename()*

95688 **CHANGE HISTORY**

95689 First released in Issue 2.

95690 **Issue 6**

95691 The *mv* utility is changed to describe processing of symbolic links as specified in the
95692 IEEE P1003.2b draft standard.

95693 The APPLICATION USAGE section is added.

95694 **Issue 7**

95695 SD5-XCU-ERN-13 is applied, making an editorial correction to the SYNOPSIS.

95696 SD5-XCU-ERN-51 is applied to the DESCRIPTION, defining the behavior for when files are
95697 being duplicated to another file system while having hard links.

95698 Changes are made related to support for finegrained timestamps. +

DRAFT

95699 **NAME**
 95700 `newgrp` — change to a new group

95701 **SYNOPSIS**
 95702 `newgrp [-l] [group]`

95703 **DESCRIPTION**
 95704 The *newgrp* utility shall create a new shell execution environment with a new real and effective
 95705 group identification. Of the attributes listed in [Section 2.12](#) (on page 2277), the new shell
 95706 execution environment shall retain the working directory, file creation mask, and exported
 95707 variables from the previous environment (that is, open files, traps, unexported variables, alias
 95708 definitions, shell functions, and *set* options may be lost). All other aspects of the process
 95709 environment that are preserved by the *exec* family of functions defined in the System Interfaces
 95710 volume of POSIX.1-200x shall also be preserved by *newgrp*; whether other aspects are preserved
 95711 is unspecified.

95712 A failure to assign the new group identifications (for example, for security or password-related
 95713 reasons) shall not prevent the new shell execution environment from being created.

95714 The *newgrp* utility shall affect the supplemental groups for the process as follows:

- 95715 • On systems where the effective group ID is normally in the supplementary group list (or
 95716 whenever the old effective group ID actually is in the supplementary group list):
 - 95717 — If the new effective group ID is also in the supplementary group list, *newgrp* shall
 95718 change the effective group ID.
 - 95719 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add
 95720 the new effective group ID to the list, if there is room to add it.
- 95721 • On systems where the effective group ID is not normally in the supplementary group list
 95722 (or whenever the old effective group ID is not in the supplementary group list):
 - 95723 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete
 95724 it.
 - 95725 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if
 95726 there is room.

95727 **Note:** The System Interfaces volume of POSIX.1-200x does not specify whether the effective group ID
 95728 of a process is included in its supplementary group list.

95729 With no operands, *newgrp* shall change the effective group back to the groups identified in the
 95730 user's user entry, and shall set the list of supplementary groups to that set in the user's group
 95731 database entries.

95732 If the first argument is '-', the results are unspecified.

95733 If a password is required for the specified group, and the user is not listed as a member of that
 95734 group in the group database, the user shall be prompted to enter the correct password for that
 95735 group. If the user is listed as a member of that group, no password shall be requested. If no
 95736 password is required for the specified group, it is implementation-defined whether users not
 95737 listed as members of that group can change to that group. Whether or not a password is
 95738 required, implementation-defined system accounting or security mechanisms may impose
 95739 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and
 95740 suppress the changing of the group identification.

newgrp

Utilities

95741
95742
95743
95744
95745
95746
95747
95748
95749
95750
95751
95752
95753
95754
95755
95756
95757
95758
95759
95760
95761
95762
95763
95764
95765
95766
95767
95768
95769
95770
95771
95772
95773
95774
95775
95776
95777
95778
95779
95780
95781
95782**OPTIONS**

The *newgrp* utility shall conform to XBD [Section 12.2](#) (on page 201), except for the unspecified usage of `'-'`.

The following option shall be supported:

`-l` (The letter ell.) Change the environment to what would be expected if the user actually logged in again.

OPERANDS

The following operand shall be supported:

group A group name from the group database or a non-negative numeric group ID. Specifies the group ID to which the real and effective group IDs shall be set. If *group* is a non-negative numeric string and exists in the group database as a group name (see *getgrnam()*), the numeric group ID associated with that group name shall be used as the group ID.

STDIN

Not used.

INPUT FILES

The file `/dev/tty` shall be used to read a single line of text for password checking, when one is required.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *newgrp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and a prompt string for a password, if one is required. Diagnostic messages may be written in cases where the exit status is not available. See the EXIT STATUS section.

OUTPUT FILES

None.

95783 **EXTENDED DESCRIPTION**

95784 None.

95785 **EXIT STATUS**

95786 If *newgrp* succeeds in creating a new shell execution environment, whether or not the group
 95787 identification was changed successfully, the exit status shall be the exit status of the shell.
 95788 Otherwise, the following exit value shall be returned:

95789 >0 An error occurred.

95790 **CONSEQUENCES OF ERRORS**

95791 The invoking shell may terminate.

95792 **APPLICATION USAGE**

95793 There is no convenient way to enter a password into the group database. Use of group
 95794 passwords is not encouraged, because by their very nature they encourage poor security
 95795 practices. Group passwords may disappear in the future.

95796 A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with
 95797 *newgrp*, which in turn overlays itself with a new shell after changing group. On some
 95798 implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

95799 The *newgrp* command is intended only for use from an interactive terminal. It does not offer a
 95800 useful interface for the support of applications.

95801 The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it
 95802 successfully invokes a new shell and the rest of the original shell script is bypassed when the
 95803 new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems.
 95804 But usage such as:

```
95805 newgrp foo
95806 echo $?
```

95807 is not useful because the new shell might not have access to any status *newgrp* may have
 95808 generated (and most historical systems do not provide this status). A zero status echoed here
 95809 does not necessarily indicate that the user has changed to the new group successfully. Following
 95810 *newgrp* with the *id* command provides a portable means of determining whether the group
 95811 change was successful or not.

95812 **EXAMPLES**

95813 None.

95814 **RATIONALE**

95815 Most historical implementations use one of the *exec* functions to implement the behavior of
 95816 *newgrp*. Errors detected before the *exec* leave the environment unchanged, while errors detected
 95817 after the *exec* leave the user in a changed environment. While it would be useful to have *newgrp*
 95818 issue a diagnostic message to tell the user that the environment changed, it would be
 95819 inappropriate to require this change to some historical implementations.

95820 The password mechanism is allowed in the group database, but how this would be
 95821 implemented is not specified.

95822 The *newgrp* utility was retained in this volume of POSIX.1-200x, even given the existence of the
 95823 multiple group permissions feature in the System Interfaces volume of POSIX.1-200x, for several
 95824 reasons. First, in some implementations, the group ownership of a newly created file is
 95825 determined by the group of the directory in which the file is created, as allowed by the System
 95826 Interfaces volume of POSIX.1-200x; on other implementations, the group ownership of a newly
 95827 created file is determined by the effective group ID. On implementations of the latter type,
 95828 *newgrp* allows files to be created with a specific group ownership. Finally, many
 95829 implementations use the real group ID in accounting, and on such systems, *newgrp* allows the
 95830 accounting identity of the user to be changed.

95831
95832
95833
95834
95835
95836
95837
95838
95839
95840
95841
95842
95843
95844
95845
95846
95847
95848
95849
95850

FUTURE DIRECTIONS

None.

SEE ALSO

[Chapter 2](#) (on page 2245), *sh*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

XSH *exec*, *getgrnam()*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS requirement.

Issue 7

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first argument is '- '.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

The *newgrp* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

95851 **NAME**
 95852 `nice` — invoke a utility with an altered nice value

95853 **SYNOPSIS**
 95854 `nice [-n increment] utility [argument...]`

95855 **DESCRIPTION**
 95856 The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see
 95857 XBD [Section 3.238](#), on page 66). With no options, the executed utility shall be run with a nice
 95858 value that is some implementation-defined quantity greater than or equal to the nice value of the
 95859 current process. If the user lacks appropriate privileges to affect the nice value in the requested
 95860 manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be
 95861 written to standard error, but this shall not prevent the invocation of *utility* or affect the exit
 95862 status.

95863 **OPTIONS**
 95864 The *nice* utility shall conform to XBD [Section 12.2](#) (on page 201).

95865 The following option is supported:

95866 `-n increment` A positive or negative decimal integer which shall have the same effect on the
 95867 execution of the utility as if the utility had called the *nice()* function with the
 95868 numeric value of the *increment* option-argument.

95869 **OPERANDS**
 95870 The following operands shall be supported:

95871 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 95872 special built-in utilities in [Section 2.14](#) (on page 2280), the results are undefined.

95873 *argument* Any string to be supplied as an argument when invoking the utility named by the
 95874 *utility* operand.

95875 **STDIN**
 95876 Not used.

95877 **INPUT FILES**
 95878 None.

95879 **ENVIRONMENT VARIABLES**
 95880 The following environment variables shall affect the execution of *nice*:

95881 *LANG* Provide a default value for the internationalization variables that are unset or null.
 95882 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 95883 variables used to determine the values of locale categories.)

95884 *LC_ALL* If set to a non-empty string value, override the values of all the other
 95885 internationalization variables.

95886 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 95887 characters (for example, single-byte as opposed to multi-byte characters in
 95888 arguments).

95889 *LC_MESSAGES*
 95890 Determine the locale that should be used to affect the format and contents of
 95891 diagnostic messages written to standard error.

95892 *XSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

95893 *PATH* Determine the search path used to locate the utility to be invoked. See XBD |
95894 Chapter 8 (on page 159).

95895 **ASYNCHRONOUS EVENTS**

95896 Default.

95897 **STDOUT**

95898 Not used.

95899 **STDERR**

95900 The standard error shall be used only for diagnostic messages.

95901 **OUTPUT FILES**

95902 None.

95903 **EXTENDED DESCRIPTION**

95904 None.

95905 **EXIT STATUS**

95906 If *utility* is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise, the *nice*
95907 utility shall exit with one of the following values:

95908 1-125 An error occurred in the *nice* utility.

95909 126 The utility specified by *utility* was found but could not be invoked.

95910 127 The utility specified by *utility* could not be found.

95911 **CONSEQUENCES OF ERRORS**

95912 Default.

95913 **APPLICATION USAGE**

95914 The only guaranteed portable uses of this utility are:

95915 *nice utility*

95916 Run *utility* with the default higher or equal nice value.

95917 *nice -n <positive integer> utility*

95918 Run *utility* with a higher nice value.

95919 On some implementations they have no discernible effect on the invoked utility and on some
95920 others they are exactly equivalent.

95921 Historical systems have frequently supported the *<positive integer>* up to 20. Since there is no
95922 error penalty associated with guessing a number that is too high, users without access to the
95923 system conformance document (to see what limits are actually in place) could use the historical 1
95924 to 20 range or attempt to use very large numbers if the job should be truly low priority.

95925 The nice value of a process can be displayed using the command:

95926 `ps -o nice`

95927 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
95928 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
95929 utility exited with an error indication”. The value 127 was chosen because it is not commonly
95930 used for other meanings; most utilities use small values for “normal error conditions” and the
95931 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
95932 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
95933 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
95934 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
95935 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
95936 any other reason.

EXAMPLES

None.

RATIONALE

The 4.3 BSD version of *nice* does not check whether *increment* is a valid decimal integer. The command *nice -x utility*, for example, would be treated the same as the command *nice --1 utility*. If the user does not have appropriate privileges, this results in a “permission denied” error. This is considered a bug.

When a user without appropriate privileges gives a negative *increment*, System V treats it like the command *nice -0 utility*, while 4.3 BSD writes a “permission denied” message and does not run the utility. The standard specifies the System V behavior together with an optional BSD-style “permission denied” message.

The C shell has a built-in version of *nice* that has a different interface from the one described in this volume of POSIX.1-200x.

The term “utility” is used, rather than “command”, to highlight the fact that shell compound commands, pipelines, and so on, cannot be used. Special built-ins also cannot be used. However, “utility” includes user application programs and shell scripts, not just utilities defined in this volume of POSIX.1-200x.

Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the default nice value being the midpoint of that range. By default, they raise the nice value of the executed utility by 10.

Some historical documentation states that the *increment* value must be within a fixed range. This is misleading; the valid *increment* values on any invocation are determined by the current process nice value, which is not always the default.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of POSIX.1-200x make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of POSIX.1-200x, and others may have no effect at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are possible.

Earlier versions of this standard allowed a *-increment* option. This form is no longer specified by POSIX.1-200x but may be present in some implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

[Chapter 2](#) (on page 2245), *renice*

[XBD Section 3.238](#) (on page 66), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

XSH *nice*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

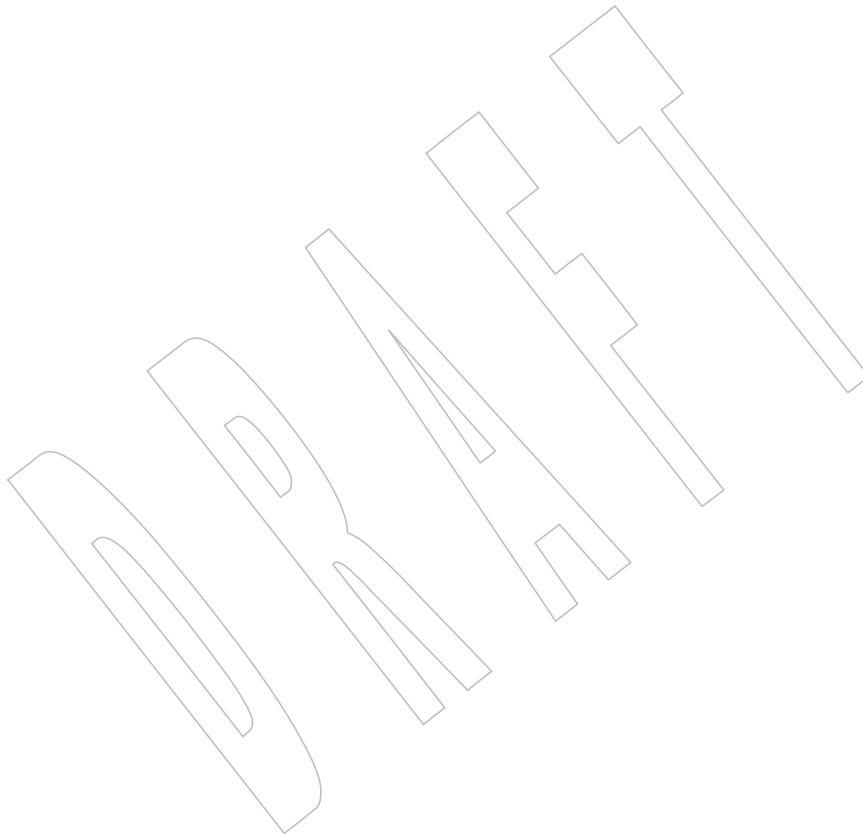
IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/18 is applied, deleting a paragraph of RATIONALE that referred to text no longer in the standard.

95983
95984
95985
95986
95987
95988**Issue 7**

Austin Group Interpretation 1003.1-2001 #027 is applied. +

SD5-XCU-ERN-32 and SD5-XCU-ERN-33 are applied, updating the DESCRIPTION, APPLICATION USAGE, and RATIONALE sections.

The *nice* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities. -



95989 **NAME**

95990 nl — line numbering filter

95991 **SYNOPSIS**

```
95992 xSI nl [-p] [-b type] [-d delim] [-f type] [-h type] [-i incr] [-l num]
95993 [-n format] [-s sep] [-v startnum] [-w width] [file]
```

95994 **DESCRIPTION**

95995 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and
 95996 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional
 95997 functionality may be provided in accordance with the command options in effect.

95998 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at
 95999 the start of each logical page. A logical page consists of a header, a body, and a footer section.
 96000 Empty sections are valid. Different line numbering options are independently available for
 96001 header, body, and footer (for example, no numbering of header and footer lines while
 96002 numbering blank lines only in the body).

96003 The starts of logical page sections shall be signaled by input lines containing nothing but the
 96004 following delimiter characters:

Line	Start of
\: \: \:	Header
\: \:	Body
\:	Footer

96009 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

96010 **OPTIONS**

96011 The *nl* utility shall conform to XBD [Section 12.2](#) (on page 201). Only one file can be named.

96012 The following options shall be supported:

96013 **-b *type*** Specify which logical page body lines shall be numbered. Recognized *types* and
 96014 their meaning are:

96015 **a** Number all lines.

96016 **t** Number only non-empty lines.

96017 **n** No line numbering.

96018 ***pstring*** Number only lines that contain the basic regular expression specified in
 96019 *string*.

96020 The default *type* for logical page body shall be **t** (text lines numbered).

96021 **-d *delim*** Specify the delimiter characters that indicate the start of a logical page section.
 96022 These can be changed from the default characters "\:" to two user-specified
 96023 characters. If only one character is entered, the second character shall remain the
 96024 default character ':'.

96025 **-f *type*** Specify the same as **b *type*** except for footer. The default for logical page footer shall
 96026 be **n** (no lines numbered).

96027 **-h *type*** Specify the same as **b *type*** except for header. The default *type* for logical page
 96028 header shall be **n** (no lines numbered).

- 96029 **-i *incr*** Specify the increment value used to number logical page lines. The default shall be
96030 1.
- 96031 **-l *num*** Specify the number of blank lines to be considered as one. For example, **-l 2** results
96032 in only the second adjacent blank line being numbered (if the appropriate **-h a**,
96033 **-b a**, or **-f a** option is set). The default shall be 1.
- 96034 **-n *format*** Specify the line numbering format. Recognized values are: **ln**, left justified, leading
96035 zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,
96036 leading zeros kept. The default *format* shall be **rn** (right justified).
- 96037 **-p** Specify that numbering should not be restarted at logical page delimiters.
- 96038 **-s *sep*** Specify the characters used in separating the line number and the corresponding
96039 text line. The default *sep* shall be a <tab>.
- 96040 **-v *startnum*** Specify the initial value used to number logical page lines. The default shall be 1.
- 96041 **-w *width*** Specify the number of characters to be used for the line number. The default *width*
96042 shall be 6.

OPERANDS

The following operand shall be supported:

file A pathname of a text file to be line-numbered.

STDIN

The standard input is a text file that is used if no *file* operand is given.

INPUT FILES

The input file named by the *file* operand is a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nl*:

LANG Provide a default value for the internationalization variables that are unset or null. |
96052 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
96053 variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other
96055 internationalization variables.

LC_COLLATE Determine the locale for the behavior of ranges, equivalence classes, and multi-
96057 character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as
96060 characters (for example, single-byte as opposed to multi-byte characters in
96061 arguments and input files), the behavior of character classes within regular
96062 expressions, and for deciding which characters are in character class **graph** (for the
96063 **-b t**, **-f t**, and **-h t** options).

LC_MESSAGES Determine the locale that should be used to affect the format and contents of
96065 diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of **LC_MESSAGES**.

ASYNCHRONOUS EVENTS

Default.

96071 **STDOUT**

96072 The standard output shall be a text file in the following format:

96073 "%s%s%s", <line number>, <separator>, <input line>

96074 where <line number> is one of the following numeric formats:

96075 %6d When the **rn** format is used (the default; see **-n**).96076 %06d When the **rz** format is used.96077 %-6d When the **ln** format is used.96078 <empty> When line numbers are suppressed for a portion of the page; the <separator> is also
96079 suppressed.96080 In the preceding list, the number 6 is the default width; the **-w** option can change this value.96081 **STDERR**

96082 The standard error shall be used only for diagnostic messages.

96083 **OUTPUT FILES**

96084 None.

96085 **EXTENDED DESCRIPTION**

96086 None.

96087 **EXIT STATUS**

96088 The following exit values shall be returned:

96089 0 Successful completion.

96090 >0 An error occurred.

96091 **CONSEQUENCES OF ERRORS**

96092 Default.

96093 **APPLICATION USAGE**96094 In using the **-d delim** option, care should be taken to escape characters that have special meaning
96095 to the command interpreter.96096 **EXAMPLES**

96097 The command:

96098 `nl -v 10 -i 10 -d \!+ file1`96099 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is
96100 "!+". Note that the '!' has to be escaped when using *csh* as a command interpreter because of
96101 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any
96102 harm.96103 **RATIONALE**

96104 None.

96105 **FUTURE DIRECTIONS**

96106 None.

96107 **SEE ALSO**96108 *pr*

96109 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

+

96110
96111
96112
96113
96114
96115
96116
96117
96118
96119

CHANGE HISTORY

First released in Issue 2.

Issue 5

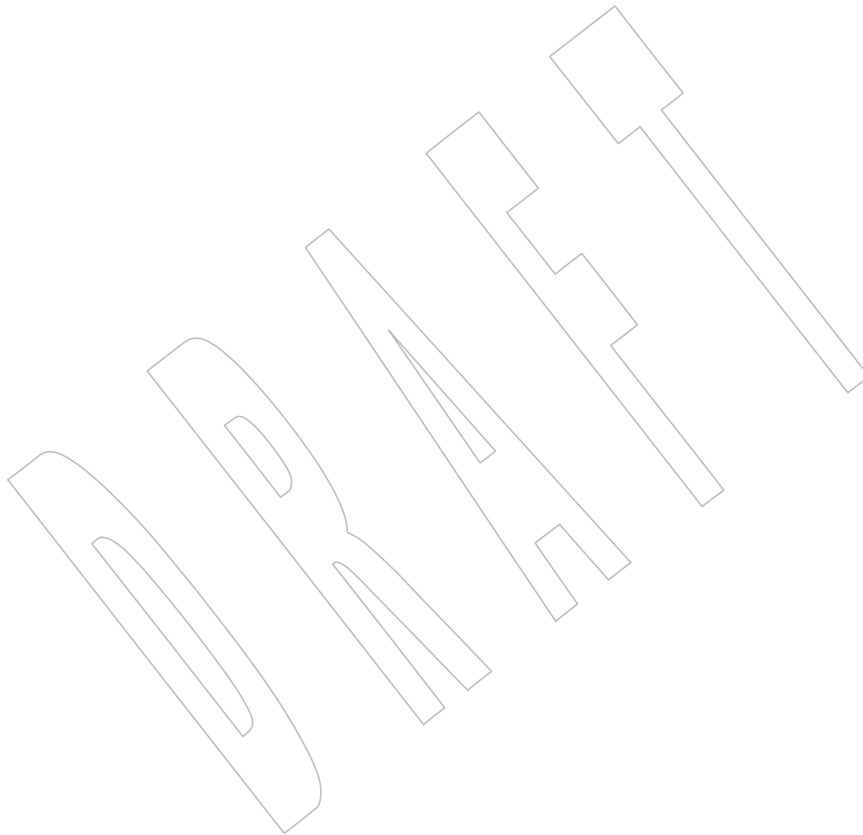
The option `[-f type]` is added to the SYNOPSIS. The option descriptions are presented in alphabetic order. The description of `-bt` is changed to “Number only non-empty lines”.

Issue 6

The obsolescent behavior allowing the options to be intermingled with the optional *file* operand is removed.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



96120 **NAME**96121 nm — write the name list of an object file (**DEVELOPMENT**)96122 **SYNOPSIS**96123 SD nm [-APv] [-g|-u] [-t *format*] *file...*96124 XSI nm [-APv] [-efox] [-g|-u] [-t *format*] *file...*96125 **DESCRIPTION**96126 The *nm* utility shall display symbolic information appearing in the object file, executable file, or
96127 object-file library named by *file*. If no symbolic information is available for a valid input file, the
96128 *nm* utility shall report that fact, but not consider it an error condition.96129 XSI The default base used when numeric values are written is unspecified. On XSI-conformant
96130 systems, it shall be decimal.96131 **OPTIONS**96132 The *nm* utility shall conform to XBD [Section 12.2](#) (on page 201).

96133 The following options shall be supported:

96134 **-A** Write the full pathname or library name of an object on each line.96135 XSI **-e** Write only external (global) and static symbol information.96136 XSI **-f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally
96137 suppressed.96138 **-g** Write only external (global) symbol information.96139 XSI **-o** Write numeric values in octal (equivalent to **-t o**).96140 **-P** Write information in a portable output format, as specified in the STDOUT section.96141 **-t *format*** Write each numeric value in the specified format. The format shall be dependent
96142 on the single character used as the *format* option-argument:

96143 XSI d The offset is written in decimal (default).

96144 o The offset is written in octal.

96145 x The offset is written in hexadecimal.

96146 **-u** Write only undefined symbols.96147 **-v** Sort output by value instead of alphabetically.96148 XSI **-x** Write numeric values in hexadecimal (equivalent to **-t x**).96149 **OPERANDS**

96150 The following operand shall be supported:

96151 *file* A pathname of an object file, executable file, or object-file library.96152 **STDIN**

96153 See the INPUT FILES section.

96154 **INPUT FILES**96155 The input file shall be an object file, an object-file library whose format is the same as those
96156 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept
96157 additional implementation-defined object library formats for the input file.

ENVIRONMENT VARIABLES

96158
96159 The following environment variables shall affect the execution of *nm*:

96160 *LANG* Provide a default value for the internationalization variables that are unset or null. |
96161 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
96162 variables used to determine the values of locale categories.)

96163 *LC_ALL* If set to a non-empty string value, override the values of all the other
96164 internationalization variables.

96165 *LC_COLLATE*
96166 Determine the locale for character collation information for the symbol-name and
96167 symbol-value collation sequences.

96168 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
96169 characters (for example, single-byte as opposed to multi-byte characters in
96170 arguments).

96171 *LC_MESSAGES*
96172 Determine the locale that should be used to affect the format and contents of
96173 diagnostic messages written to standard error.

96174 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

96175
96176 Default.

STDOUT

96177
96178 If symbolic information is present in the input files, then for each file or for each member of an
96179 archive, the *nm* utility shall write the following information to standard output. By default, the
96180 format is unspecified, but the output shall be sorted alphabetically by symbol name:

- 96181 • Library or object name, if **-A** is specified
- 96182 • Symbol name
- 96183 • Symbol type, which shall either be one of the following single characters or an
96184 implementation-defined type represented by a single character:
 - 96185 A Global absolute symbol.
 - 96186 a Local absolute symbol.
 - 96187 B Global “bss” (that is, uninitialized data space) symbol.
 - 96188 b Local bss symbol.
 - 96189 D Global data symbol.
 - 96190 d Local data symbol.
 - 96191 T Global text symbol.
 - 96192 t Local text symbol.
 - 96193 U Undefined symbol.
- 96194 • Value of the symbol
- 96195 • The size associated with the symbol, if applicable

96196 This information may be supplemented by additional information specific to the
96197 implementation.

96198 If the **-P** option is specified, the previous information shall be displayed using the following
96199 portable format. The three versions differ depending on whether **-t d**, **-t o**, or **-t x** was specified,

96200 respectively:

96201 "%s%s %s %d %d\n", <library/object name>, <name>, <type>,
96202 <value>, <size>

96203 "%s%s %s %o %o\n", <library/object name>, <name>, <type>,
96204 <value>, <size>

96205 "%s%s %s %x %x\n", <library/object name>, <name>, <type>,
96206 <value>, <size>

96207 where <library/object name> shall be formatted as follows:

- 96208 • If **-A** is not specified, <library/object name> shall be an empty string.
- 96209 • If **-A** is specified and the corresponding *file* operand does not name a library:
96210 "%s: ", <file>
- 96211 • If **-A** is specified and the corresponding *file* operand names a library. In this case,
96212 <object file> shall name the object file in the library containing the symbol being described:
96213 "%s[%s]: ", <file>, <object file>

96214 If **-A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is
96215 specified and it names a library, *nm* shall write a line identifying the object containing the
96216 following symbols before the lines containing those symbols, in the form:

- 96217 • If the corresponding *file* operand does not name a library:
96218 "%s:\n", <file>
- 96219 • If the corresponding *file* operand names a library; in this case, <object file> shall be the
96220 name of the file in the library containing the following symbols:
96221 "%s[%s]:\n", <file>, <object file>

96222 If **-P** is specified, but **-t** is not, the format shall be as if **-t x** had been specified.

96223 **STDERR**

96224 The standard error shall be used only for diagnostic messages.

96225 **OUTPUT FILES**

96226 None.

96227 **EXTENDED DESCRIPTION**

96228 None.

96229 **EXIT STATUS**

96230 The following exit values shall be returned:

- 96231 0 Successful completion.
- 96232 >0 An error occurred.

96233 **CONSEQUENCES OF ERRORS**

96234 Default.

APPLICATION USAGE

Mechanisms for dynamic linking make this utility less meaningful when applied to an executable file because a dynamically linked executable may omit numerous library routines that would be found in a statically linked executable.

EXAMPLES

None.

RATIONALE

Historical implementations of *nm* have used different bases for numeric output and supplied different default types of symbols that were reported. The `-t format` option, similar to that used in *od* and *strings*, can be used to specify the numeric base; `-g` and `-u` can be used to restrict the amount of output or the types of symbols included in the output.

The compromise of using `-t format versus` using `-d`, `-o`, and other similar options was necessary because of differences in the meaning of `-o` between implementations. The `-o` option from BSD has been provided here as `-A` to avoid confusion with the `-o` from System V (which has been provided here as `-t` and as `-o` on XSI-conformant systems).

The option list was significantly reduced from that provided by historical implementations.

The *nm* description is a subset of both the System V and BSD *nm* utilities with no specified default output.

It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to an executable file (because a dynamically linked executable file may omit numerous library routines that would be found in a statically linked executable file), but the value of *nm* during software development was judged to outweigh other limitations.

The default output format of *nm* is not specified because of differences in historical implementations. The `-P` option was added to allow some type of portable output format. After a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to create one that did not match the current format of any of these four systems. The format devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary depending on locale (because no English descriptions are included). All of the systems currently have the information available to use this format.

The format given in *nm* STDOUT uses spaces between the fields, which may be any number of <blank>s required to align the columns. The single-character types were selected to match historical practice, and the requirement that implementation additions also be single characters made parsing the information easier for shell scripts.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *c99*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

First released in Issue 2.

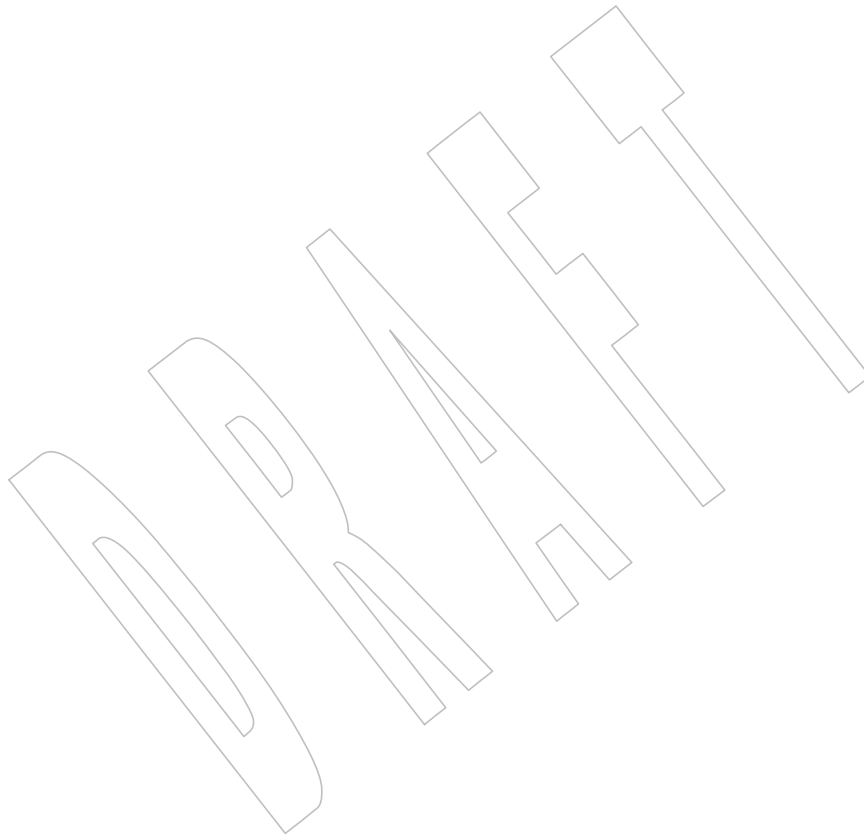
Issue 6

This utility is marked as supported when both the User Portability Utilities option and the Software Development Utilities option are supported.

96278
96279
96280
96281**Issue 7**

The *nm* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



96282 **NAME**
 96283 nohup — invoke a utility immune to hangups

96284 **SYNOPSIS**
 96285 nohup *utility* [*argument...*]

96286 **DESCRIPTION**
 96287 The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied
 96288 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be
 96289 set to be ignored.

96290 If standard input is associated with a terminal, the *nohup* utility may redirect standard input
 96291 from an unspecified file.

96292 If the standard output is a terminal, all output written by the named *utility* to its standard output
 96293 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot
 96294 be created or opened for appending, the output shall be appended to the end of the file
 96295 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be
 96296 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's
 96297 permission bits shall be set to S_IRUSR | S_IWUSR.

96298 If standard error is a terminal and standard output is open but is not a terminal, all output
 96299 written by the named utility to its standard error shall be redirected to the same open file
 96300 description as the standard output. If standard error is a terminal and standard output either is a
 96301 terminal or is closed, the same output shall instead be appended to the end of the **nohup.out** file
 96302 as described above.

96303 **OPTIONS**
 96304 None.

96305 **OPERANDS**
 96306 The following operands shall be supported:
 96307 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
 96308 special built-in utilities in [Section 2.14](#) (on page 2280), the results are undefined.
 96309 *argument* Any string to be supplied as an argument when invoking the utility named by the
 96310 *utility* operand.

96311 **STDIN**
 96312 Not used.

96313 **INPUT FILES**
 96314 None.

96315 **ENVIRONMENT VARIABLES**
 96316 The following environment variables shall affect the execution of *nohup*:

96317 *HOME* Determine the pathname of the user's home directory: if the output file **nohup.out**
 96318 cannot be created in the current directory, the *nohup* utility shall use the directory
 96319 named by *HOME* to create the file.

96320 *LANG* Provide a default value for the internationalization variables that are unset or null.
 96321 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 96322 variables used to determine the values of locale categories.)

96323 *LC_ALL* If set to a non-empty string value, override the values of all the other
 96324 internationalization variables.

96325 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 96326 characters (for example, single-byte as opposed to multi-byte characters in
 96327 arguments).

96328 *LC_MESSAGES*
 96329 Determine the locale that should be used to affect the format and contents of
 96330 diagnostic messages written to standard error.

96331 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

96332 *PATH* Determine the search path that is used to locate the utility to be invoked. See XBD
 96333 Chapter 8 (on page 159).

96334 ASYNCHRONOUS EVENTS

96335 The *nohup* utility shall take the standard action for all signals except that SIGHUP shall be
 96336 ignored.

96337 STDOUT

96338 If the standard output is not a terminal, the standard output of *nohup* shall be the standard
 96339 output generated by the execution of the *utility* specified by the operands. Otherwise, nothing
 96340 shall be written to the standard output.

96341 STDERR

96342 If the standard output is a terminal, a message shall be written to the standard error, indicating
 96343 the name of the file to which the output is being appended. The name of the file shall be either
 96344 **nohup.out** or **\$HOME/nohup.out**.

96345 OUTPUT FILES

96346 Output written by the named utility is appended to the file **nohup.out** (or **\$HOME/nohup.out**),
 96347 if the conditions hold as described in the DESCRIPTION.

96348 EXTENDED DESCRIPTION

96349 None.

96350 EXIT STATUS

96351 The following exit values shall be returned:

96352 126 The utility specified by *utility* was found but could not be invoked.

96353 127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be
 96354 found.

96355 Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.

96356 CONSEQUENCES OF ERRORS

96357 Default.

96358 APPLICATION USAGE

96359 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
 96360 an error occurs so that applications can distinguish “failure to find a utility” from “invoked
 96361 utility exited with an error indication”. The value 127 was chosen because it is not commonly
 96362 used for other meanings; most utilities use small values for “normal error conditions” and the
 96363 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
 96364 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
 96365 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
 96366 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
 96367 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for
 96368 any other reason.

EXAMPLES

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *nohup* applies to everything in the file.

Alternatively, the following command can be used to apply *nohup* to a complex command:

```
nohup sh -c 'complex-command-line' </dev/null
```

RATIONALE

The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails instead of trying to use **\$HOME/nohup.out**.

The *cs* utility has a built-in version of *nohup* that acts differently from the *nohup* defined in this volume of POSIX.1-200x.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

Historical versions of the *nohup* utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this volume of POSIX.1-200x.

Historical versions of *nohup* did not affect standard input, but that causes problems in the common scenario where the user logs into a system, types the command:

```
nohup make &
```

at the prompt, and then logs out. If standard input is not affected by *nohup*, the login session may not terminate for quite some time, since standard input remains open until *make* exits. To avoid this problem, POSIX.1-200x allows implementations to redirect standard input if it is a terminal. Since the behavior is implementation-defined, portable applications that may run into the problem should redirect standard input themselves. For example, instead of:

```
nohup make &
```

an application can invoke:

```
nohup make </dev/null &
```

FUTURE DIRECTIONS

None.

SEE ALSO

[Chapter 2](#) (on page 2245), *sh*

[XBD Chapter 8](#) (on page 159)

XSH *signal()*

CHANGE HISTORY

First released in Issue 2.

Issue 7

Austin Group Interpretations 1003.1-2001 #104, #105, and #106 are applied.

96410 NAME

96411 od — dump files in various formats

96412 SYNOPSIS

96413 od [-v] [-A *address_base*] [-j *skip*] [-N *count*] [-t *type_string*]...
96414 [*file*...]96415 XSI od [-bcdosx] [*file*] [[+]offset[.][b]]

96416 DESCRIPTION

96417 The *od* utility shall write the contents of its input files to standard output in a user-specified
96418 format.

96419 OPTIONS

96420 The *od* utility shall conform to XBD Section 12.2 (on page 201), except that the order of
96421 presentation of the **-t** options and the **-bcdosx** options is significant.

96422 The following options shall be supported:

96423 **-A** *address_base*96424 Specify the input offset base. See the EXTENDED DESCRIPTION section. The
96425 application shall ensure that the *address_base* option-argument is a character. The
96426 characters 'd', 'o', and 'x' specify that the offset base shall be written in
96427 decimal, octal, or hexadecimal, respectively. The character 'n' specifies that the
96428 offset shall not be written.96429 XSI **-b** Interpret bytes in octal. This shall be equivalent to **-t o1**.96430 XSI **-c** Interpret bytes as characters specified by the current setting of the *LC_CTYPE*
96431 category. Certain non-graphic characters appear as C escapes: "NUL=\0",
96432 "BS=\b", "FF=\f", "NL=\n", "CR=\r", "HT=\t"; others appear as 3-digit octal
96433 numbers.96434 XSI **-d** Interpret *words* (two-byte units) in unsigned decimal. This shall be equivalent to
96435 **-t u2**.96436 **-j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or
96437 seek past the first *skip* bytes in the concatenated input files. If the combined input is
96438 not at least *skip* bytes long, the *od* utility shall write a diagnostic message to
96439 standard error and exit with a non-zero exit status.96440 By default, the *skip* option-argument shall be interpreted as a decimal number.
96441 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;
96442 otherwise, with a leading '0', the offset shall be interpreted as an octal number.
96443 Appending the character 'b', 'k', or 'm' to offset shall cause it to be interpreted
96444 as a multiple of 512, 1024, or 1048576 bytes, respectively. If the *skip* number is
96445 hexadecimal, any appended 'b' shall be considered to be the final hexadecimal
96446 digit.96447 **-N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as
96448 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a
96449 hexadecimal number; otherwise, with a leading '0', it shall be interpreted as an
96450 octal number. If *count* bytes of input (after successfully skipping, if **-j** *skip* is
96451 specified) are not available, it shall not be considered an error; the *od* utility shall
96452 format the input that is available.

96453	XSI	-o	Interpret <i>words</i> (two-byte units) in octal. This shall be equivalent to -t o2 .
96454	XSI	-s	Interpret <i>words</i> (two-byte units) in signed decimal. This shall be equivalent to
96455		-t d2 .	
96456		-t <i>type_string</i>	
96457			Specify one or more output types. See the EXTENDED DESCRIPTION section. The
96458			application shall ensure that the <i>type_string</i> option-argument is a string specifying
96459			the types to be used when writing the input data. The string shall consist of the
96460			type specification characters a, c, d, f, o, u, and x, specifying named character,
96461			character, signed decimal, floating point, octal, unsigned decimal, and
96462			hexadecimal, respectively. The type specification characters d, f, o, u, and x can be
96463			followed by an optional unsigned decimal integer that specifies the number of
96464			bytes to be transformed by each instance of the output type. The type specification
96465			character f can be followed by an optional F, D, or L indicating that the conversion
96466			should be applied to an item of type float , double , or long double , respectively.
96467			The type specification characters d, o, u, and x can be followed by an optional C, S,
96468			I, or L indicating that the conversion should be applied to an item of type char ,
96469			short , int , or long , respectively. Multiple types can be concatenated within the
96470			same <i>type_string</i> and multiple -t options can be specified. Output lines shall be
96471			written for each type specified in the order in which the type specification
96472			characters are specified.
96473		-v	Write all input data. Without the -v option, any number of groups of output lines,
96474			which would be identical to the immediately preceding group of output lines
96475			(except for the byte offsets), shall be replaced with a line containing only an
96476			asterisk ('*').
96477	XSI	-x	Interpret <i>words</i> (two-byte units) in hexadecimal. This shall be equivalent to -t x2 .
96478	XSI		Multiple types can be specified by using multiple -bcdostx options. Output lines are written for
96479			each type specified in the order in which the types are specified.

OPERANDS

The following operands shall be supported:

96482		<i>file</i>	A pathname of a file to be read. If no <i>file</i> operands are specified, the standard input
96483			shall be used.
96484			If there are no more than two operands, none of the -A , -j , -N , -t , or -v options is
96485			specified, and either of the following is true: the first character of the last operand
96486			is a plus sign ('+'), or there are two operands and the first character of the last
96487	XSI		operand is numeric; the last operand shall be interpreted as an offset operand on
96488			XSI-conformant systems. Under these conditions, the results are unspecified on
96489			systems that are not XSI-conformant systems.
96490	XSI	[+] <i>offset</i> [.] [b]	The <i>offset</i> operand specifies the offset in the file where dumping is to commence.
96491			This operand is normally interpreted as octal bytes. If '.' is appended, the offset
96492			shall be interpreted in decimal. If 'b' is appended, the offset shall be interpreted
96493			in units of 512 bytes.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

96499 **ENVIRONMENT VARIABLES**96500 The following environment variables shall affect the execution of *od*:96501 *LANG* Provide a default value for the internationalization variables that are unset or null. |
96502 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
96503 variables used to determine the values of locale categories.)96504 *LC_ALL* If set to a non-empty string value, override the values of all the other
96505 internationalization variables.96506 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
96507 characters (for example, single-byte as opposed to multi-byte characters in
96508 arguments and input files).96509 *LC_MESSAGES*
96510 Determine the locale that should be used to affect the format and contents of
96511 diagnostic messages written to standard error.96512 *LC_NUMERIC*
96513 Determine the locale for selecting the radix character used when writing floating-
96514 point formatted output.96515 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.96516 **ASYNCHRONOUS EVENTS**

96517 Default.

96518 **STDOUT**

96519 See the EXTENDED DESCRIPTION section.

96520 **STDERR**

96521 The standard error shall be used only for diagnostic messages.

96522 **OUTPUT FILES**

96523 None.

96524 **EXTENDED DESCRIPTION**96525 The *od* utility shall copy sequentially each input file to standard output, transforming the input
96526 data according to the output types specified by the *-t* option or the *-bcdosx* options. If no
96527 output type is specified, the default output shall be as if *-t oS* had been specified.96528 The number of bytes transformed by the output type specifier *c* may be variable depending on
96529 the *LC_CTYPE* category.96530 The default number of bytes transformed by output type specifiers *d*, *f*, *o*, *u*, and *x* corresponds
96531 to the various C-language types as follows. If the *c99* compiler is present on the system, these
96532 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes
96533 may vary among systems that conform to POSIX.1-200x.

- 96534 • For the type specifier characters
- d*
- ,
- o*
- ,
- u*
- , and
- x*
- , the default number of bytes shall
-
- 96535 correspond to the size of the underlying implementation's basic integer type. For these
-
- 96536 specifier characters, the implementation shall support values of the optional number of
-
- 96537 bytes to be converted corresponding to the number of bytes in the C-language types
- char**
- ,
-
- 96538
- short**
- ,
- int**
- , and
- long**
- . These numbers can also be specified by an application as the
-
- 96539 characters 'C', 'S', 'I', and 'L', respectively. The implementation shall also support
-
- 96540 the values 1, 2, 4, and 8, even if it provides no C-Language types of those sizes. The
-
- 96541 implementation shall support the decimal value corresponding to the C-language type
-
- 96542
- long long**
- . The byte order used when interpreting numeric values is implementation-
-
- 96543 defined, but shall correspond to the order in which a constant of the corresponding type is
-
- 96544 stored in memory on the system.

- For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating-point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float**, **double**, and **long double**. These numbers can also be specified by an application as the characters `'F'`, `'D'`, and `'L'`, respectively.

The type specifier character `a` specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in the following table shall be written using the corresponding names for those characters.

Table 4-12 Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

Note: The "\012" value may be written either as `lf` or `nl`.

The type specifier character `c` specifies that bytes shall be interpreted as characters specified by the current setting of the `LC_CTYPE` locale category. Characters listed in the table in XBD Chapter 5 (on page 107) (`'\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`) shall be written as the corresponding escape sequences, except that backslash shall be written as a single backslash and a NUL shall be written as `'\0'`. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence `"**"` shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the `-j skip` or `-N count` option is specified along with the `c` type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more `<blank>s`.

If, as a result of the specification of the `-N` option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently with null bytes to write the last byte of the input.

Unless `-A n` is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any `<blank>s`, shall start at the first character of the output line, and shall be followed by one or more `<blank>s`. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any `<blank>s`.

96594 If no `-A` option is specified, the input offset base is unspecified.

96595 EXIT STATUS

96596 The following exit values shall be returned:

96597 0 All input files were processed successfully.

96598 >0 An error occurred.

96599 CONSEQUENCES OF ERRORS

96600 Default.

96601 APPLICATION USAGE

96602 XSI-conformant applications are warned not to use filenames starting with '+' or a first
96603 operand starting with a numeric character so that the old functionality can be maintained by
96604 implementations, unless they specify one of the `-A`, `-j`, or `-N` options. To guarantee that one of
96605 these filenames is always interpreted as a filename, an application could always specify the
96606 address base format with the `-A` option.

96607 EXAMPLES

96608 If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as
96609 standard input to the command:

96610 `od -A d -t a`

96611 on an implementation using an input block size of 16 bytes, the standard output, independent of
96612 the current locale setting, would be similar to:

```
96613 0000000 nul soh stx etx eot enq ack bel bs ht nl vt ff cr so si
96614 0000016 dle dc1 dc2 dc3 dc4 nak syn etb can em sub esc fs gs rs us
96615 0000032 sp ! " # $ % & ' ( ) * + , - . /
96616 0000048 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
96617 0000064 @ A B C D E F G H I J K L M N O
96618 0000080 P Q R S T U V W X Y Z [ \ ] ^ _
96619 0000096 ` a b c d e f g h i j k l m n o
96620 0000112 p q r s t u v w x y z { | } ~ del
96621 0000128
```

96622 Note that this volume of POSIX.1-200x allows `nl` or `lf` to be used as the name for the
96623 ISO/IEC 646:1991 standard IRV character with decimal value 10. The IRV names this character
96624 `lf` (line feed), but traditional implementations have referred to this character as newline (`nl`) and
96625 the POSIX locale character set symbolic name for the corresponding character is a `<newline>`.

96626 The command:

96627 `od -A o -t o2x2x -N 18`

96628 on a system with 32-bit words and an implementation using an input block size of 16 bytes
96629 could write 18 bytes in approximately the following format:

```
96630 0000000 032056 031440 041123 042040 052516 044530 020043 031464
96631          342e 3320 4253 4420 554e 4958 2023 3334
96632          342e3320 42534420 554e4958 20233334
96633 0000020 032472
96634          353a
96635          353a0000
96636 0000022
```

96637 The command:

96638 `od -A d -t f -t o4 -t x4 -N 24 -j 0x15`

96639 on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point

```

96640 format) would skip 21 bytes of input data and then write 24 bytes in approximately the
96641 following format:
96642 0000000 1.0000000000000000e+00 1.5735000000000000e+01
96643 07774000000 00000000000 10013674121 35341217270
96644 3ff00000 00000000 402f3851 eb851eb8
96645 0000016 1.4066823000000000e+02
96646 10030312542 04370303230
96647 40619562 23e18698
96648 0000024

```

RATIONALE

96649 The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently
 96650 *hexdump*. There were several objections to all of these based on the following reasons:
 96651

- 96652 • The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- 96653 • The *hexdump* description was much more complex than needed for a simple dump utility.
- 96654 • The *od* utility has been available on all historical implementations and there was no need to
 96655 create a new name for a utility so similar to the historical *od* utility.

96656 The original reasons for not standardizing historical *od* were also fairly widespread. Those
 96657 reasons are given below along with rationale explaining why the standard developers believe
 96658 that this version does not suffer from the indicated problem:

- 96659 • The BSD and System V versions of *od* have diverged, and the intersection of features
 96660 provided by both does not meet the needs of the user community. In fact, the System V
 96661 version only provides a mechanism for dumping octal bytes and **shorts**, signed and
 96662 unsigned decimal **shorts**, hexadecimal **shorts**, and ASCII characters. BSD added the ability
 96663 to dump **floats**, **doubles**, named ASCII characters, and octal, signed decimal, unsigned
 96664 decimal, and hexadecimal **longs**. The version presented here provides more normalized
 96665 forms for dumping bytes, **shorts**, **ints**, and **longs** in octal, signed decimal, unsigned
 96666 decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as
 96667 current locale characters.
- 96668 • It would not be possible to come up with a compatible superset of the BSD and System V
 96669 flags that met the requirements of the standard developers. The historical default *od* output
 96670 is the specified default output of this utility. None of the option letters chosen for this
 96671 version of *od* conflict with any of the options to historical versions of *od*.
- 96672 • On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps
 96673 of **ints**, even in the BSD version. Because of the way options are named, the name space
 96674 could not be extended to solve these problems. This is why the **-t** option was added (with
 96675 type specifiers more closely matched to the *printf()* formats used in the rest of this volume
 96676 of POSIX.1-200x) and the optional field sizes were added to the **d**, **f**, **o**, **u**, and **x** type
 96677 specifiers. It is also one of the reasons why the historical practice was not mandated as a
 96678 required obsolescent form of *od*. (Although the old versions of *od* are not listed as an
 96679 obsolescent form, implementations are urged to continue to recognize the older forms for
 96680 several more years.) The **a**, **c**, **f**, **o**, and **x** types match the meaning of the corresponding
 96681 format characters in the historical implementations of *od* except for the default sizes of the
 96682 fields converted. The **d** format is signed in this volume of POSIX.1-200x to match the
 96683 *printf()* notation. (Historical versions of *od* used **d** as a synonym for **u** in this version. The
 96684 System V implementation uses **s** for signed decimal; BSD uses **i** for signed decimal and **s**
 96685 for null-terminated strings.) Other than **d** and **u**, all of the type specifiers match format
 96686 characters in the historical BSD version of *od*.

96687 The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are
 96688 used even though it is recognized that there may be zero or more than one compiler for the

C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **shorts**, 2 bytes **ints**, and 4 bytes **longs**, while another compiler (or an option to the same compiler) uses 2 bytes **shorts**, 4 bytes **ints**, and 4 bytes **longs**.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of POSIX.1-200x or this volume of POSIX.1-200x. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**, **double**, and **long double**, respectively. If the optional *c99* utility is provided by the implementation and used as specified by this volume of POSIX.1-200x, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the *od* utility is able to interpret binary data output by such a program correctly.

This volume of POSIX.1-200x requires that the numeric values of these lengths be recognized by the *od* utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of **unsigned long** data elements using *od -t uL*.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The **-A** option now specifies the address base and the **-S** option specifies a starting offset.
- It would be difficult to break the dependence on U.S. ASCII to achieve an internationalized utility. It does not seem to be any harder for *od* to dump characters in the current locale than it is for the *ed* or *sed* I commands. The **c** type specifier does this without difficulty and is completely compatible with the historical implementations of the **c** format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The **a** type specifier (from the BSD **a** format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of their base codeset.

The use of "***" as an indication of continuation of a multi-byte character in **c** specifier output was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

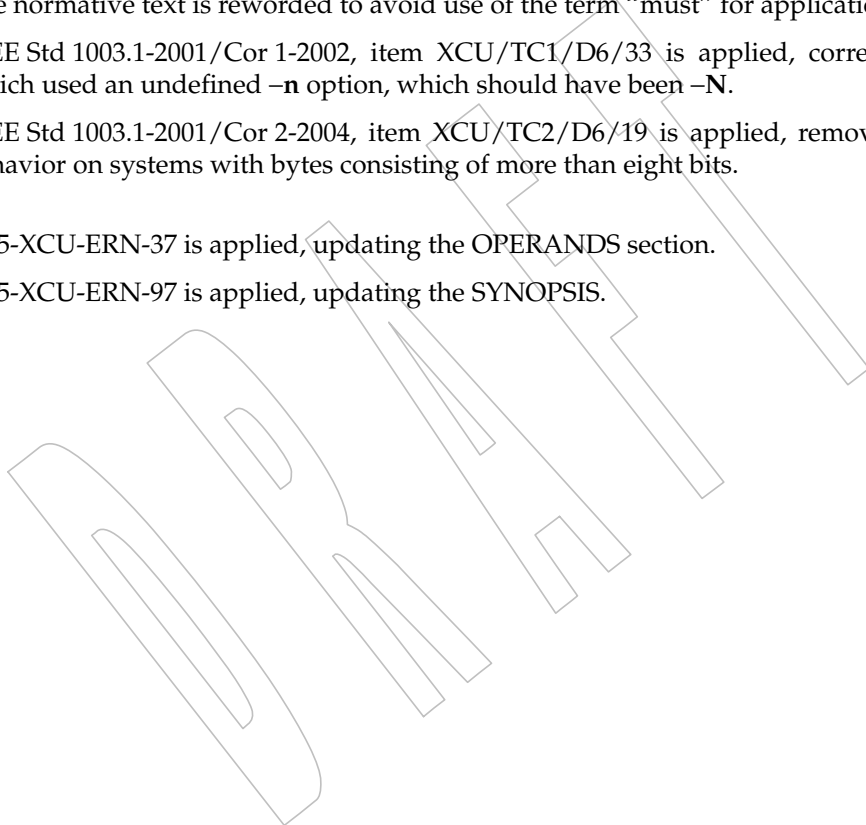
An early proposal used **-S** and **-n**, respectively, for the **-j** and **-N** options eventually selected. These were changed to avoid conflicts with historical implementations.

The original standard specified **-t o2** as the default when no output type was given. This was changed to **-t oS** (the length of a **short**) to accommodate a supercomputer implementation that historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at the same time to address an historical implementation that had no two-byte data types in its C compiler.

The use of a basic integer data type is intended to allow the implementation to choose a word size commonly used by applications on that architecture.

Earlier versions of this standard allowed for implementations with bytes other than eight bits, but this has been modified in this version.

96736	FUTURE DIRECTIONS	
96737	All option and operand interfaces marked XSI may be removed in a future version.	
96738	SEE ALSO	
96739	<i>c99</i> , <i>sed</i>	
96740	XBD Chapter 5 (on page 107), Chapter 8 (on page 159), Section 12.2 (on page 201)	+
96741	CHANGE HISTORY	
96742	First released in Issue 2.	
96743	Issue 5	
96744	In the description of the <code>-c</code> option, the phrase “This is equivalent to <code>-t c.</code> ” is deleted.	
96745	The FUTURE DIRECTIONS section is modified.	
96746	Issue 6	
96747	The <i>od</i> utility is changed to remove the assumption that short was a two-byte entity, as per the revisions in the IEEE P1003.2b draft standard.	
96748		
96749	The normative text is reworded to avoid use of the term “must” for application requirements.	
96750	IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/33 is applied, correcting the examples which used an undefined <code>-n</code> option, which should have been <code>-N</code> .	
96751		
96752	IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/19 is applied, removing text describing behavior on systems with bytes consisting of more than eight bits.	
96753		
96754	Issue 7	
96755	SD5-XCU-ERN-37 is applied, updating the OPERANDS section.	
96756	SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.	



96757 **NAME**
 96758 `paste` — merge corresponding or subsequent lines of files

96759 **SYNOPSIS**
 96760 `paste [-s] [-d list] file...`

96761 **DESCRIPTION**
 96762 The *paste* utility shall concatenate the corresponding lines of the given input files, and write the
 96763 resulting lines to standard output.

96764 The default operation of *paste* shall concatenate the corresponding lines of the input files. The
 96765 <newline> of every line except the line from the last input file shall be replaced with a <tab>.

96766 If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall
 96767 behave as though empty lines were read from the files on which end-of-file was detected, unless
 96768 the `-s` option is specified.

96769 **OPTIONS**
 96770 The *paste* utility shall conform to XBD [Section 12.2](#) (on page 201).

96771 The following options shall be supported:

96772 `-d list` Unless a backslash character appears in *list*, each character in *list* is an element
 96773 specifying a delimiter character. If a backslash character appears in *list*, the
 96774 backslash character and one or more characters following it are an element
 96775 specifying a delimiter character as described below. These elements specify one or
 96776 more delimiters to use, instead of the default <tab>, to replace the <newline> of
 96777 the input lines. The elements in *list* shall be used circularly; that is, when the list is
 96778 exhausted the first element from the list is reused. When the `-s` option is specified:

- The last <newline> in a file shall not be modified.
- The delimiter shall be reset to the first element of *list* after each *file* operand is processed.

96782 When the `-s` option is not specified:

- The <newline>s in the file specified by the last *file* operand shall not be modified.
- The delimiter shall be reset to the first element of *list* each time a line is processed from each file.

96787 If a backslash character appears in *list*, it and the character following it shall be
 96788 used to represent the following delimiter characters:

96789 `\n` <newline>.

96790 `\t` <tab>.

96791 `\\` Backslash character.

96792 `\0` Empty string (not a null character). If `'\0'` is immediately followed by the
 96793 character `'x'`, the character `'X'`, or any character defined by the `LC_CTYPE`
 96794 **digit** keyword (see XBD [Chapter 7](#), on page 121), the results are unspecified. |

96795 If any other characters follow the backslash, the results are unspecified.

96796 `-s` Concatenate all of the lines of each separate input file in command line order. The
 96797 <newline> of every line except the last line in each input file shall be replaced with
 96798 the <tab>, unless otherwise specified by the `-d` option.

96799 **OPERANDS**

96800 The following operand shall be supported:

96801 *file* A pathname of an input file. If '-' is specified for one or more of the *files*, the
 96802 standard input shall be used; the standard input shall be read one line at a time,
 96803 circularly, for each instance of '-'. Implementations shall support pasting of at
 96804 least 12 *file* operands.

96805 **STDIN**96806 The standard input shall be used only if one or more *file* operands is '-'. See the INPUT FILES
96807 section.96808 **INPUT FILES**

96809 The input files shall be text files, except that line lengths shall be unlimited.

96810 **ENVIRONMENT VARIABLES**96811 The following environment variables shall affect the execution of *paste*:

96812 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 96813 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 96814 used to determine the values of locale categories.)

96815 *LC_ALL* If set to a non-empty string value, override the values of all the other
 96816 internationalization variables.

96817 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 96818 characters (for example, single-byte as opposed to multi-byte characters in
 96819 arguments and input files).

96820 *LC_MESSAGES*96821 Determine the locale that should be used to affect the format and contents of
96822 diagnostic messages written to standard error.96823 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.96824 **ASYNCHRONOUS EVENTS**

96825 Default.

96826 **STDOUT**96827 Concatenated lines of input files shall be separated by the <tab> (or other characters under the
96828 control of the -d option) and terminated by a <newline>.96829 **STDERR**

96830 The standard error shall be used only for diagnostic messages.

96831 **OUTPUT FILES**

96832 None.

96833 **EXTENDED DESCRIPTION**

96834 None.

96835 **EXIT STATUS**

96836 The following exit values shall be returned:

96837 0 Successful completion.

96838 >0 An error occurred.

96839 **CONSEQUENCES OF ERRORS**96840 If one or more input files cannot be opened when the -s option is not specified, a diagnostic
96841 message shall be written to standard error, but no output is written to standard output. If the -s
96842 option is specified, the *paste* utility shall provide the default behavior described in Section 1.4 (on
96843 page 2235).

APPLICATION USAGE

When the escape sequences of the *list* option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the '\ ' as a special character.

Conforming applications should only use the specific backslash escaped delimiters presented in this volume of POSIX.1-200x. Historical implementations treat '\x', where 'x' is not in this list, as 'x', but future implementations are free to expand this list to recognize other common escapes similar to those accepted by *printf* and other standard utilities.

Most of the standard utilities work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in *file* that are longer than 500 + {LINE_MAX} bytes. The original file can be recreated from **file1** and **file2** using the command:

```
paste -d "\0" file1 file2 > file
```

The commands:

```
paste -d "\0" ...
paste -d " " ...
```

are not necessarily equivalent; the latter is not specified by this volume of POSIX.1-200x and may result in an error. The construct '\0' is used to mean "no separator" because historical versions of *paste* did not follow the syntax guidelines, and the command:

```
paste -d " " ...
```

could not be handled properly by *getopt()*.

EXAMPLES

1. Write out a directory in four columns:

```
ls | paste - - - -
```

2. Combine pairs of lines from a file into single lines:

```
paste -s -d "\t\n" file
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

[Section 1.4](#) (on page 2235), *cut*, *grep*, *pr*

[XBD Chapter 7](#) (on page 121), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

First released in Issue 2.

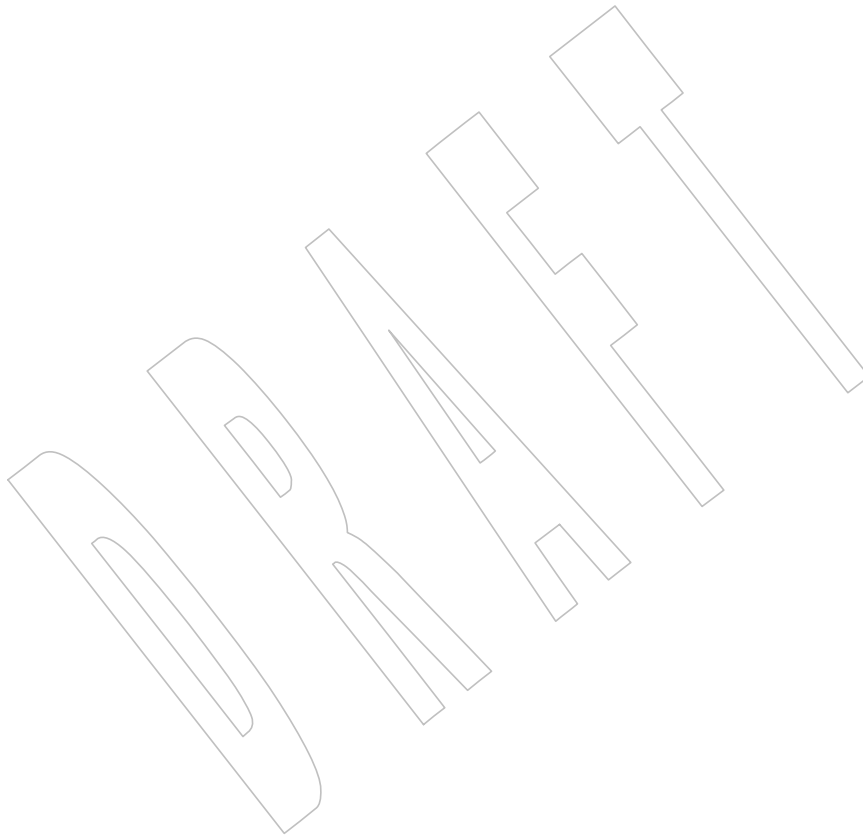
Issue 6

The normative text is reworded to avoid use of the term "must" for application requirements.

96885
96886

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



96887 NAME

96888 patch — apply changes to files

96889 SYNOPSIS

96890 patch [-blNR] [-c|-e|-n|-u] [-d *dir*] [-D *define*] [-i *patchfile*]
96891 [-o *outfile*] [-p *num*] [-r *rejectfile*] [*file*]

96892 DESCRIPTION

96893 The *patch* utility shall read a source (patch) file containing any of four forms of difference (diff)
96894 listings produced by the *diff* utility (normal, copied context, unified context, or in the style of *ed*)
96895 and apply those differences to a file. By default, *patch* shall read from the standard input.96896 The *patch* utility shall attempt to determine the type of the *diff* listing, unless overruled by a *-c*,
96897 *-e*, *-n*, or *-u* option.96898 If the patch file contains more than one patch, *patch* shall attempt to apply each of them as if they
96899 came from separate patch files. (In this case, the application shall ensure that the name of the
96900 patch file is determinable for each *diff* listing.)

96901 OPTIONS

96902 The *patch* utility shall conform to XBD Section 12.2 (on page 201).

96903 The following options shall be supported:

96904 **-b** Save a copy of the original contents of each modified file, before the differences are
96905 applied, in a file of the same name with the suffix **.orig** appended to it. If the file
96906 already exists, it shall be overwritten; if multiple patches are applied to the same
96907 file, the **.orig** file shall be written only for the first patch. When the *-o outfile* option
96908 is also specified, *file.orig* shall not be created but, if *outfile* already exists, *outfile.orig*
96909 shall be created.96910 **-c** Interpret the patch file as a copied context difference (the output of the utility *diff*
96911 when the *-c* or *-C* options are specified).96912 **-d *dir*** Change the current directory to *dir* before processing as described in the
96913 EXTENDED DESCRIPTION section.96914 **-D *define*** Mark changes with one of the following C preprocessor constructs:96915 #ifdef *define*
96916 ...
96917 #endif

96918 #ifndef *define*
96919 ...
96920 #endif96921 optionally combined with the C preprocessor construct **#else**. If the patched file is
96922 processed with the C preprocessor, where the macro *define* is defined, the output
96923 shall contain the changes from the patch file; otherwise, the output shall not
96924 contain the patches specified in the patch file.96925 **-e** Interpret the patch file as an *ed* script, rather than a *diff* script.96926 **-i *patchfile*** Read the patch information from the file named by the pathname *patchfile*, rather
96927 than the standard input.96928 **-l** (The letter ell.) Cause any sequence of <blank>s in the difference script to match
96929 any sequence of <blank>s in the input file. Other characters shall be matched
96930 exactly.

- 96931 **-n** Interpret the script as a normal difference.
- 96932 **-N** Ignore patches where the differences have already been applied to the file; by
96933 default, already-applied patches shall be rejected.
- 96934 **-o** *outfile* Instead of modifying the files (specified by the *file* operand or the difference
96935 listings) directly, write a copy of the file referenced by each patch, with the
96936 appropriate differences applied, to *outfile*. Multiple patches for a single file shall be
96937 applied to the intermediate versions of the file created by any previous patches,
96938 and shall result in multiple, concatenated versions of the file being written to
96939 *outfile*.
- 96940 **-p** *num* For all pathnames in the patch file that indicate the names of files to be patched,
96941 delete *num* pathname components from the beginning of each pathname. If the
96942 pathname in the patch file is absolute, any leading slashes shall be considered the
96943 first component (that is, **-p 1** shall remove the leading slashes). Specifying **-p 0**
96944 shall cause the full pathname to be used. If **-p** is not specified, only the basename
96945 (the final pathname component) shall be used.
- 96946 **-R** Reverse the sense of the patch script; that is, assume that the difference script was
96947 created from the new version to the old version. The **-R** option cannot be used
96948 with *ed* scripts. The *patch* utility shall attempt to reverse each portion of the script
96949 before applying it. Rejected differences shall be saved in swapped format. If this
96950 option is not specified, and until a portion of the patch file is successfully applied,
96951 *patch* attempts to apply each portion in its reversed sense as well as in its normal
96952 sense. If the attempt is successful, the user shall be prompted to determine whether
96953 the **-R** option should be set.
- 96954 **-r** *rejectfile* Override the default reject filename. In the default case, the reject file shall have the
96955 same name as the output file, with the suffix **.rej** appended to it; see [Patch](#)
96956 [Application](#) (on page 2916).
- 96957 **-u** Interpret the patch file as a unified context difference (the output of the *diff* utility
96958 when the **-u** or **-U** options are specified).

OPERANDS

96959 The following operand shall be supported:

96960 *file* A pathname of a file to patch.

STDIN

96961 See the INPUT FILES section.

INPUT FILES

96962 Input files shall be text files.

ENVIRONMENT VARIABLES

96963 The following environment variables shall affect the execution of *patch*:

96964 **LANG** Provide a default value for the internationalization variables that are unset or null. |
96965 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables |
96966 used to determine the values of locale categories.)

96967 **LC_ALL** If set to a non-empty string value, override the values of all the other
96968 internationalization variables.

96969 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
96970 characters (for example, single-byte as opposed to multi-byte characters in
96971 arguments and input files).

96976		LC_MESSAGES	
96977			Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.
96978			
96979			
96980	XSI	NLSPATH	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
96981		LC_TIME	Determine the locale for recognizing the format of file timestamps written by the <i>diff</i> utility in a context-difference input file.
96982			
96983		ASYNCHRONOUS EVENTS	
96984			Default.
96985		STDOUT	
96986			Not used.
96987		STDERR	
96988			The standard error shall be used for diagnostic and informational messages.
96989		OUTPUT FILES	
96990			The output of the <i>patch</i> utility, the save files (.orig suffixes), and the reject files (.rej suffixes) shall be text files.
96991			
96992		EXTENDED DESCRIPTION	
96993			A patch file may contain patching instructions for more than one file; filenames shall be determined as specified in Filename Determination (on page 2916). When the -b option is specified, for each patched file, the original shall be saved in a file of the same name with the suffix .orig appended to it.
96994			
96995			
96996			
96997			For each patched file, a reject file may also be created as noted in Patch Application (on page 2916). In the absence of a -r option, the name of this file shall be formed by appending the suffix .rej to the original filename.
96998			
96999			
97000		Patch File Format	
97001			The patch file shall contain zero or more lines of header information followed by one or more patches. Each patch shall contain zero or more lines of filename identification in the format produced by the -c , -C , -u , or -U options of the <i>diff</i> utility, and one or more sets of <i>diff</i> output, which are customarily called <i>hunks</i> .
97002			
97003			
97004			
97005			The <i>patch</i> utility shall recognize the following expression in the header information:
97006		Index: <i>pathname</i>	
97007			The file to be patched is named <i>pathname</i> .
97008			If all lines (including headers) within a patch begin with the same leading sequence of <blank>s, the <i>patch</i> utility shall remove this sequence before proceeding. Within each patch, if the type of difference is common context, the <i>patch</i> utility shall recognize the following expressions:
97009			
97010			
97011		*** <i>filename timestamp</i>	
97012			The patches arose from <i>filename</i> .
97013		--- <i>filename timestamp</i>	
97014			The patches should be applied to <i>filename</i> .
97015			If the type of difference is unified context, the <i>patch</i> utility shall recognize the following expressions:
97016			
97017		--- <i>filename timestamp</i>	
97018			The patches arose from <i>filename</i> .

97019 +++ *filename timestamp*
 97020 The patches should be applied to *filename*.

97021 Each hunk within a patch shall be the *diff* output to change a line range within the original file.
 97022 The line numbers for successive hunks within a patch shall occur in ascending order.

97023 Filename Determination

97024 If no *file* operand is specified, *patch* shall perform the following steps to determine the filename
 97025 to use:

- 97026 1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as
 97027 specified by the `-p` option) from the filename on the line beginning with "****" (if copied
 97028 context) or "----" (if unified context), then test for the existence of this file relative to the
 97029 current directory (or the directory specified with the `-d` option). If the file exists, the *patch*
 97030 utility shall use this filename.
- 97031 2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as
 97032 specified by the `-p` option) from the filename on the line beginning with "----" (if copied
 97033 context) or "+++ " (if unified context), then test for the existence of this file relative to the
 97034 current directory (or the directory specified with the `-d` option). If the file exists, the *patch*
 97035 utility shall use this filename.
- 97036 3. If the header information contains a line beginning with the string **Index:**, the *patch* utility
 97037 shall delete pathname components (as specified by the `-p` option) from this line, then test
 97038 for the existence of this file relative to the current directory (or the directory specified
 97039 with the `-d` option). If the file exists, the *patch* utility shall use this filename.
- 97040 4. If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a `get -e`
 97041 **SCCS/s.filename** command to retrieve an editable version of the file. If the file exists, the
 97042 *patch* utility shall use this filename.
- 97043 5. The *patch* utility shall write a prompt to standard output and request a filename
 97044 interactively from the controlling terminal (for example, `/dev/tty`).

97045 Patch Application

97046 If the `-c`, `-e`, `-n`, or `-u` option is present, the *patch* utility shall interpret information within each
 97047 hunk as a copied context difference, an *ed* difference, a normal difference, or a unified context
 97048 difference, respectively. In the absence of any of these options, the *patch* utility shall determine
 97049 the type of difference based on the format of information within the hunk.

97050 For each hunk, the *patch* utility shall begin to search for the place to apply the patch at the line
 97051 number at the beginning of the hunk, plus or minus any offset used in applying the previous
 97052 hunk. If lines matching the hunk context are not found, *patch* shall scan both forwards and
 97053 backwards at least 1 000 bytes for a set of lines that match the hunk context.

97054 If no such place is found and it is a context difference, then another scan shall take place,
 97055 ignoring the first and last line of context. If that fails, the first two and last two lines of context
 97056 shall be ignored and another scan shall be made. Implementations may search more extensively
 97057 for installation locations.

97058 If no location can be found, the *patch* utility shall append the hunk to the reject file. A rejected
 97059 hunk that is a copied context difference, an *ed* difference, or a normal difference shall be written
 97060 in copied-context-difference format regardless of the format of the patch file. It is
 97061 implementation-defined whether a rejected hunk that is a unified context difference is written in
 97062 copied-context-difference format or in unified-context-difference format. If the input was a
 97063 normal or *ed*-style difference, the reject file may contain differences with zero lines of context.
 97064 The line numbers on the hunks in the reject file may be different from the line numbers in the
 97065 patch file since they shall reflect the approximate locations for the failed hunks in the new file

97066 rather than the old one.

97067 If the type of patch is an *ed* diff, the implementation may accomplish the patching by invoking
97068 the *ed* utility.

97069 EXIT STATUS

97070 The following exit values shall be returned:

- 97071 0 Successful completion.
- 97072 1 One or more lines were written to a reject file.
- 97073 >1 An error occurred.

97074 CONSEQUENCES OF ERRORS

97075 Patches that cannot be correctly placed in the file shall be written to a reject file.

97076 APPLICATION USAGE

97077 The **-R** option does not work with *ed* scripts because there is too little information to reconstruct
97078 the reverse operation.

97079 The **-p** option makes it possible to customize a patch file to local user directory structures
97080 without manually editing the patch file. For example, if the filename in the patch file was:

97081 `/curds/whey/src/blurfl/blurfl.c`

97082 Setting **-p 0** gives the entire pathname unmodified; **-p 1** gives:

97083 `curds/whey/src/blurfl/blurfl.c`

97084 without the leading slash, **-p 4** gives:

97085 `blurfl/blurfl.c`

97086 and not specifying **-p** at all gives:

97087 `blurfl.c` .

97088 EXAMPLES

97089 None.

97090 RATIONALE

97091 Some of the functionality in historical *patch* implementations was not specified. The following
97092 documents those features present in historical implementations that have not been specified.

97093 A deleted piece of functionality was the '+' pseudo-option allowing an additional set of options
97094 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

97095 In historical implementations, if the string "Prereq:" appeared in the header, the *patch* utility
97096 would search for the corresponding version information (the string specified in the header,
97097 delimited by <blank>s or the beginning or end of a line or the file) anywhere in the original file.
97098 This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize.
97099 For example, if:

97100 `Prereq: 1.2`

97101 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the
97102 prerequisite.

97103 The following options were dropped from historical implementations of *patch* as insufficiently
97104 useful to standardize:

- 97105 **-b** The **-b** option historically provided a method for changing the name extension of
97106 the backup file from the default **.orig**. This option has been modified and retained
97107 in this volume of POSIX.1-200x.

- 97108 **-F** The **-F** option specified the number of lines of a context diff to ignore when
97109 searching for a place to install a patch.
- 97110 **-f** The **-f** option historically caused *patch* not to request additional information from
97111 the user.
- 97112 **-r** The **-r** option historically provided a method of overriding the extension of the
97113 reject file from the default **.rej**.
- 97114 **-s** The **-s** option historically caused *patch* to work silently unless an error occurred.
- 97115 **-x** The **-x** option historically set internal debugging flags.

97116 In some file system implementations, the saving of a **.orig** file may produce unwanted results. In
97117 the case of 12, 13, or 14-character filenames (on file systems supporting 14-character maximum
97118 filenames), the **.orig** file overwrites the new file. The reject file may also exceed this filename
97119 limit. It was suggested, due to some historical practice, that a tilde ('~') suffix be used instead
97120 of **.orig** and some other character instead of the **.rej** suffix. This was rejected because it is not
97121 obvious to the user which file is which. The suffixes **.orig** and **.rej** are clearer and more
97122 understandable.

97123 The **-b** option has the opposite sense in some historical implementations—do not save the **.orig**
97124 file. The default case here is not to save the files, making *patch* behave more consistently with the
97125 other standard utilities.

97126 The **-w** option in early proposals was changed to **-l** to match historical practice.

97127 The **-N** option was included because without it, a non-interactive application cannot reject
97128 previously applied patches. For example, if a user is piping the output of *diff* into the *patch*
97129 utility, and the user only wants to patch a file to a newer version non-interactively, the **-N** option
97130 is required.

97131 Changes to the **-l** option description were proposed to allow matching across <newline>s in
97132 addition to just <blank>s. Since this is not historical practice, and since some ambiguities could
97133 result, it is suggested that future developments in this area utilize another option letter, such as
97134 **-L**.

97135 The **-u** option of GNU *patch* has been added, along with support for unified context formats.

97136 **FUTURE DIRECTIONS**

97137 None.

97138 **SEE ALSO**

97139 [diff, ed](#)

97140 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

97141 **CHANGE HISTORY**

97142 First released in Issue 4.

97143 **Issue 5**

97144 The FUTURE DIRECTIONS section is added.

97145 **Issue 6**

97146 This utility is marked as part of the User Portability Utilities option.

97147 The description of the **-D** option and the steps in [Filename Determination](#) (on page 2916) are
97148 changed to match historical practice as defined in the IEEE P1003.2b draft standard.

97149 The normative text is reworded to avoid use of the term “must” for application requirements.

97150 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/34 is applied, clarifying the way that the
97151 *patch* utility performs **ifdef** selection for the **-D** option.

97152

Issue 7

97153

The *patch* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

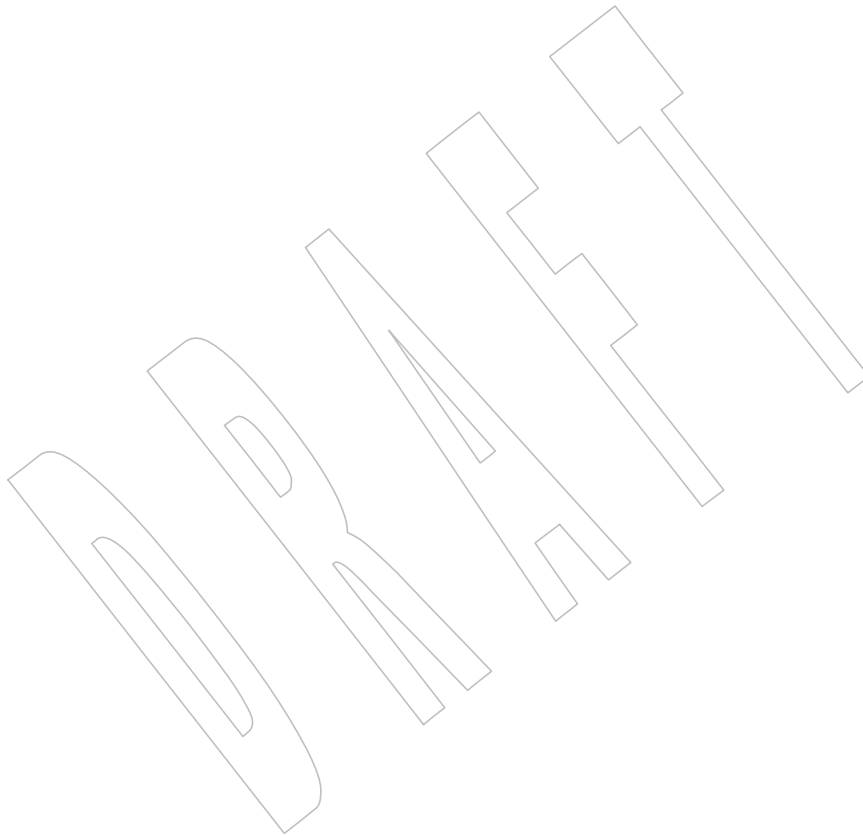
97154

97155

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

97156

SD5-XCU-ERN-103 and SD5-XCU-ERN-120 are applied, adding the `-u` option.



97157 **NAME**

97158 pathchk — check pathnames

97159 **SYNOPSIS**97160 pathchk [-p] [-P] *pathname*...97161 **DESCRIPTION**97162 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used
97163 to access or create a file without causing syntax errors) and portable (that is, no filename
97164 truncation results). More extensive portability checks are provided by the **-p** and **-P** options.97165 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the
97166 underlying file system. A diagnostic shall be written for each *pathname* operand that:

- 97167
- Is longer than {PATH_MAX} bytes (see **Pathname Variable Values** in XBD [Chapter 13](#) (on
97168 page 205), **<limits.h>**)
 - Contains any component longer than {NAME_MAX} bytes in its containing directory
 - Contains any component in a directory that is not searchable
 - Contains any character in any component that is not valid in its containing directory

97172 The format of the diagnostic message is not specified, but shall indicate the error detected and
97173 the corresponding *pathname* operand.97174 It shall not be considered an error if one or more components of a *pathname* operand do not exist
97175 as long as a file matching the pathname specified by the missing components could be created
97176 that does not violate any of the checks specified above.97177 **OPTIONS**97178 The *pathchk* utility shall conform to XBD [Section 12.2](#) (on page 201).

97179 The following option shall be supported:

97180 **-p** Instead of performing checks based on the underlying file system, write a
97181 diagnostic for each *pathname* operand that:

- 97182
- Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in XBD
97183 [Chapter 13](#) (on page 205), **<limits.h>**)
 - Contains any component longer than {_POSIX_NAME_MAX} bytes
 - Contains any character in any component that is not in the portable filename
97186 character set

97187 **-P** Write a diagnostic for each *pathname* operand that:

- 97188
- Contains a component whose first character is the hyphen character
 - Is empty

97190 **OPERANDS**

97191 The following operand shall be supported:

97192 *pathname* A pathname to be checked.97193 **STDIN**

97194 Not used.

97195 **INPUT FILES**

97196 None.

97197 **ENVIRONMENT VARIABLES**97198 The following environment variables shall affect the execution of *pathchk*:

97199 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 97200 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 97201 used to determine the values of locale categories.)

97202 *LC_ALL* If set to a non-empty string value, override the values of all the other
 97203 internationalization variables.

97204 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 97205 characters (for example, single-byte as opposed to multi-byte characters in
 97206 arguments).

97207 *LC_MESSAGES*

97208 Determine the locale that should be used to affect the format and contents of
 97209 diagnostic messages written to standard error.

97210 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

97211 **ASYNCHRONOUS EVENTS**

97212 Default.

97213 **STDOUT**

97214 Not used.

97215 **STDERR**

97216 The standard error shall be used only for diagnostic messages.

97217 **OUTPUT FILES**

97218 None.

97219 **EXTENDED DESCRIPTION**

97220 None.

97221 **EXIT STATUS**

97222 The following exit values shall be returned:

97223 0 All *pathname* operands passed all of the checks.

97224 >0 An error occurred.

97225 **CONSEQUENCES OF ERRORS**

97226 Default.

97227 **APPLICATION USAGE**

97228 The *test* utility can be used to determine whether a given pathname names an existing file; it
 97229 does not, however, give any indication of whether or not any component of the pathname was
 97230 truncated in a directory where the *_POSIX_NO_TRUNC* feature is not in effect. The *pathchk*
 97231 utility does not check for file existence; it performs checks to determine whether a pathname
 97232 does exist or could be created with no pathname component truncation.

97233 The *noclobber* option in the shell (see the *set* special built-in) can be used to atomically create a
 97234 file. As with all file creation semantics in the System Interfaces volume of POSIX.1-200x, it
 97235 guarantees atomic creation, but still depends on applications to agree on conventions and
 97236 cooperate on the use of files after they have been created.

97237 To verify that a pathname meets the requirements of filename portability, applications should
 97238 use both the *-p* and *-P* options together.

EXAMPLES

97239
97240
97241
97242
97243
97244
97245
97246
97247
97248
97249
97250
97251
97252
97253
97254
97255
97256
97257
97258
97259
97260
97261
97262
97263
97264
97265
97266
97267
97268
97269
97270
97271
97272
97273
97274
97275
97276
97277
97278
97279
97280
97281
97282
97283
97284
97285
97286
97287

To verify that all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system:

```
# This example assumes that no pathnames in the archive
# contain <newline> characters.
pax -f archive | sed -e 's/^[^[:alnum:]]/\\&/g' | xargs pathchk --
if [ $? -eq 0 ]
then
    pax -r -f archive
else
    echo Investigate problems before importing files.
    exit 1
fi
```

To verify that all files in the current directory hierarchy could be moved to any system conforming to the System Interfaces volume of POSIX.1-200x that also supports the *pax* utility:

```
find . -exec pathchk -p -P {} +
if [ $? -eq 0 ]
then
    pax -w -f ../archive .
else
    echo Portable archive cannot be created.
    exit 1
fi
```

To verify that a user-supplied pathname names a readable file and that the application can create a file extending the given path without truncation and without overwriting any existing file:

```
case $- in
    *C*)   reset="";;
    *)    reset="set +C"
          set -C;;
esac
test -r "$path" && pathchk "$path.out" &&
rm "$path.out" > "$path.out"
if [ $? -ne 0 ]; then
    printf "%s: %s not found or %s.out fails \
creation checks.\n" $0 "$path" "$path"
    $reset # Reset the noclobber option in case a trap
          # on EXIT depends on it.
    exit 1
fi
$reset
PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

1. **PROCESSING** represents the code that is used by the application to use **\$path** once it is verified that **\$path.out** works as intended.
2. The state of the *noclobber* option is unknown when this code is invoked and should be set on exit to the state it was in when this code was invoked. (The **reset** variable is used in this example to restore the initial state.)
3. Note the usage of:

```
rm "$path.out" > "$path.out"
```

- 97288 a. The *pathchk* command has already verified, at this point, that **\$path.out** is not
97289 truncated.
- 97290 b. With the *noclobber* option set, the shell verifies that **\$path.out** does not already exist
97291 before invoking *rm*.
- 97292 c. If the shell succeeded in creating **\$path.out**, *rm* removes it so that the application
97293 can create the file again in the **PROCESSING** step.
- 97294 d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:
97295 `rm "$path.out" > "$path.out"`
97296 should be replaced with:
97297 `> "$path.out"`
97298 which verifies that the file did not already exist, but leaves **\$path.out** in place for
97299 use by **PROCESSING**.

RATIONALE

97300 The *pathchk* utility was new for the ISO POSIX-2:1993 standard. It, along with the *set*
97301 `-C(noclobber)` option added to the shell, replaces the *mktmp*, *validfnam*, and *create* utilities that
97302 appeared in early proposals. All of these utilities were attempts to solve several common
97303 problems:
97304

- 97305 • Verify the validity (for several different definitions of “valid”) of a pathname supplied by a
97306 user, generated by an application, or imported from an external source.
- 97307 • Atomically create a file.
- 97308 • Perform various string handling functions to generate a temporary filename.

97309 The *create* utility, included in an early proposal, provided checking and atomic creation in a
97310 single invocation of the utility; these are orthogonal issues and need not be grouped into a single
97311 utility. Note that the *noclobber* option also provides a way of creating a lock for process
97312 synchronization; since it provides an atomic *create*, there is no race between a test for existence
97313 and the following creation if it did not exist.

97314 Having a function like *tmpnam()* in the ISO C standard is important in many high-level
97315 languages. The shell programming language, however, has built-in string manipulation
97316 facilities, making it very easy to construct temporary filenames. The names needed obviously
97317 depend on the application, but are frequently of a form similar to:

97318 `$TMPDIR/application_abbreviation$$suffix`

97319 In cases where there is likely to be contention for a given suffix, a simple shell **for** or **while** loop
97320 can be used with the shell *noclobber* option to create a file without risk of collisions, as long as
97321 applications trying to use the same filename name space are cooperating on the use of files after
97322 they have been created.

97323 For historical purposes, `-p` does not check for the use of the hyphen character as the first
97324 character in a component of the pathname, or for an empty *pathname* operand.

FUTURE DIRECTIONS

97325 None.
97326

SEE ALSO

97327 [Section 2.7](#) (on page 2259), *set* (on page 2302), *test*
97328

97329 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), `<limits.h>` +

97330

CHANGE HISTORY

97331

First released in Issue 4.

97332

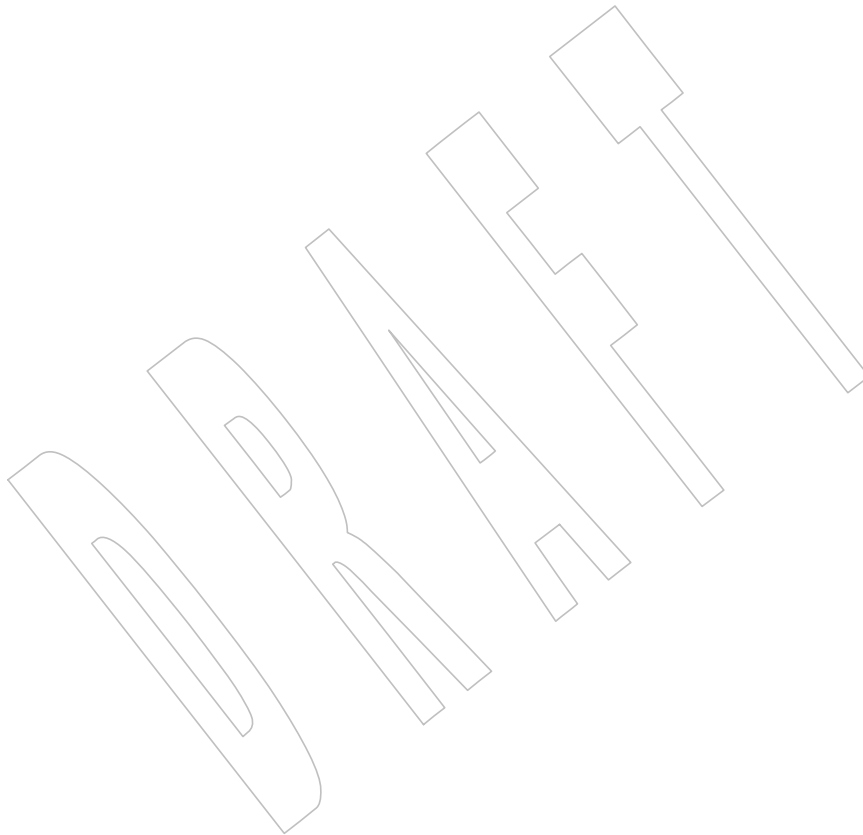
Issue 7

97333

Austin Group Interpretations 1003.1-2001 #039, #040, and #094 are applied.

97334

SD5-XCU-ERN-121 is applied, updating the EXAMPLES section.



97335 **NAME**
 97336 pax — portable archive interchange

97337 **SYNOPSIS**

97338 pax [-dv] [-c|-n] [-H|-L] [-o options] [-f archive] [-s replstr]...
 97339 [pattern...]

97340 pax -r[-c|-n] [-dikuv] [-H|-L] [-f archive] [-o options]... [-p string]...
 97341 [-s replstr]... [pattern...]

97342 pax -w [-dituvX] [-H|-L] [-b blocksize] [[-a] [-f archive]] [-o options]... |
 97343 [-s replstr]... [-x format] [file...]

97344 pax -r -w [-diklntuvX] [-H|-L] [-o options]... [-p string]... |
 97345 [-s replstr]... [file...] directory

97346 **DESCRIPTION**

97347 The *pax* utility shall read, write, and write lists of the members of archive files and copy
 97348 directory hierarchies. A variety of archive formats shall be supported; see the *-x format* option.

97349 The action to be taken depends on the presence of the *-r* and *-w* options. The four combinations
 97350 of *-r* and *-w* are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes,
 97351 corresponding respectively to the four forms shown in the SYNOPSIS section.

97352 **list** In **list** mode (when neither *-r* nor *-w* are specified), *pax* shall write the names of
 97353 the members of the archive file read from the standard input, with pathnames
 97354 matching the specified patterns, to standard output. If a named file is of type
 97355 directory, the file hierarchy rooted at that file shall be listed as well.

97356 **read** In **read** mode (when *-r* is specified, but *-w* is not), *pax* shall extract the members of
 97357 the archive file read from the standard input, with pathnames matching the
 97358 specified patterns. If an extracted file is of type directory, the file hierarchy rooted
 97359 at that file shall be extracted as well. The extracted files shall be created performing
 97360 pathname resolution with the directory in which *pax* was invoked as the current
 97361 working directory.

97362 If an attempt is made to extract a directory when the directory already exists, this
 97363 shall not be considered an error. If an attempt is made to extract a FIFO when the
 97364 FIFO already exists, this shall not be considered an error.

97365 The ownership, access, and modification times, and file mode of the restored files
 97366 are discussed under the *-p* option.

97367 **write** In **write** mode (when *-w* is specified, but *-r* is not), *pax* shall write the contents of
 97368 the *file* operands to the standard output in an archive format. If no *file* operands are
 97369 specified, a list of files to copy, one per line, shall be read from the standard input
 97370 and each entry in this list shall be processed as if it had been a *file* operand on the
 97371 command line. A file of type directory shall include all of the files in the file
 97372 hierarchy rooted at the file.

97373 **copy** In **copy** mode (when both *-r* and *-w* are specified), *pax* shall copy the *file* operands
 97374 to the destination directory.

97375 If no *file* operands are specified, a list of files to copy, one per line, shall be read
 97376 from the standard input. A file of type directory shall include all of the files in the
 97377 file hierarchy rooted at the file.

97378 The effect of the **copy** shall be as if the copied files were written to a *pax* format
 97379 archive file and then subsequently extracted, except that there may be hard links

between the original and the copied files. If the destination directory is a subdirectory of one of the files to be copied, the results are unspecified. If the destination directory is a file of a type not defined by the System Interfaces volume of POSIX.1-200x, the results are implementation-defined; otherwise, it shall be an error for the file named by the *directory* operand not to exist, not be writable by the user, or not be a file of type directory.

In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member, *pax* shall perform actions equivalent to the *mkdir()* function defined in the System Interfaces volume of POSIX.1-200x, called with the following arguments:

- The intermediate directory used as the *path* argument
- The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument

If any specified *pattern* or *file* operands are not matched by at least one file or archive member, *pax* shall write a diagnostic message to standard error for each one that did not match and exit with a non-zero exit status.

The archive formats described in the EXTENDED DESCRIPTION section shall be automatically detected on input. The default output archive format shall be implementation-defined.

A single archive can span multiple files. The *pax* utility shall determine, in an implementation-defined manner, what file to read or write as the next file.

If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted, except that if the files to be linked are symbolic links and the system is not capable of making hard links to symbolic links, then separate copies of the symbolic link shall be created instead. For archive formats that do not store file contents with each name that causes a hard link, if the file that contains the data is not extracted during this *pax* session, either the data shall be restored from the original file, or a diagnostic message shall be displayed with the name of a file that can be used to extract the data. In traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall write a diagnostic message to standard error and shall terminate.

OPTIONS

The *pax* utility shall conform to XBD Section 12.2 (on page 201), except that the order of presentation of the **-o**, **-p**, and **-s** options is significant.

The following options shall be supported:

- r** Read an archive file from standard input.
- w** Write files to the standard output in the specified archive format.
- a** Append files to the end of the archive. It is implementation-defined which devices on the system support appending. Additional file formats unspecified by this volume of POSIX.1-200x may impose restrictions on appending.
- b *blocksize*** Block the output at a positive decimal integer number of bytes per write to the archive file. Devices and archive formats may impose restrictions on blocking. Blocking shall be automatically determined on input. Conforming applications shall not specify a *blocksize* value larger than 32256. Default blocking when creating archives depends on the archive format. (See the **-x** option below.)
- c** Match all file or archive members except those specified by the *pattern* or *file* operands.

- 97425 **-d** Cause files of type directory being copied or archived or archive members of type
97426 directory being extracted or listed to match only the file or archive member itself
97427 and not the file hierarchy rooted at the file.
- 97428 **-f** *archive* Specify the pathname of the input or output archive, overriding the default
97429 standard input (in **list** or **read** modes) or standard output (**write** mode).
- 97430 **-H** If a symbolic link referencing a file of type directory is specified on the command
97431 line, *pax* shall archive the file hierarchy rooted in the file referenced by the link,
97432 using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic
97433 link referencing a file of any other file type which *pax* can normally archive is
97434 specified on the command line, then *pax* shall archive the file referenced by the
97435 link, using the name of the link. The default behavior, when neither **-H** or **-L** are
97436 specified, shall be to archive the symbolic link itself.
- 97437 **-i** Interactively rename files or archive members. For each archive member matching
97438 a *pattern* operand or file matching a *file* operand, a prompt shall be written to the
97439 file **/dev/tty**. The prompt shall contain the name of the file or archive member, but
97440 the format is otherwise unspecified. A line shall then be read from **/dev/tty**. If this
97441 line is blank, the file or archive member shall be skipped. If this line consists of a
97442 single period, the file or archive member shall be processed with no modification
97443 to its name. Otherwise, its name shall be replaced with the contents of the line. The
97444 *pax* utility shall immediately exit with a non-zero exit status if end-of-file is
97445 encountered when reading a response or if **/dev/tty** cannot be opened for reading
97446 and writing.
- 97447 The results of extracting a hard link to a file that has been renamed during
97448 extraction are unspecified.
- 97449 **-k** Prevent the overwriting of existing files.
- 97450 **-l** (The letter ell.) In **copy** mode, hard links shall be made between the source and
97451 destination file hierarchies whenever possible. If specified in conjunction with **-H**
97452 or **-L**, when a symbolic link is encountered, the hard link created in the destination
97453 file hierarchy shall be to the file referenced by the symbolic link. If specified when
97454 neither **-H** nor **-L** is specified, when a symbolic link is encountered, the
97455 implementation shall create a hard link to the symbolic link in the source file
97456 hierarchy or copy the symbolic link to the destination.
- 97457 **-L** If a symbolic link referencing a file of type directory is specified on the command
97458 line or encountered during the traversal of a file hierarchy, *pax* shall archive the file
97459 hierarchy rooted in the file referenced by the link, using the name of the link as the
97460 root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any
97461 other file type which *pax* can normally archive is specified on the command line or
97462 encountered during the traversal of a file hierarchy, *pax* shall archive the file
97463 referenced by the link, using the name of the link. The default behavior, when
97464 neither **-H** or **-L** are specified, shall be to archive the symbolic link itself.
- 97465 **-n** Select the first archive member that matches each *pattern* operand. No more than
97466 one archive member shall be matched for each pattern (although members of type
97467 directory shall still match the file hierarchy rooted at that file).
- 97468 **-o** *options* Provide information to the implementation to modify the algorithm for extracting
97469 or writing files. The value of *options* shall consist of one or more comma-separated
97470 keywords of the form:
97471 *keyword*[[:]=*value*][,*keyword*[[:]=*value*], ...]
- 97472 Some keywords apply only to certain file formats, as indicated with each
97473 description. Use of keywords that are inapplicable to the file format being

processed produces undefined results.

Keywords in the *options* argument shall be a string that would be a valid portable filename as described in XBD [Section 3.275](#) (on page 71).

Note: Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames.

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of *value*. A comma as the final character, or a comma followed solely by white space as the final characters, in *options* shall be ignored. Multiple `-o` options can be specified; if keywords given to these multiple `-o` options conflict, the keywords and values appearing later in command line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

delete=pattern

(Applicable only to the `-x pax` format.) When used in **write** or **copy** mode, *pax* shall omit from extended header records that it produces any keywords matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore any keywords matching the string pattern in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in [Section 2.13.1](#) (on page 2278) and [Section 2.13.2](#) (on page 2279). For example:

```
-o delete=security.*
```

would suppress security-related information. See [pax Extended Header](#) (on page 2938) for extended header record keyword usage.

When multiple `-odelete=pattern` options are specified, the patterns shall be additive; all keywords matching the specified string patterns shall be omitted from extended header records that *pax* produces.

exthdr.name=string

(Applicable only to the `-x pax` format.) This keyword allows user control over the name that is written into the **ustar** header blocks for the extended header produced under the circumstances described in [pax Header Block](#) (on page 2937). The name shall be the contents of *string*, after the following character substitutions have been made:

<i>string</i>	
Includes:	Replaced By:
%d	The directory name of the file, equivalent to the result of the <i>dirname</i> utility on the translated pathname.
%f	The filename of the file, equivalent to the result of the <i>basename</i> utility on the translated pathname.
%p	The process ID of the <i>pax</i> process.
%%	A '%' character.

Any other '%' characters in *string* produce undefined results.

If no `-o exthdr.name=string` is specified, *pax* shall use the following default value:

```
%d/PaxHeaders.%p/%f
```


globexthdr.name=string

(Applicable only to the **-x pax** format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records with **ustar** header blocks that will be treated as regular files by previous versions of *pax*. This keyword allows user control over the name that is written into the **ustar** header blocks for global extended header records. The name shall be the contents of *string*, after the following character substitutions have been made:

<i>string</i> Includes:	Replaced By:
%n	An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
%p	The process ID of the <i>pax</i> process.
%%	A '%' character.

Any other '%' characters in *string* produce undefined results.

If no **-o globexthdr.name=string** is specified, *pax* shall use the following default value:

```
$TMPDIR/GlobalHead.%p.%n
```

where *\$TMPDIR* represents the value of the *TMPDIR* environment variable. If *TMPDIR* is not set, *pax* shall use **/tmp**.

invalid=action

(Applicable only to the **-x pax** format.) This keyword allows user control over the action *pax* takes upon encountering values in an extended header record that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list** mode, cannot be written in the codeset and current locale of the implementation. The following are invalid values that shall be recognized by *pax*:

- In **read** or **copy** mode, a filename or link name that contains character encodings invalid in the destination hierarchy. (For example, the name may contain embedded NULs.)
- In **read** or **copy** mode, a filename or link name that is longer than the maximum allowed in the destination hierarchy (for either a pathname component or the entire pathname).
- In **list** mode, any character string value (filename, link name, user name, and so on) that cannot be written in the codeset and current locale of the implementation.

The following mutually-exclusive values of the *action* argument are supported:

binary In **write** mode, *pax* shall generate a **hdrcharset=BINARY** extended header record for each file with a filename, link name, group name, owner name, or any other field in an extended header record that cannot be translated to the UTF-8 codeset, allowing the archive to contain the files with unencoded extended header record values. In **read** or **copy** mode, *pax* shall use the values specified in the header without translation, regardless of whether this may overwrite an existing file with a valid name. In **list** mode, *pax* shall behave identically to the **bypass** action.

97567 97568 97569 97570	bypass	In read or copy mode, <i>pax</i> shall bypass the file, causing no change to the destination hierarchy. In list mode, <i>pax</i> shall write all requested valid values for the file, but its method for writing invalid values is unspecified.
97571 97572 97573 97574	rename	In read or copy mode, <i>pax</i> shall act as if the -i option were in effect for each file with invalid filename or link name values, allowing the user to provide a replacement name interactively. In list mode, <i>pax</i> shall behave identically to the bypass action.
97575 97576 97577 97578 97579 97580 97581 97582 97583	UTF-8	When used in read , copy , or list mode and a filename, link name, owner name, or any other field in an extended header record cannot be translated from the pax UTF-8 codeset format to the codeset and current locale of the implementation, <i>pax</i> shall use the actual UTF-8 encoding for the name. If a hdrcharset extended header record is in effect for this file, the character set specified by that record shall be used instead of UTF-8. If a hdrcharset=BINARY extended header record is in effect for this file, no translation shall be performed.
97584 97585 97586 97587	write	In read or copy mode, <i>pax</i> shall write the file, translating the name, regardless of whether this may overwrite an existing file with a valid name. In list mode, <i>pax</i> shall behave identically to the bypass action.
97588 97589 97590 97591		If no -o invalid=option is specified, <i>pax</i> shall act as if -o invalid=bypass were specified. Any overwriting of existing files that may be allowed by the -o invalid= actions shall be subject to permission (-p) and modification time (-u) restrictions, and shall be suppressed if the -k option is also specified.
97592 97593 97594 97595	linkdata	(Applicable only to the -x pax format.) In write mode, <i>pax</i> shall write the contents of a file to the archive even when that file is merely a hard link to a file whose contents have already been written to the archive.
97596 97597 97598 97599 97600 97601 97602 97603 97604	listopt=format	This keyword specifies the output format of the table of contents produced when the -v option is specified in list mode. See List Mode Format Specifications (on page 2933). To avoid ambiguity, the listopt=format shall be the only or final keyword=value pair in a -o option-argument; all characters in the remainder of the option-argument shall be considered part of the format string. When multiple -olistopt=format options are specified, the format strings shall be considered a single, concatenated string, evaluated in command line order.
97605 97606 97607 97608	times	(Applicable only to the -x pax format.) When used in write or copy mode, <i>pax</i> shall include atime and mtime extended header records for each file. See pax Extended Header File Times (on page 2941).
97609 97610 97611 97612		In addition to these keywords, if the -x pax format is specified, any of the keywords and values defined in pax Extended Header (on page 2938), including implementation extensions, can be used in -o option-arguments, in either of two modes:
97613 97614 97615 97616	keyword=value	When used in write or copy mode, these keyword/value pairs shall be included at the beginning of the archive as typeflag g global extended header records. When used in read or list mode, these keyword/value pairs shall act

97617 as if they had been at the beginning of the archive as **typeflag g** global
97618 extended header records.

97619 **keyword:=value**

97620 When used in **write** or **copy** mode, these keyword/value pairs shall be
97621 included as records at the beginning of a **typeflag x** extended header for each
97622 file. (This shall be equivalent to the equal-sign form except that it creates no
97623 **typeflag g** global extended header records.) When used in **read** or **list** mode,
97624 these keyword/value pairs shall act as if they were included as records at the
97625 end of each extended header; thus, they shall override any global or file-
97626 specific extended header record keywords of the same names. For example, in
97627 the command:

```
97628 pax -r -o "  
97629 gname:=mygroup,  
97630 " <archive
```

97631 the group name will be forced to a new value for all files read from the
97632 archive.

97633 The precedence of **-o** keywords over various fields in the archive is described in
97634 [pax Extended Header Keyword Precedence](#) (on page 2941).

97635 **-p string**

97636 Specify one or more file characteristic options (privileges). The *string* option-
97637 argument shall be a string specifying file characteristics to be retained or discarded
97638 on extraction. The string shall consist of the specification characters a, e, m, o, and
97639 p. Other implementation-defined characters can be included. Multiple
97640 characteristics can be concatenated within the same string and multiple **-p** options
can be specified. The meaning of the specification characters are as follows:

- 97641 a Do not preserve file access times.
- 97642 e Preserve the user ID, group ID, file mode bits (see XBD [Section 3.168](#), on page |
97643 56), access time, modification time, and any other implementation-defined file |
97644 characteristics.
- 97645 m Do not preserve file modification times.
- 97646 o Preserve the user ID and group ID.
- 97647 p Preserve the file mode bits. Other implementation-defined file mode attributes
97648 may be preserved.

97649 In the preceding list, "preserve" indicates that an attribute stored in the archive
97650 shall be given to the extracted file, subject to the permissions of the invoking
97651 process. The access and modification times of the file shall be preserved unless
97652 otherwise specified with the **-p** option or not stored in the archive. All attributes
97653 that are not preserved shall be determined as part of the normal file creation action
97654 (see [Section 1.1.1.4](#), on page 2228).

97655 If neither the e nor the o specification character is specified, or the user ID and
97656 group ID are not preserved for any reason, *pax* shall not set the S_ISUID and
97657 S_ISGID bits of the file mode.

97658 If the preservation of any of these items fails for any reason, *pax* shall write a
97659 diagnostic message to standard error. Failure to preserve these items shall affect
97660 the final exit status, but shall not cause the extracted file to be deleted.

97661 If file characteristic letters in any of the *string* option-arguments are duplicated or
97662 conflict with each other, the ones given last shall take precedence. For example, if
97663 **-p eme** is specified, file modification times are preserved.

97664 -s replstr Modify file or archive member names named by *pattern* or *file* operands according
97665 to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts
97666 of “address” and “line” are meaningless in the context of the *pax* utility, and shall
97667 not be supplied. The format shall be:

97668 -s /old/new/[gp]

97669 where as in *ed*, *old* is a basic regular expression and *new* can contain an ampersand,
97670 ‘\n’ (where *n* is a digit) back-references, or subexpression matching. The *old*
97671 string shall also be permitted to contain <newline>s.

97672 Any non-null character can be used as a delimiter (‘/’ shown here). Multiple -s
97673 expressions can be specified; the expressions shall be applied in the order
97674 specified, terminating with the first successful substitution. The optional trailing
97675 ‘g’ is as defined in the *ed* utility. The optional trailing ‘p’ shall cause successful
97676 substitutions to be written to standard error. File or archive member names that
97677 substitute to the empty string shall be ignored when reading and writing archives.

97678 -t When reading files from the file system, and if the user has the permissions
97679 required by *utime()* to do so, set the access time of each file read to the access time
97680 that it had before being read by *pax*.

97681 -u Ignore files that are older (having a less recent file modification time) than a pre-
97682 existing file or archive member with the same name. In **read** mode, an archive
97683 member with the same name as a file in the file system shall be extracted if the
97684 archive member is newer than the file. In **write** mode, an archive file member with
97685 the same name as a file in the file system shall be superseded if the file is newer
97686 than the archive member. If -a is also specified, this is accomplished by appending
97687 to the archive; otherwise, it is unspecified whether this is accomplished by actual
97688 replacement in the archive or by appending to the archive. In **copy** mode, the file
97689 in the destination hierarchy shall be replaced by the file in the source hierarchy or
97690 by a link to the file in the source hierarchy if the file in the source hierarchy is
97691 newer.

97692 -v In **list** mode, produce a verbose table of contents (see the STDOUT section).
97693 Otherwise, write archive member pathnames to standard error (see the STDERR
97694 section).

97695 -x format Specify the output archive format. The *pax* utility shall support the following
97696 formats:

97697 **cpio** The **cpio** interchange format; see the EXTENDED DESCRIPTION
97698 section. The default *blocksize* for this format for character special
97699 archive files shall be 5120. Implementations shall support all
97700 *blocksize* values less than or equal to 32256 that are multiples of 512.

97701 **pax** The **pax** interchange format; see the EXTENDED DESCRIPTION
97702 section. The default *blocksize* for this format for character special
97703 archive files shall be 5120. Implementations shall support all
97704 *blocksize* values less than or equal to 32256 that are multiples of 512.

97705 **ustar** The **tar** interchange format; see the EXTENDED DESCRIPTION
97706 section. The default *blocksize* for this format for character special
97707 archive files shall be 10240. Implementations shall support all
97708 *blocksize* values less than or equal to 32256 that are multiples of 512.

97709 Implementation-defined formats shall specify a default block size as well as any
97710 other block sizes supported for character special archive files.

97711 Any attempt to append to an archive file in a format different from the existing

97712 archive format shall cause *pax* to exit immediately with a non-zero exit status.

97713 **-X** When traversing the file hierarchy specified by a pathname, *pax* shall not descend
97714 into directories that have a different device ID (*st_dev*; see the System Interfaces
97715 volume of POSIX.1-200x, *stat()*).

97716 Specifying more than one of the mutually-exclusive options **-H** and **-L** shall not be considered
97717 an error and the last option specified shall determine the behavior of the utility.

97718 The options that operate on the names of files or archive members (**-c**, **-i**, **-n**, **-s**, **-u**, and **-v**)
97719 shall interact as follows. In **read** mode, the archive members shall be selected based on the user-
97720 specified *pattern* operands as modified by the **-c**, **-n**, and **-u** options. Then, any **-s** and **-i**
97721 options shall modify, in that order, the names of the selected files. The **-v** option shall write
97722 names resulting from these modifications.

97723 In **write** mode, the files shall be selected based on the user-specified pathnames as modified by
97724 the **-n** and **-u** options. Then, any **-s** and **-i** options shall modify, in that order, the names of
97725 these selected files. The **-v** option shall write names resulting from these modifications.

97726 If both the **-u** and **-n** options are specified, *pax* shall not consider a file selected unless it is
97727 newer than the file to which it is compared.

97728 List Mode Format Specifications

97729 In **list** mode with the **-o listopt=format** option, the *format* argument shall be applied for each
97730 selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file.
97731 The *format* argument shall be used as the *format* string described in XBD Chapter 5 (on page 107),
97732 with the exceptions 1. through 5. defined in the EXTENDED DESCRIPTION section of *printf*,
97733 plus the following exceptions:

- 97734 6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion
97735 argument is defined by the value of *keyword*. The implementation shall support the
97736 following keywords:

- 97737 — Any of the Field Name entries in Table 4-13 (on page 2942) and Table 4-15 (on page
97738 2945). The implementation may support the *cpio* keywords without the leading *c_* in
97739 addition to the form required by Table 4-15 (on page 2945).

- 97740 — Any keyword defined for the extended header in **pax Extended Header** (on page
97741 2938).

- 97742 — Any keyword provided as an implementation-defined extension within the extended
97743 header defined in **pax Extended Header** (on page 2938).

97744 For example, the sequence "%(charset)s" is the string value of the name of the character
97745 set in the extended header.

97746 The result of the keyword conversion argument shall be the value from the applicable
97747 header field or extended header, without any trailing NULs.

97748 All keyword values used as conversion arguments shall be translated from the UTF-8
97749 encoding (or alternative encoding specified by any **hdrcharset** extended header record) to
97750 the character set appropriate for the local file system, user database, and so on, as
97751 applicable.

- 97752 7. An additional conversion specifier character, **T**, shall be used to specify time formats. The **T**
97753 conversion specifier character can be preceded by the sequence (*keyword=subformat*), where
97754 *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime**
97755 and the default subformat shall be:

97756 %b %e %H:%M %Y

- 97757 8. An additional conversion specifier character, *M*, shall be used to specify the file mode string
 97758 as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used.
 97759 For example, % . 1M writes the single character corresponding to the <entry type> field of the
 97760 *ls -l* command.
- 97761 9. An additional conversion specifier character, *D*, shall be used to specify the device for block
 97762 or special files, if applicable, in an implementation-defined format. If not applicable, and
 97763 (*keyword*) is specified, then this conversion shall be equivalent to %(*keyword*)u. If not
 97764 applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.
- 97765 10. An additional conversion specifier character, *F*, shall be used to specify a pathname. The *F*
 97766 conversion character can be preceded by a sequence of comma-separated keywords:
 97767 (*keyword*[,*keyword*] . . .)
- 97768 The values for all the keywords that are non-null shall be concatenated together, each
 97769 separated by a ' / '. The default shall be (**path**) if the keyword **path** is defined; otherwise,
 97770 the default shall be (**prefix,name**).
- 97771 11. An additional conversion specifier character, *L*, shall be used to specify a symbolic link
 97772 expansion. If the current file is a symbolic link, then %L shall expand to:
 97773 "%s -> %s", <value of keyword>, <contents> of link
- 97774 Otherwise, the %L conversion specification shall be the equivalent of %F.

OPERANDS

97775 The following operands shall be supported:

- 97776 *directory* The destination directory pathname for **copy** mode.
- 97777 *file* A pathname of a file to be copied or archived.
- 97778 *pattern* A pattern matching one or more pathnames of archive members. A pattern must
 97779 be given in the name-generating notation of the pattern matching notation in
 97780 [Section 2.13](#) (on page 2278), including the filename expansion rules in [Section](#)
 97781 [2.13.3](#) (on page 2279). The default, if no *pattern* is specified, is to select all members
 97782 in the archive.
 97783

STDIN

97784 In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a
 97785 text file containing a list of pathnames, one per line, without leading or trailing <blank>s.
 97786

97787 In **list** and **read** modes, if **-f** is not specified, the standard input shall be an archive file.

97788 Otherwise, the standard input shall not be used.

INPUT FILES

97789 The input file named by the *archive* option-argument, or standard input when the archive is read
 97790 from there, shall be a file formatted according to one of the specifications in the EXTENDED
 97791 DESCRIPTION section or some other implementation-defined format.
 97792

97793 The file */dev/tty* shall be used to write prompts and read responses.

ENVIRONMENT VARIABLES

97794 The following environment variables shall affect the execution of *pax*:

- 97796 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 97797 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables |
 97798 used to determine the values of locale categories.)
- 97799 *LC_ALL* If set to a non-empty string value, override the values of all the other
 97800 internationalization variables.

97801		<i>LC_COLLATE</i>	
97802			Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the pattern matching expressions for the <i>pattern</i> operand, the basic regular expression for the -s option, and the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category.
97803			
97804			
97805			
97806			
97807		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes used in the extended regular expression defined for the yesexpr locale keyword in the <i>LC_MESSAGES</i> category, and pattern matching.
97808			
97809			
97810			
97811			
97812		<i>LC_MESSAGES</i>	
97813			Determine the locale for the processing of affirmative responses that should be used to affect the format and contents of diagnostic messages written to standard error.
97814			
97815			
97816		<i>LC_TIME</i>	Determine the format and contents of date and time strings when the -v option is specified.
97817			
97818	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
97819		<i>TMPDIR</i>	Determine the pathname that provides part of the default global extended header record file, as described for the -o globexthdr= keyword in the OPTIONS section.
97820			
97821		<i>TZ</i>	Determine the timezone used to calculate date and time strings when the -v option is specified. If <i>TZ</i> is unset or null, an unspecified default timezone shall be used.
97822			

ASYNCHRONOUS EVENTS

Default.

STDOUT

In **write** mode, if **-f** is not specified, the standard output shall be the archive formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format (see **-x format**).

In **list** mode, when the **-olistopt=format** has been specified, the selected archive members shall be written to standard output using the format described under [List Mode Format Specifications](#) (on page 2933). In **list** mode without the **-olistopt=format** option, the table of contents of the selected archive members shall be written to standard output using the following format:

```
"%s\n", <pathname>
```

If the **-v** option is specified in **list** mode, the table of contents of the selected archive members shall be written to standard output using the following formats.

For pathnames representing hard links to previous members of the archive:

```
"%sΔ==Δ%s\n", <ls -l listing>, <linkname>
```

For all other pathnames:

```
"%s\n", <ls -l listing>
```

where *<ls -l listing>* shall be the format specified by the *ls* utility with the **-l** option. When writing pathnames in this format, it is unspecified what is written for fields for which the underlying archive format does not have the correct information, although the correct number of *<blank>*-separated fields shall be written.

In **list** mode, standard output shall not be buffered more than a line at a time.

97845 **STDERR**

97846 If **-v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the
 97847 standard error output using the following format:

97848 "%s\n", <pathname>

97849 These pathnames shall be written as soon as processing is begun on the file or archive member,
 97850 and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is
 97851 written when the file has been read or written.

97852 If the **-s** option is specified, and the replacement string has a trailing 'p', substitutions shall be
 97853 written to standard error in the following format:

97854 "%sΔ>Δ%s\n", <original pathname>, <new pathname>

97855 In all operating modes of *pax*, optional messages of unspecified format concerning the input
 97856 archive format and volume number, the number of files, blocks, volumes, and media parts as
 97857 well as other diagnostic messages may be written to standard error.

97858 In all formats, for both standard output and standard error, it is unspecified how non-printable
 97859 characters in pathnames or link names are written.

97860 When using the **-xpax** archive format, if a filename, link name, group name, owner name, or any
 97861 other field in an extended header record cannot be translated between the codeset in use for that
 97862 extended header record and the character set of the current locale, *pax* shall write a diagnostic
 97863 message to standard error, shall process the file as described for the **-o invalid=** option, and then
 97864 shall continue processing with the next file.

97865 **OUTPUT FILES**

97866 In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the
 97867 copied output files shall be the type of the file being copied. In either mode, existing files in the
 97868 destination hierarchy shall be overwritten only when all permission (**-p**), modification time (**-u**),
 97869 and invalid-value (**-o invalid=**) tests allow it.

97870 In **write** mode, the output file named by the **-f** option-argument shall be a file formatted
 97871 according to one of the specifications in the EXTENDED DESCRIPTION section, or some other
 97872 implementation-defined format.

97873 **EXTENDED DESCRIPTION**97874 **pax Interchange Format**

97875 A *pax* archive tape or file produced in the **-xpax** format shall contain a series of blocks. The
 97876 physical layout of the archive shall be identical to the **ustar** format described in [ustar](#)
 97877 [Interchange Format](#) (on page 2942). Each file archived shall be represented by the following
 97878 sequence:

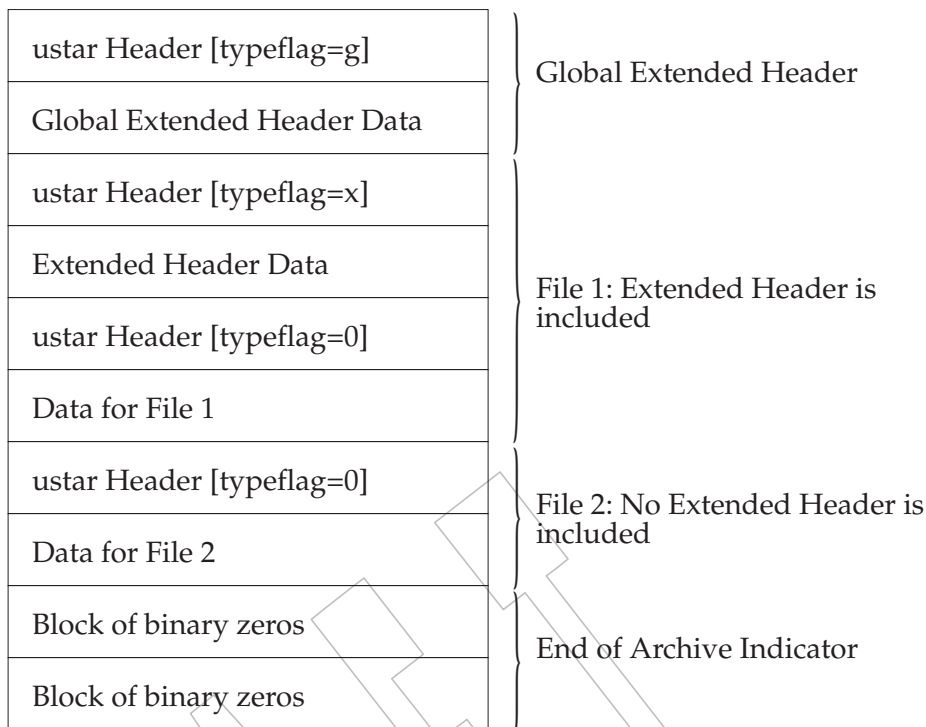
- 97879 • An optional header block with extended header records. This header block is of the form
 97880 described in [pax Header Block](#) (on page 2937), with a *typeflag* value of **x** or **g**. The
 97881 extended header records, described in [pax Extended Header](#) (on page 2938), shall be
 97882 included as the data for this header block.
- 97883 • A header block that describes the file. Any fields in the preceding optional extended
 97884 header shall override the associated fields in this header block for this file.
- 97885 • Zero or more blocks that contain the contents of the file.

97886 At the end of the archive file there shall be two 512-byte blocks filled with binary zeros,
 97887 interpreted as an end-of-archive indicator.

97888 A schematic of an example archive with global extended header records and two actual files is
 97889 shown in [Figure 4-1](#) (on page 2937). In the example, the second file in the archive has no

97890

extended header preceding it, presumably because it has no need for extended attributes.



97891

Figure 4-1 pax Format Archive Example

97892

pax Header Block

97893

The **pax** header block shall be identical to the **ustar** header block described in [ustar Interchange Format](#) (on page 2942), except that two additional *typeflag* values are defined:

97894

97895

x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in [pax Extended Header](#) (on page 2938).

97896

97897

97898

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in [pax Extended Header](#) (on page 2938). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* **g** global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

97899

97900

97901

97902

97903

97904

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to the ISO POSIX-2:1993 standard, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

97905

97906

97907

97908

97909

97910

A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit one) (hard link) may be included, which means that the *size* field may be greater than zero. Archives created by *pax -o linkdata* shall include these data blocks with the hard links.

97911

97912

97913

pax Extended Header97914
97915
97916
97917
97918
97919
97920

A **pax** extended header contains values that are inappropriate for the **ustar** header block because of limitations in that format: fields requiring a character encoding other than that described in the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar** header, and fields whose format or length do not fit the requirements of the **ustar** header. The values in an extended header add attributes to the following file (or files; see the description of the *typeflag* **g** header block) or override values in the following header block(s), as indicated in the following list of keywords.

97921

An extended header shall consist of one or more records, each constructed as follows:

97922

```
"%d %s=%s\n", <length>, <keyword>, <value>
```

97923
97924
97925
97926
97927
97928
97929
97930
97931

The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard UTF-8 encoding. The *<length>* field, *<blank>*, equals sign, and *<newline>* shown shall be limited to the portable character set, as encoded in UTF-8. The *<keyword>* fields can be any UTF-8 characters. The *<length>* field shall be the decimal length of the extended header record in octets, including the trailing *<newline>*. If there is a **hdrcharset** extended header in effect for a file, the *value* field for any **gname**, **linkpath**, **path**, and **uname** extended header records shall be encoded using the character set specified by the **hdrcharset** extended header record; otherwise, the *value* field shall be encoded using UTF-8. The *value* field for all other keywords specified by POSIX.1-200x shall be encoded using UTF-8.

97932
97933
97934
97935
97936
97937
97938
97939
97940

The *<keyword>* field shall be one of the entries from the following list or a keyword provided as an implementation extension. Keywords consisting entirely of lowercase letters, digits, and periods are reserved for future standardization. A keyword shall not include an equals sign. (In the following list, the notations "file(s)" or "block(s)" is used to acknowledge that a keyword affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode shall apply only when such a record has not already been provided through the use of the **-o** option. When used in **copy** mode, *pax* shall behave as if an archive had been created with applicable extended header records and then extracted.)

97941
97942
97943
97944
97945

atime The file access time for the following file(s), equivalent to the value of the *st_atime* member of the **stat** structure for a file, as described by the *stat()* function. The access time shall be restored if the process has the appropriate privilege required to do so. The format of the *<value>* shall be as described in [pax Extended Header File Times](#) (on page 2941).

97946
97947
97948

charset The name of the character set used to encode the data in the following file(s). The entries in the following table are defined to refer to known standards; additional names may be agreed on between the originator and recipient.

97949
97950
97951
97952
97953
97954
97955
97956
97957
97958
97959
97960
97961
97962
97963
97964
97965
97966

<value>	Formal Standard
ISO-IRΔ646Δ1990	ISO/IEC 646:1990
ISO-IRΔ8859Δ1Δ1998	ISO/IEC 8859-1:1998
ISO-IRΔ8859Δ2Δ1999	ISO/IEC 8859-2:1999
ISO-IRΔ8859Δ3Δ1999	ISO/IEC 8859-3:1999
ISO-IRΔ8859Δ4Δ1998	ISO/IEC 8859-4:1998
ISO-IRΔ8859Δ5Δ1999	ISO/IEC 8859-5:1999
ISO-IRΔ8859Δ6Δ1999	ISO/IEC 8859-6:1999
ISO-IRΔ8859Δ7Δ1987	ISO/IEC 8859-7:1987
ISO-IRΔ8859Δ8Δ1999	ISO/IEC 8859-8:1999
ISO-IRΔ8859Δ9Δ1999	ISO/IEC 8859-9:1999
ISO-IRΔ8859Δ10Δ1998	ISO/IEC 8859-10:1998
ISO-IRΔ8859Δ13Δ1998	ISO/IEC 8859-13:1998
ISO-IRΔ8859Δ14Δ1998	ISO/IEC 8859-14:1998
ISO-IRΔ8859Δ15Δ1999	ISO/IEC 8859-15:1999
ISO-IRΔ10646Δ2000	ISO/IEC 10646:2000
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

The encoding is included in an extended header for information only; when *pax* is used as described in POSIX.1-200x, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

97967
97968
97969

97970
97971

comment A series of characters used as a comment. All characters in the <value> field shall be ignored by *pax*.

97972
97973

gid The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

97974
97975
97976
97977
97978

gname The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the characters cannot be translated, and if neither the **-oinvalid=UTF-8** option nor the **-oinvalid=binary** option is specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

97979
97980
97981
97982
97983
97984
97985
97986
97987
97988

hdrcharset The name of the character set used to encode the value field of the **gname**, **linkpath**, **path**, and **uname** *pax* extended header records. The entries in the following table are defined to refer to known standards; additional names may be agreed between the originator and the recipient.

97989
97990
97991
97992

<value>	Formal Standard
ISO-IRΔ10646Δ2000ΔUTF-8	ISO/IEC 10646, UTF-8 encoding
BINARY	None.

97993
97994
97995

If no **hdrcharset** extended header record is specified, the default character set used to encode all values in extended header records shall be the ISO/IEC 10646-1:2000

97996
97997

97998		standard UTF-8 encoding.
97999		The BINARY entry indicates that all values recorded in extended headers for affected files are unencoded binary data from the underlying system.
98000		
98001	linkpath	The pathname of a link being created to another file, of any type, previously archived. This record shall override the <i>linkname</i> field in the following ustar header block(s). The following ustar header block shall determine the type of link created. If <i>typeflag</i> of the following header block is 1, it shall be a hard link. If <i>typeflag</i> is 2, it shall be a symbolic link and the linkpath value shall be the contents of the symbolic link. The <i>pax</i> utility shall translate the name of the link (contents of the symbolic link) from the encoding in the header to the character set appropriate for the local file system. When used in write or copy mode, <i>pax</i> shall include a linkpath extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.
98002		
98003		
98004		
98005		
98006		
98007		
98008		
98009		
98010		
98011		
98012	mtime	The file modification time of the following file(s), equivalent to the value of the <i>st_mtime</i> member of the stat structure for a file, as described in the <i>stat()</i> function. This record shall override the <i>mtime</i> field in the following header block(s). The modification time shall be restored if the process has the appropriate privilege required to do so. The format of the <i><value></i> shall be as described in pax Extended Header File Times (on page 2941).
98013		
98014		
98015		
98016		
98017		
98018	path	The pathname of the following file(s). This record shall override the <i>name</i> and <i>prefix</i> fields in the following header block(s). The <i>pax</i> utility shall translate the pathname of the file from the encoding in the header to the character set appropriate for the local file system.
98019		
98020		
98021		
98022		When used in write or copy mode, <i>pax</i> shall include a <i>path</i> extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.
98023		
98024		
98025	realtime.any	The keywords prefixed by “realtime.” are reserved for future standardization.
98026	security.any	The keywords prefixed by “security.” are reserved for future standardization.
98027	size	The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the <i>size</i> field in the following header block(s). When used in write or copy mode, <i>pax</i> shall include a <i>size</i> extended header record for each file with a size value greater than 8 589 934 591 (octal 77 777 777 777).
98028		
98029		
98030		
98031		
98032	uid	The user ID of the file owner, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the <i>uid</i> field in the following header block(s). When used in write or copy mode, <i>pax</i> shall include a <i>uid</i> extended header record for each file whose owner ID is greater than 2 097 151 (octal 7 777 777).
98033		
98034		
98035		
98036		
98037	uname	The owner of the following file(s), formatted as a user name in the user database. This record shall override the <i>uid</i> and <i>uname</i> fields in the following header block(s), and any <i>uid</i> extended header record. When used in read , copy , or list mode, <i>pax</i> shall translate the name from the encoding in the header record to the character set appropriate for the user database on the receiving system. If any of the characters cannot be translated, and if neither the -oinvalid=UTF-8 option nor the -oinvalid=binary option is specified, the results are implementation-defined. When used in write or copy mode, <i>pax</i> shall include a uname extended header record for each file whose user name cannot be represented entirely with the letters and digits of the portable character set.
98038		
98039		
98040		
98041		
98042		
98043		
98044		
98045		
98046		

98047 If the *<value>* field is zero length, it shall delete any header block field, previously entered
98048 extended header value, or global extended header value of the same name.

98049 If a keyword in an extended header record (or in a **-o** option-argument) overrides or deletes a
98050 corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block
98051 field.

98052 Unlike the **ustar** header block fields, NULs shall not delimit *<value>*s; all characters within the
98053 *<value>* field shall be considered data for the field. None of the length limitations of the **ustar**
98054 header block fields in Table 4-13 (on page 2942) shall apply to the extended header records.

98055 **pax Extended Header Keyword Precedence**

98056 This section describes the precedence in which the various header records and fields and
98057 command line options are selected to apply to a file in the archive. When *pax* is used in **read** or
98058 **list** modes, it shall determine a file attribute in the following sequence:

- 98059 1. If **-odelete=keyword-prefix** is used, the affected attributes shall be determined from step
98060 7., if applicable, or ignored otherwise.
- 98061 2. If **-okeyword:=** is used, the affected attributes shall be ignored.
- 98062 3. If **-okeyword:=value** is used, the affected attribute shall be assigned the value.
- 98063 4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the
98064 *<value>*. When extended header records conflict, the last one given in the header shall
98065 take precedence.
- 98066 5. If **-okeyword=value** is used, the affected attribute shall be assigned the value.
- 98067 6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be
98068 assigned the *<value>*. When global extended header records conflict, the last one given in
98069 the global header shall take precedence.
- 98070 7. Otherwise, the attribute shall be determined from the **ustar** header block.

98071 **pax Extended Header File Times**

98072 The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's
98073 modification time cannot be represented exactly in the **ustar** header logical record described in
98074 [ustar Interchange Format](#) (on page 2942). This can occur if the time is out of **ustar** range, or if
98075 the file system of the underlying implementation supports non-integer time granularities and
98076 the time is not an integer. All of these time records shall be formatted as a decimal representation
98077 of the time in seconds since the Epoch. If a period (*' . '*) decimal point character is present, the
98078 digits to the right of the point shall represent the units of a subsecond timing granularity, where
98079 the first digit is tenths of a second and each subsequent digit is a tenth of the previous digit. In
98080 **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value that is not
98081 greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time
98082 exactly if it can be represented exactly as a decimal number, and otherwise shall generate only
98083 enough digits so that the same time shall be recovered if the file is extracted on a system whose
98084 underlying implementation supports the same time granularity.

98085

ustar Interchange Format

98086

98087

98088

98089

98090

98091

98092

A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a fixed-size logical record of 512 octets (see below). Although this format may be thought of as being stored on 9-track industry-standard 12.7 mm (0.5 in) magnetic tape, other types of transportable media are not excluded. Each file archived shall be represented by a header logical record that describes the file, followed by zero or more logical records that give the contents of the file. At the end of the archive file there shall be two 512-octet logical records filled with binary zeros, interpreted as an end-of-archive indicator.

98093

98094

98095

98096

98097

The logical records may be grouped for physical I/O operations, as described under the **-b** *blocksize* and **-x** *ustar* options. Each group of logical records may be written with a single operation equivalent to the *write()* function. On magnetic tape, the result of this write shall be a single tape physical block. The last physical block shall always be the full size, so logical records after the two zero logical records may contain undefined data.

98098

98099

The header logical record shall be structured as shown in the following table. All lengths and offsets are in decimal.

98100

Table 4-13 ustar Header Block

98101

98102

98103

98104

98105

98106

98107

98108

98109

98110

98111

98112

98113

98114

98115

98116

98117

Field Name	Octet Offset	Length (in Octets)
<i>name</i>	0	100
<i>mode</i>	100	8
<i>uid</i>	108	8
<i>gid</i>	116	8
<i>size</i>	124	12
<i>mtime</i>	136	12
<i>chksum</i>	148	8
<i>typeflag</i>	156	1
<i>linkname</i>	157	100
<i>magic</i>	257	6
<i>version</i>	263	2
<i>uname</i>	265	32
<i>gname</i>	297	32
<i>devmajor</i>	329	8
<i>devminor</i>	337	8
<i>prefix</i>	345	155

98118

98119

98120

98121

98122

98123

98124

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646: 1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

98125

98126

98127

98128

98129

However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described in POSIX.1-200x. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

98130

98131

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

98132

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character.

98133 The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all
 98134 characters in the array contain non-NUL characters including the last character. The *version* field
 98135 is two octets containing the characters "00" (zero-zero). The *typeflag* contains a single character.
 98136 All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991
 98137 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

98138 The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be
 98139 formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up
 98140 to the first NUL character), a slash character, and *name*; otherwise, *name* is used alone. In either
 98141 case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall
 98142 be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname
 98143 does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any
 98144 part of the file—header or data—on the medium.

98145 The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a
 98146 *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall
 98147 notify the user of the error, and shall not attempt to store the link on the medium.

98148 The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit
 98149 representation. The encoded bits shall represent the following values:

98150 **Table 4-14** *ustar mode* Field

Bit Value	POSIX.1-200x Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

98164 When appropriate privilege is required to set one of these mode bits, and the user restoring the
 98165 files from the archive does not have the appropriate privilege, the mode bits for which the user
 98166 does not have appropriate privilege shall be ignored. Some of the mode bits in the archive
 98167 format are not mentioned elsewhere in this volume of POSIX.1-200x. If the implementation does
 98168 not support those bits, they may be ignored.

98169 The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

98170 The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type
 98171 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to
 98172 specify a file of type 5 (directory), the *size* field shall be interpreted as described under the
 98173 definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag*
 98174 field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size*
 98175 field is unspecified by this volume of POSIX.1-200x, and no data logical records shall be stored
 98176 on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the
 98177 *typeflag* field is set to any other value, the number of logical records written following the header
 98178 shall be $(size+511)/512$, ignoring any fraction in the result of the division.

98179 The *mtime* field shall be the modification time of the file at the time it was archived. It is the
 98180 ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained
 98181 from the *stat()* function.

98182 The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of
 98183 the simple sum of all octets in the header logical record. Each octet in the header shall be treated
 98184 as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the
 98185 precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is
 98186 treated as if it were all spaces.

98187 The *typeflag* field specifies the type of file archived. If a particular implementation does not
 98188 recognize the type, or the user does not have appropriate privilege to create that type, the file
 98189 shall be extracted as if it were a regular file if the file type is defined to have a meaning for the
 98190 *size* field that could cause data logical records to be written on the medium (see the previous
 98191 description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error
 98192 indicating that the conversion took place. All of the *typeflag* fields shall be coded in the
 98193 ISO/IEC 646:1991 standard IRV:

- | | | |
|---|------|--|
| 98194
98195
98196
98197 | 0 | Represents a regular file. For backwards-compatibility, a <i>typeflag</i> value of binary zero ('\0') should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a <i>typeflag</i> value of the ISO/IEC 646:1991 standard IRV '0'. |
| 98198
98199
98200
98201
98202 | 1 | Represents a file linked to another file, of any type, previously archived. Such files are identified by having the same device and file serial numbers, and pathnames that refer to different directory entries. All such files shall be archived as linked files. The linked-to name is specified in the <i>linkname</i> field with a NUL-character terminator if it is less than 100 octets in length. |
| 98203
98204 | 2 | Represents a symbolic link. The contents of the symbolic link shall be stored in the <i>linkname</i> field. |
| 98205
98206
98207
98208 | 3, 4 | Represent character special files and block special files respectively. In this case the <i>devmajor</i> and <i>devminor</i> fields shall contain information defining the device, the format of which is unspecified by this volume of POSIX.1-200x. Implementations may map the device specifications to their own local specification or may ignore the entry. |
| 98209
98210
98211
98212
98213 | 5 | Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the <i>size</i> field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A <i>size</i> field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the <i>size</i> field. |
| 98214
98215 | 6 | Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents. |
| 98216
98217
98218 | 7 | Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0). |
| 98219
98220 | A-Z | The letters 'A' to 'Z', inclusive, are reserved for custom implementations. All other values are reserved for future versions of this standard. |

98221 It is unspecified whether files with pathnames that refer to the same directory entry are archived
 98222 as linked files or as separate files. If they are archived as linked files, this means that attempting
 98223 to extract both pathnames from the resulting archive will always cause an error (unless the *-u*
 98224 option is used) because the link cannot be created.

98225 It is unspecified whether files with the same device and file serial numbers being appended to
 98226 an archive are treated as linked files to members that were in the archive before the append.

98227 Attempts to archive a socket using *ustar* interchange format shall produce a diagnostic message.
 98228 Handling of other file types is implementation-defined.

98229 The *magic* field is the specification that this archive was output in this archive format. If this field

98230 contains **ustar** (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by
 98231 NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646:1991 standard IRV
 98232 representation of the owner and group of the file, respectively (truncated to fit, if necessary).
 98233 When the file is restored by a privileged, protection-preserving version of the utility, the user
 98234 and group databases shall be scanned for these names. If found, the user and group IDs
 98235 contained within these files shall be used rather than the values contained within the *uid* and *gid*
 98236 fields.

98237 **cpio Interchange Format**

98238 The octet-oriented **cpio** archive format shall be a series of entries, each comprising a header that
 98239 describes the file, the name of the file, and then the contents of the file.

98240 An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used
 98241 only to make physical I/O more efficient. The last group of blocks shall always be at the full
 98242 size.

98243 For the octet-oriented **cpio** archive format, the individual entry information shall be in the order
 98244 indicated and described by the following table; see also the **<cpio.h>** header.

98245 **Table 4-15** Octet-Oriented cpio Archive Entry

Header Field Name	Length (in Octets)	Interpreted as
<i>c_magic</i>	6	Octal number
<i>c_dev</i>	6	Octal number
<i>c_ino</i>	6	Octal number
<i>c_mode</i>	6	Octal number
<i>c_uid</i>	6	Octal number
<i>c_gid</i>	6	Octal number
<i>c_nlink</i>	6	Octal number
<i>c_rdev</i>	6	Octal number
<i>c_mtime</i>	11	Octal number
<i>c_namesize</i>	6	Octal number
<i>c_filesize</i>	11	Octal number
Filename Field Name	Length	Interpreted as
<i>c_name</i>	<i>c_namesize</i>	Pathname string
File Data Field Name	Length	Interpreted as
<i>c_filedata</i>	<i>c_filesize</i>	Data

98262 **cpio Header**

98263 For each file in the archive, a header as defined previously shall be written. The information in
 98264 the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted
 98265 as octal numbers. The octal numbers shall be extended to the necessary length by appending the
 98266 ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result
 98267 is written to the most-significant digit of the stream of octets first. The fields shall be interpreted
 98268 as follows:

98269 *c_magic* Identify the archive as being a transportable archive by containing the identifying
 98270 value "070707".

98271 *c_dev, c_ino* Contains values that uniquely identify the file within the archive (that is, no files
 98272 contain the same pair of *c_dev* and *c_ino* values unless they are links to the same
 98273 file). The values shall be determined in an unspecified manner.

98274 *c_mode* Contains the file type and access permissions as defined in the following table.

98275 **Table 4-16** Values for *cpio c_mode* Field

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set <i>uid</i>
C_ISGID	002 000	Set <i>gid</i>
C_ISVTX	001 000	Reserved
File Type Name	Value	Indicates
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

98298 Directories, FIFOs, symbolic links, and regular files shall be supported on a system
 98299 conforming to this volume of POSIX.1-200x; additional values defined previously
 98300 are reserved for compatibility with existing systems. Additional file types may be
 98301 supported; however, such files should not be written to archives intended to be
 98302 transported to other systems.

98303 *c_uid* Contains the user ID of the owner.

98304 *c_gid* Contains the group ID of the group.

98305 *c_nlink* Contains a number greater than or equal to the number of links in the archive
 98306 referencing the file. If the *-a* option is used to append to a *cpio* archive, then the *pax*
 98307 utility need not account for the files in the existing part of the archive when
 98308 calculating the *c_nlink* values for the appended part of the archive, and need not
 98309 alter the *c_nlink* values in the existing part of the archive if additional files with the
 98310 same *c_dev* and *c_ino* values are appended to the archive.

98311 *c_rdev* Contains implementation-defined information for character or block special files.

98312 *c_mtime* Contains the latest time of modification of the file at the time the archive was
 98313 created.

98314 *c_namesize* Contains the length of the pathname, including the terminating NUL character.

98315 *c_filesize* Contains the length in octets of the data section following the header structure.

98316

cpio Filename

98317

The *c_name* field shall contain the pathname of the file. The length of this field in octets is the value of *c_namesize*.

98318

98319

If a filename is found on the medium that would create an invalid pathname, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored.

98320

98321

98322

All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes. However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described previously in this volume of POSIX.1-200x. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the local file system and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

98323

98324

98325

98326

98327

98328

98329

98330

98331

98332

98333

cpio File Data

98334

Following *c_name*, there shall be *c_filesize* octets of data. Interpretation of such data occurs in a manner dependent on the file. For regular files, the data shall consist of the contents of the file. For symbolic links, the data shall consist of the contents of the symbolic link. If *c_filesize* is zero, no data shall be contained in *c_filedata*.

98335

98336

98337

98338

When restoring from an archive:

98339

- If the user does not have the appropriate privilege to create a file of the specified type, *pax* shall ignore the entry and write an error message to standard error.

98340

98341

- Only regular files and symbolic links have data to be restored. Presuming a regular file meets any selection criteria that might be imposed on the format-reading utility by the user, such data shall be restored.

98342

98343

98344

- If a user does not have appropriate privilege to set a particular mode flag, the flag shall be ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this volume of POSIX.1-200x. If the implementation does not support those flags, they may be ignored.

98345

98346

98347

98348

cpio Special Entries

98349

FIFO special files, directories, and the trailer shall be recorded with *c_filesize* equal to zero. Symbolic links shall be recorded with *c_filesize* equal to the length of the contents of the symbolic link. For other special files, *c_filesize* is unspecified by this volume of POSIX.1-200x. The header for the next file entry in the archive shall be written directly after the last octet of the file entry preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive; the contents of octets in the last block of the archive following such a header are undefined.

98350

98351

98352

98353

98354

98355

EXIT STATUS

98356

The following exit values shall be returned:

98357

0 All files were processed successfully.

98358

>0 An error occurred.

CONSEQUENCES OF ERRORS

98359
98360
98361
98362
98363
98364

If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an archive, or cannot preserve the user ID, group ID, or file mode when the **-p** option is specified, a diagnostic message shall be written to standard error and a non-zero exit status shall be returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax* shall not, by default, create a second copy of the file.

98365
98366
98367
98368
98369

If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may have only partially extracted the file or (if the **-n** option was not specified) may have extracted a file of the same name as that specified by the user, but which is not the file the user wanted. Additionally, the file modes of extracted directories may have additional bits from the **S_IRWXU** mask set as well as incorrect modification and access times.

APPLICATION USAGE

98370
98371
98372
98373
98374
98375
98376
98377
98378
98379
98380

Caution is advised when using the **-a** option to append to a *cpio* format archive. If any of the files being appended happen to be given the same *c_dev* and *c_ino* values as a file in the existing part of the archive, then they may be treated as links to that file on extraction. Thus, it is risky to use **-a** with *cpio* format except when it is done on the same system that the original archive was created on, and with the same *pax* utility, and in the knowledge that there has been little or no file system activity since the original archive was created that could lead to any of the files appended being given the same *c_dev* and *c_ino* values as an unrelated file in the existing part of the archive. Also, when (intentionally) appending additional links to a file in the existing part of the archive, the *c_nlink* values in the modified archive can be smaller than the number of links to the file in the archive, which may mean that the links are not preserved on extraction.

98381
98382
98383
98384
98385

The **-p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio* implementations. In particular, the two utilities use **-m** in diametrically opposed ways. The **-p** option also provides a consistent means of extending the ways in which future file attributes can be addressed, such as for enhanced security systems or high-performance files. Although it may seem complex, there are really two modes that are most commonly used:

98386
98387
98388
98389

-p e “Preserve everything”. This would be used by the historical superuser, someone with all the appropriate privileges, to preserve all aspects of the files as they are recorded in the archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined attributes.

98390
98391
98392
98393

-p p “Preserve” the file mode bits. This would be used by the user with regular privileges who wished to preserve aspects of the file other than the ownership. The file times are preserved by default, but two other flags are offered to disable these and use the time of extraction.

98394
98395
98396
98397
98398

The one pathname per line format of standard input precludes pathnames containing `<newline>`s. Although such pathnames violate the portable filename guidelines, they may exist and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from historical archive programs. The problem can be avoided by listing filename arguments on the command line instead of on standard input.

98399
98400
98401
98402

It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this volume of POSIX.1-200x. Specifically, creating files of type block special or character special, restoring file access times unless the files are owned by the user (the **-t** option), or preserving file owner, group, and mode (the **-p** option) all probably require appropriate privileges.

98403
98404
98405

In **read** mode, implementations are permitted to overwrite files when the archive has multiple members with the same name. This may fail if permissions on the first version of the file do not permit it to be overwritten.

98406

The **cpio** and **ustar** formats can only support files up to 8 589 934 592 bytes ($8 * 2^{30}$) in size.

98407

When archives containing binary header information are listed, the filenames printed may

98408 cause strange behavior on some terminals.

98409 EXAMPLES

98410 The following command:

```
98411 pax -w -f /dev/rmt/lm .
```

98412 copies the contents of the current directory to tape drive 1, medium density (assuming historical
98413 System V device naming procedures—the historical BSD device name would be `/dev/rmt9`).

98414 The following commands:

```
98415 mkdir newdir
```

```
98416 pax -rw olddir newdir
```

98417 copy the `olddir` directory hierarchy to `newdir`.

```
98418 pax -r -s ',^//*usr//*,,' -f a.pax
```

98419 reads the archive `a.pax`, with all files rooted in `/usr` in the archive extracted relative to the current
98420 directory.

98421 Using the option:

```
98422 -o listopt="%M %(atime)T %(size)D %(name)s"
```

98423 overrides the default output description in Standard Output and instead writes:

```
98424 -rw-rw--- Jan 12 15:53 2003 1492 /usr/foo/bar
```

98425 Using the options:

```
98426 -o listopt='%L\t%(size)D\n%.7' \
```

```
98427 -o listopt='(name)s\n%(atime)T\n%T'
```

98428 overrides the default output description in Standard Output and instead writes:

```
98429 /usr/foo/bar -> /tmp 1492
```

```
98430 /usr/fo
```

```
98431 Jan 12 15:53 1991
```

```
98432 Jan 31 15:53 2003
```

98433 RATIONALE

98434 The `pax` utility was new for the ISO POSIX-2:1993 standard. It represents a peaceful compromise
98435 between advocates of the historical `tar` and `cpio` utilities.

98436 A fundamental difference between `cpio` and `tar` was in the way directories were treated. The `cpio`
98437 utility did not treat directories differently from other files, and to select a directory and its
98438 contents required that each file in the hierarchy be explicitly specified. For `tar`, a directory
98439 matched every file in the file hierarchy it rooted.

98440 The `pax` utility offers both interfaces; by default, directories map into the file hierarchy they root.
98441 The `-d` option causes `pax` to skip any file not explicitly referenced, as `cpio` historically did. The `tar`
98442 `-style` behavior was chosen as the default because it was believed that this was the more
98443 common usage and because `tar` is the more commonly available interface, as it was historically
98444 provided on both System V and BSD implementations.

98445 The data interchange format specification in this volume of POSIX.1-200x requires that processes
98446 with “appropriate privileges” shall always restore the ownership and permissions of extracted
98447 files exactly as archived. If viewed from the historic equivalence between superuser and
98448 “appropriate privileges”, there are two problems with this requirement. First, users running as
98449 superusers may unknowingly set dangerous permissions on extracted files. Second, it is
98450 needlessly limiting, in that superusers cannot extract files and own them as superuser unless the
98451 archive was created by the superuser. (It should be noted that restoration of ownerships and

98452 permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to
 98453 avoid these two problems, the *pax* specification has an additional “privilege” mechanism, the **-p**
 98454 option. Only a *pax* invocation with the privileges needed, and which has the **-p** option set using
 98455 the **e** specification character, has the “appropriate privilege” to restore full ownership and
 98456 permission information.

98457 Note also that this volume of POSIX.1-200x requires that the file ownership and access
 98458 permissions shall be set, on extraction, in the same fashion as the *creat()* function when provided
 98459 with the mode stored in the archive. This means that the file creation mask of the user is applied
 98460 to the file permissions.

98461 Users should note that directories may be created by *pax* while extracting files with permissions
 98462 that are different from those that existed at the time the archive was created. When extracting
 98463 sensitive information into a directory hierarchy that no longer exists, users are encouraged to set
 98464 their file creation mask appropriately to protect these files during extraction.

98465 The table of contents output is written to standard output to facilitate pipeline processing.

98466 An early proposal had hard links displaying for all pathnames. This was removed because it
 98467 complicates the output of the case where **-v** is not specified and does not match historical *cpio*
 98468 usage. The hard-link information is available in the **-v** display.

98469 The description of the **-l** option allows implementations to make hard links to symbolic links.
 98470 POSIX.1-200x does not specify any way to create a hard link to a symbolic link, but many
 98471 implementations provide this capability as an extension. If there are hard links to symbolic links
 98472 when an archive is created, the implementation is required to archive the hard link in the archive
 98473 (unless **-H** or **-L** is specified). When in **read** mode and in **copy** mode, implementations
 98474 supporting hard links to symbolic links should use them when appropriate.

98475 The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have
 98476 been brought along from historical usage. For example, there are restrictions on the length of
 98477 pathnames stored in the archive. When *pax* is used in **copy(-rw)** mode (copying directory
 98478 hierarchies), the ability to use extensions from the **-xpax** format overcomes these restrictions.

98479 The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard
 98480 block-size values for *cpio*, set when the **-B** option is specified. (The other default block-size value
 98481 for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240
 98482 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The
 98483 maximum block size of 32 256 bytes (2^{15} -512 bytes) is the largest multiple of 512 bytes that fits
 98484 into a signed 16-bit tape controller transfer register. There are known limitations in some
 98485 historical systems that would prevent larger blocks from being accepted. Historical values were
 98486 chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate
 98487 archives. Also, default block sizes for any file type other than character special file has been
 98488 deleted from this volume of POSIX.1-200x as unimportant and not likely to affect the structure of
 98489 the resulting archive.

98490 Implementations are permitted to modify the block-size value based on the archive format or the
 98491 device to which the archive is being written. This is to provide implementations with the
 98492 opportunity to take advantage of special types of devices, and it should not be used without a
 98493 great deal of consideration as it almost certainly decreases archive portability.

98494 The intended use of the **-n** option was to permit extraction of one or more files from the archive
 98495 without processing the entire archive. This was viewed by the standard developers as offering
 98496 significant performance advantages over historical implementations. The **-n** option in early
 98497 proposals had three effects; the first was to cause special characters in patterns to not be treated
 98498 specially. The second was to cause only the first file that matched a pattern to be extracted. The
 98499 third was to cause *pax* to write a diagnostic message to standard error when no file was found
 98500 matching a specified pattern. Only the second behavior is retained by this volume of
 98501 POSIX.1-200x, for many reasons. First, it is in general not acceptable for a single option to have

98502 multiple effects. Second, the ability to make pattern matching characters act as normal characters
 98503 is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the
 98504 special characters is useful because users may wish to normalize only a single special character
 98505 in a single filename. Fourth, given a more general escape mechanism, the previous behavior of
 98506 the `-n` option can be easily obtained using the `-s` option or a *sed* script. Finally, writing a
 98507 diagnostic message when a pattern specified by the user is unmatched by any file is useful
 98508 behavior in all cases.

98509 In this version, the `-n` was removed from the **copy** mode synopsis of *pax*; it is inapplicable
 98510 because there are no pattern operands specified in this mode.

98511 There is another method than *pax* for copying subtrees in POSIX.1-200x described as part of the
 98512 *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface,
 98513 while *pax* offers a finer granularity of control. Each provides additional functionality to the
 98514 other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is
 98515 the intention of the standard developers that the results be similar (using appropriate option
 98516 combinations in both utilities). The results are not required to be identical; there seemed
 98517 insufficient gain to applications to balance the difficulty of implementations having to guarantee
 98518 that the results would be exactly identical.

98519 A single archive may span more than one file. It is suggested that implementations provide
 98520 informative messages to the user on standard error whenever the archive file is changed.

98521 The `-d` option (do not create intermediate directories not listed in the archive) found in early
 98522 proposals was originally provided as a complement to the historic `-d` option of *cpio*. It has been
 98523 deleted.

98524 The `-s` option in early proposals specified a subset of the substitution command from the *ed*
 98525 utility. As there was no reason for only a subset to be supported, the `-s` option is now compatible
 98526 with the current *ed* specification. Since the delimiter can be any non-null character, the following
 98527 usage with single spaces is valid:

```
98528 pax -s " foo bar " ...
```

98529 The `-t` description is worded so as to note that this may cause the access time update caused by
 98530 some other activity (which occurs while the file is being read) to be overwritten.

98531 The default behavior of *pax* with regard to file modification times is the same as historical
 98532 implementations of *tar*. It is not the historical behavior of *cpio*.

98533 Because the `-i` option uses `/dev/tty`, utilities without a controlling terminal are not able to use
 98534 this option.

98535 The `-y` option, found in early proposals, has been deleted because a line containing a single
 98536 period for the `-i` option has equivalent functionality. The special lines for the `-i` option (a single
 98537 period and the empty line) are historical practice in *cpio*.

98538 In early drafts, a `-charmap` option was included to increase portability of files between systems
 98539 using different coded character sets. This option was omitted because it was apparent that
 98540 consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate
 98541 substitute.

98542 The ISO POSIX-2:1993 standard and ISO POSIX-1 standard requirements for *pax*, however,
 98543 made it very difficult to create a single archive containing files created using extended characters
 98544 provided by different locales. This version adds the **hdrcharset** keyword to make it possible to
 98545 archive files in these cases without dropping files due to translation errors.

98546 Translating filenames and other attributes from a locale's encoding to UTF-8 and then back again
 98547 can lose information, as the resulting filename might not be byte-for-byte equivalent to the
 98548 original. To avoid this problem, users can specify the `-o hdrcharset=binary` option, which will
 98549 cause the resulting archive to use binary format for all names and attributes. Such archives are

98550 not portable among hosts that use different native encodings (e.g., EBCDIC *versus* ASCII-based
 98551 encodings), but they will allow interchange among the vast majority of POSIX file systems in
 98552 practical use. Also, the `-o hdrcharset=binary` option will cause *pax* in **copy** mode to behave
 98553 more like other standard utilities such as *cp*.

98554 If the values specified by the `-o exthdr.name=value`, `-o globexthdr.name=value`, or by
 98555 `$TMPDIR` (if `-o globexthdr.name` is not specified) require a character encoding other than that
 98556 described in the ISO/IEC 646:1991 standard, a **path** extended header record will have to be
 98557 created for the file. If a **hdrcharset** extended header record is active for such headers, it will
 98558 determine the codeset used for the value field in these extended **path** header records. These **path**
 98559 extended header records always need to be created when writing an archive even if
 98560 **hdrcharset=binary** has been specified and would contain the same (binary) data that appears in
 98561 the **ustar** header record prefix and *name* fields. (In other words, an extended header **path** record
 98562 is always required to be generated if the *prefix* or *name* fields contain non-ASCII characters even
 98563 when **hdrcharset=binary** is also in effect for that file.)

98564 The `-k` option was added to address international concerns about the dangers involved in the
 98565 character set transformations of `-e` (if the target character set were different from the source, the
 98566 filenames might be transformed into names matching existing files) and also was made more
 98567 general to protect files transferred between file systems with different {NAME_MAX} values
 98568 (truncating a filename on a smaller system might also inadvertently overwrite existing files). As
 98569 stated, it prevents any overwriting, even if the target file is older than the source. This version
 98570 adds more granularity of options to solve this problem by introducing the `-oinvalid=option`—
 98571 specifically the **UTF-8** and **binary** actions. (Note that an existing file is still subject to overwriting
 98572 in this case. The `-k` option closes that loophole.)

98573 Some of the file characteristics referenced in this volume of POSIX.1-200x might not be
 98574 supported by some archive formats. For example, neither the **tar** nor **cpio** formats contain the
 98575 file access time. For this reason, the *e* specification character has been provided, intended to
 98576 cause all file characteristics specified in the archive to be retained.

98577 It is required that extracted directories, by default, have their access and modification times and
 98578 permissions set to the values specified in the archive. This has obvious problems in that the
 98579 directories are almost certainly modified after being extracted and that directory permissions
 98580 may not permit file creation. One possible solution is to create directories with the mode
 98581 specified in the archive, as modified by the *umask* of the user, with sufficient permissions to
 98582 allow file creation. After all files have been extracted, *pax* would then reset the access and
 98583 modification times and permissions as necessary.

98584 The list-mode formatting description borrows heavily from the one defined by the *printf* utility.
 98585 However, since there is no separate operand list to get conversion arguments, the format was
 98586 extended to allow specifying the name of the conversion argument as part of the conversion
 98587 specification.

98588 The **T** conversion specifier allows time fields to be displayed in any of the date formats. Unlike
 98589 the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past.
 98590 This makes parsing the output more predictable.

98591 The **D** conversion specifier handles the ability to display the major/minor or file size, as with *ls*,
 98592 by using `%-8(size)D`.

98593 The **L** conversion specifier handles the *ls* display for symbolic links.

98594 Conversion specifiers were added to generate existing known types used for *ls*.

98595
98596
98597
98598
98599
98600
98601
98602
98603
98604
98605
98606
98607
98608
98609
98610
98611
98612
98613
98614
98615
98616
98617
98618
98619
98620
98621
98622
98623
98624
98625
98626
98627
98628
98629
98630
98631
98632
98633
98634
98635
98636
98637
98638
98639

pax Interchange Format

The new POSIX data interchange format was developed primarily to satisfy international concerns that the **ustar** and **cpio** formats did not provide for file, user, and group names encoded in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

- Support international character encodings and locale information
- Support security information (ACLs, and so on)
- Support future file types, such as realtime or contiguous files
- Include data areas for implementation use
- Support systems with words larger than 32 bits and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:

- Encryption
- Compression
- Data translation between locales and codesets
- *inode* storage

The format chosen to support the goals is an extension of the **ustar** format. Of the two formats previously available, only the **ustar** format was selected for extensions because:

- It was easier to extend in an upwards-compatible way. It offered version flags and header block type fields with room for future standardization. The **cpio** format, while possessing a more flexible file naming methodology, could not be extended without breaking some theoretical implementation or using a dummy filename that could be a legitimate filename.
- Industry experience since the original “tar wars” fought in developing the ISO POSIX-1 standard has clearly been in favor of the **ustar** format, which is generally the default output format selected for *pax* implementations on new systems.

The new format was designed with one additional goal in mind: reasonable behavior when an older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated that a “format-reading utility” had to treat unrecognized *typeflag* values as regular files, this allowed the format to include all the extended information in a pseudo-regular file that preceded each real file. An option is given that allows the archive creator to set up reasonable names for these files on the older systems. Also, the normative text suggests that reasonable file access values be used for this **ustar** header block. Making these header files inaccessible for convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous version of *pax*), mandated the behavior of the format-reading utility when it encountered an unknown *typeflag*, but was silent about the other two fields.

Early proposals for the first version of this standard contained a proposed archive format that was based on compatibility with the standard for tape files (ISO 1001, similar to the format used historically on many mainframes and minicomputers). This format was overly complex and required considerable overhead in volume and header records. Furthermore, the standard developers felt that it would not be acceptable to the community of POSIX developers, so it was later changed to be a format more closely related to historical practice on POSIX systems.

98640 The prefix and name split of pathnames in **ustar** was replaced by the single path extended
 98641 header record for simplicity.

98642 The concept of a global extended header (*typeflag*) was controversial. If this were applied to an
 98643 archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape
 98644 could be a serious problem; a utility attempting to extract as many files as possible from a
 98645 damaged archive could lose a large percentage of file header information in this case. However,
 98646 if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers
 98647 considerable potential size reductions by eliminating redundant information. Thus, the text
 98648 warns against using the global method for unreliable media and provides a method for
 98649 implanting global information in the extended header for each file, rather than in the *typeflag* **g**
 98650 records.

98651 No facility for data translation or filtering on a per-file basis is included because the standard
 98652 developers could not invent an interface that would allow this in an efficient manner. If a filter,
 98653 such as encryption or compression, is to be applied to all the files, it is more efficient to apply the
 98654 filter to the entire archive as a single file. The standard developers considered interfaces that
 98655 would invoke a shell script for each file going into or out of the archive, but the system overhead
 98656 in this approach was considered to be too high.

98657 One such approach would be to have **filter=** records that give a pathname for an executable.
 98658 When the program is invoked, the file and archive would be open for standard input/output
 98659 and all the header fields would be available as environment variables or command-line
 98660 arguments. The standard developers did discuss such schemes, but they were omitted from
 98661 POSIX.1-200x due to concerns about excessive overhead. Also, the program itself would need to
 98662 be in the archive if it were to be used portably.

98663 There is currently no portable means of identifying the character set(s) used for a file in the file
 98664 system. Therefore, *pax* has not been given a mechanism to generate charset records
 98665 automatically. The only portable means of doing this is for the user to write the archive using the
 98666 **-ocharset=string** command line option. This assumes that all of the files in the archive use the
 98667 same encoding. The “implementation-defined” text is included to allow for a system that can
 98668 identify the encodings used for each of its files.

98669 The table of standards that accompanies the charset record description is acknowledged to be
 98670 very limited. Only a limited number of character set standards is reasonable for maximal
 98671 interchange. Any character set is, of course, possible by prior agreement. It was suggested that
 98672 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal
 98673 standards, and then only those with reasonably large followings, can be included here, simply as
 98674 a matter of practicality. The *<value>*s represent names of officially registered character sets in the
 98675 format required by the ISO 2375:1985 standard.

98676 The normal comma or *<blank>*-separated list rules are not followed in the case of keyword
 98677 options to allow ease of argument parsing for *getopts*.

98678 Further information on character encodings is in [pax Archive Character Set Encoding/Decoding](#)
 98679 (on page 2956).

98680 The standard developers have reserved keyword name space for vendor extensions. It is
 98681 suggested that the format to be used is:

98682 *VENDOR.keyword*

98683 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further
 98684 suggested that the keyword following the period be named differently than any of the standard
 98685 keywords so that it could be used for future standardization, if appropriate, by omitting the
 98686 *VENDOR* prefix.

98687 The *<length>* field in the extended header record was included to make it simpler to step
 98688 through the records, even if a record contains an unknown format (to a particular *pax*) with

98689 complex interactions of special characters. It also provides a minor integrity checkpoint within
 98690 the records to aid a program attempting to recover files from a damaged archive.

98691 There are no extended header versions of the *devmajor* and *devminor* fields because the
 98692 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific
 98693 extended keywords (such as *VENDOR.devmajor*) should be used.

98694 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly
 98695 on a symbolic name basis, as in **ustar**.

98696 Just as with the **ustar** format descriptions, the new format makes no special arrangements for
 98697 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file
 98698 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing
 98699 their labels, and mounting each in the proper sequence are considered to be implementation
 98700 details that cannot be described portably.

98701 The **pax** format is intended for interchange, not only for backup on a single (family of) systems.
 98702 It is not as densely packed as might be possible for backup:

- 98703 • It contains information as coded characters that could be coded in binary.
- 98704 • It identifies extended records with name fields that could be omitted in favor of a fixed-
 98705 field layout.
- 98706 • It translates names into a portable character set and identifies locale-related information,
 98707 both of which are probably unnecessary for backup.

98708 The requirements on restoring from an archive are slightly different from the historical wording,
 98709 allowing for non-monolithic privilege to bring forward as much as possible. In particular,
 98710 attributes such as “high performance file” might be broadly but not universally granted while
 98711 set-user-ID or *chown()* might be much more restricted. There is no implication in POSIX.1-200x
 98712 that the security information be honored after it is restored to the file hierarchy, in spite of what
 98713 might be improperly inferred by the silence on that topic. That is a topic for another standard.

98714 Links are recorded in the fashion described here because a link can be to any file type. It is
 98715 desirable in general to be able to restore part of an archive selectively and restore all of those files
 98716 completely. If the data is not associated with each link, it is not possible to do this. However, the
 98717 data associated with a file can be large, and when selective restoration is not needed, this can be
 98718 a significant burden. The archive is structured so that files that have no associated data can
 98719 always be restored by the name of any link name of any link, and the user may choose whether
 98720 data is recorded with each instance of a file that contains data. The format permits mixing of
 98721 both types of links in a single archive; this can be done for special needs, and *pax* is expected to
 98722 interpret such archives on input properly, despite the fact that there is no *pax* option that would
 98723 force this mixed case on output. (When **-o linkdata** is used, the output must contain the
 98724 duplicate data, but the implementation is free to include it or omit it when **-o linkdata** is not
 98725 used.)

98726 The time values are included as extended header records for those implementations needing
 98727 more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be
 98728 negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject
 98729 the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a
 98730 leading ‘-’. Even though some implementations can support finer file-time granularities than
 98731 seconds, the normative text requires support only for seconds since the Epoch because the
 98732 ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new
 98733 format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will
 98734 be affected by the **-p a** and **-p e** options. The *ctime* creation time (actually *inode* modification
 98735 time) is described with “appropriate privilege” so that it can be ignored when writing to the file
 98736 system. POSIX does not provide a portable means to change file creation time. Nothing is
 98737 intended to prevent a non-portable implementation of *pax* from restoring the value.

98738 The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the
 98739 sizes specified in the regular *tar* header. New file system architectures are emerging that will
 98740 exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits
 98741 for user and group IDs, but the extended header values were included for completeness,
 98742 allowing overrides for all of the decimal values in the *tar* header.

98743 The standard developers intended to describe the effective results of *pax* with regard to file
 98744 ownerships and permissions; implementations are not restricted in timing or sequencing the
 98745 restoration of such, provided the results are as specified.

98746 Much of the text describing the extended headers refers to use in “**write or copy modes**”. The
 98747 **copy** mode references are due to the normative text: “The effect of the copy shall be as if the
 98748 copied files were written to an archive file and then subsequently extracted ...”. There is
 98749 certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode,
 98750 but the effects must be as if it had.

98751 **pax Archive Character Set Encoding/Decoding**

98752 There is a need to exchange archives of files between systems of different native codesets.
 98753 Filenames, group names, and user names must be preserved to the fullest extent possible when
 98754 an archive is read on the receiving platform. Translation of the contents of files is not within the
 98755 scope of the *pax* utility.

98756 There will also be the need to represent characters that are not available on the receiving
 98757 platform. These unsupported characters cannot be automatically folded to the local set of
 98758 characters due to the chance of collisions. This could result in overwriting previous extracted
 98759 files from the archive or pre-existing files on the system.

98760 For these reasons, the codeset used to represent characters within the extended header records of
 98761 the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields
 98762 requiring translation include, at a minimum, filenames, user names, group names, and link
 98763 pathnames. Implementations may wish to have localized extended keywords that use non-
 98764 portable characters.

98765 The standard developers considered the following options:

- 98766 • The archive creator specifies the well-defined name of the source codeset. The receiver
 98767 must then recognize the codeset name and perform the appropriate translations to the
 98768 destination codeset.
- 98769 • The archive creator includes within the archive the character mapping table for the source
 98770 codeset used to encode extended header records. The receiver must then read the
 98771 character mapping table and perform the appropriate translations to the destination
 98772 codeset.
- 98773 • The archive creator translates the extended header records in the source codeset into a
 98774 canonical form. The receiver must then perform the appropriate translations to the
 98775 destination codeset.

98776 The approach that incorporates the name of the source codeset poses the problem of codeset
 98777 name registration, and makes the archive useless to *pax* archive decoders that do not recognize
 98778 that codeset.

98779 Because parts of an archive may be corrupted, the standard developers felt that including the
 98780 character map of the source codeset was too fragile. The loss of this one key component could
 98781 result in making the entire archive useless. (The difference between this and the global extended
 98782 header decision was that the latter has a workaround—duplicating extended header records on
 98783 unreliable media—but this would be too burdensome for large character set maps.)

98784 Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the

98785 cross-product of all source and destination codesets.

98786 To simplify the translation from the source codeset to the canonical form and from the canonical
 98787 form to the destination codeset, the standard developers decided that the internal representation
 98788 should be a stateless encoding. A stateless encoding is one where each codepoint has the same
 98789 meaning, without regard to the decoder being in a specific state. An example of a stateful
 98790 encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the
 98791 ISO/IEC 646: 1991 standard (equivalent to 7-bit ASCII).

98792 For these reasons, the standard developers decided to adopt a canonical format for the
 98793 representation of file information strings. The obvious, well-endorsed candidate is the
 98794 ISO/IEC 10646-1: 2000 standard (based in part on Unicode), which can be used to represent the
 98795 characters of virtually all standardized character sets. The standard developers initially agreed
 98796 upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters
 98797 provides a sufficiently rich set to represent all commonly-used codesets.

98798 However, the standard developers found that the 16-bit Unicode representation had some
 98799 problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character
 98800 made the extended header records twice as long for the case of strings coded entirely from
 98801 historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the
 98802 ISO/IEC 10646-1: 2000 standard. This multi-byte representation encodes UCS2 or UCS4
 98803 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In
 98804 addition, NUL octets and other characters possibly confusing to POSIX file systems do not
 98805 appear, except to represent themselves. It was realized that certain national codesets take up
 98806 more space after the encoding, due to their placement within the UCS range; it was felt that the
 98807 usefulness of the encoding of the names outweighs the disadvantage of size increase for file,
 98808 user, and group names.

98809 The encoding of UTF-8 is as follows:

98810	UCS4 Hex Encoding	UTF-8 Binary Encoding
98811	00000000-0000007F	0xxxxxxx
98812	00000080-000007FF	110xxxxx 10xxxxxx
98813	00000800-0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
98814	00010000-001FFFFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
98815	00200000-03FFFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
98816	04000000-7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

98817 where each 'x' represents a bit value from the character being translated.

98818 **ustar Interchange Format**

98819 The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of
 98820 the historical *tar* utility. The goal of these changes was not only to provide the functional
 98821 enhancements desired, but also to retain compatibility between new and old versions. This
 98822 compatibility has been retained. Archives written using the old archive format are compatible
 98823 with the new format.

98824 Implementors should be aware that the previous file format did not include a mechanism to
 98825 archive directory type files. For this reason, the convention of using a filename ending with slash
 98826 was adopted to specify a directory on the archive.

98827 The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for
 98828 {PATH_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname
 98829 be stored there without the use of the *prefix* field. Although the name field is known to be too
 98830 small to contain {PATH_MAX} characters, the value was not changed in this version of the
 98831 archive file format to retain backwards-compatibility, and instead the prefix was introduced.
 98832 Also, because of the earlier version of the format, there is no way to remove the restriction on the

98833 *linkname* field being limited in size to just that of the *name* field.

98834 The *size* field is required to be meaningful in all implementation extensions, although it could be
98835 zero. This is required so that the data blocks can always be properly counted.

98836 It is suggested that if device special files need to be represented that cannot be represented in the
98837 standard format, that one of the extension types (A-Z) be used, and that the additional
98838 information for the special file be represented as data and be reflected in the *size* field.

98839 Attempting to restore a special file type, where it is converted to ordinary data and conflicts with
98840 an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax*
98841 should not be able to overwrite the entries in, for example, */dev* in any case (whether the file is
98842 converted to another type or not). If run as a privileged user, it should be able to do so, and it
98843 would be considered a bug if it did not. The same is true of ordinary data files and similarly
98844 named special files; it is impossible to anticipate the needs of the user (who could really intend
98845 to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the
98846 protection system as required.

98847 The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a
98848 **ustar** archive. POSIX.1-200x does not require the contiguous file extension, but does define a
98849 standard way of archiving such files so that all conforming systems can interpret these file types
98850 in a meaningful and consistent manner. On a system that does not support extended file types,
98851 the *pax* utility should do the best it can with the file and go on to the next.

98852 The file protection modes are those conventionally used by the *ls* utility. This is extended beyond
98853 the usage in the ISO POSIX-2 standard to support the “shared text” or “sticky” bit. It is intended
98854 that the conformance document should not document anything beyond the existence of and
98855 support of such a mode. Further extensions are expected to these bits, particularly with
98856 overloading the set-user-ID and set-group-ID flags.

98857 **cpio Interchange Format**

98858 The reference to appropriate privilege in the **cpio** format refers to an error on standard output;
98859 the **ustar** format does not make comparable statements.

98860 The model for this format was the historical System V *cpio-c* data interchange format. This
98861 model documents the portable version of the **cpio** format and not the binary version. It has the
98862 flexibility to transfer data of any type described within POSIX.1-200x, yet is extensible to transfer
98863 data types specific to extensions beyond POSIX.1-200x (for example, contiguous files). Because it
98864 describes existing practice, there is no question of maintaining upwards-compatibility.

98865 **cpio Header**

98866 There has been some concern that the size of the *c_ino* field of the header is too small to handle
98867 those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used
98868 strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as
98869 the *inode* number of the file in the location from which that file is extracted.

98870 The name *c_magic* is based on historical usage.

98871 **cpio Filename**

98872 For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to
98873 describe the pathname without the addition of any other header fields (the NUL character
98874 would be included in this count). {PATH_MAX} is the minimum value for pathname size,
98875 documented as 256 bytes. However, an implementation may use *c_namesize* to determine the
98876 exact length of the pathname. With the current description of the **<cpio.h>** header, this
98877 pathname size can be as large as a number that is described in six octal digits.

98878 Two values are documented under the *c_mode* field values to provide for extensibility for known

98879 file types:

98880 **0110 000** Reserved for contiguous files. The implementation may treat the rest of the
 98881 information for this archive like a regular file. If this file type is undefined, the
 98882 implementation may create the file as a regular file.

98883 This provides for extensibility of the **cpio** format while allowing for the ability to read old
 98884 archives. Files of an unknown type may be read as “regular files” on some implementations. On
 98885 a system that does not support extended file types, the *pax* utility should do the best it can with
 98886 the file and go on to the next.

98887 FUTURE DIRECTIONS

98888 None.

98889 SEE ALSO

98890 [Chapter 2](#) (on page 2245), *cp*, *ed*, *getopts*, *ls*, *printf*

98891 XBD [Section 3.168](#) (on page 56), [Chapter 5](#) (on page 107), [Chapter 8](#) (on page 159), [Section 12.2](#)
 98892 (on page 201), [<cpio.h>](#)

98893 XSH *chown*, *creat()*, *fstatat()*, *mkdir*, *mkfifo*, *utime()*, *write*

98894 CHANGE HISTORY

98895 First released in Issue 4.

98896 Issue 5

98897 A note is added to the APPLICATION USAGE indicating that the **cpio** and **tar** formats can only
 98898 support files up to 8 gigabytes in size.

98899 Issue 6

98900 The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- 98901 • Support has been added for symbolic links in the options and interchange formats.
- 98902 • A new format has been devised, based on extensions to **ustar**.
- 98903 • References to the “extended” **tar** and **cpio** formats derived from the POSIX.1-1990
 98904 standard have been changed to remove the “extended” adjective because this could cause
 98905 confusion with the extended *tar* header added in this version. (All references to *tar* are
 98906 actually to **ustar**.)

98907 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

98908 IEEE PASC Interpretation 1003.2 #168 is applied, clarifying that *mkdir()* and *mkfifo()* calls can
 98909 ignore an [EEXIST] error when extracting an archive.

98910 IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when
 98911 in **read** mode.

98912 IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **-t** option.

98913 IEEE PASC Interpretation 1003.2 #195 is applied.

98914 IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **-H**, **-L**,
 98915 and **-l** options.

98916 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/35 is applied, adding the process ID of
 98917 the *pax* process into certain fields. This change provides a method for the implementation to
 98918 ensure that different instances of *pax* extracting a file named **/a/b/foo** will not collide when
 98919 processing the extended header information associated with **foo**.

98920 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/36 is applied, changing **-x B** to **-x pax** in
 98921 the OPTIONS section.

98922 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/20 is applied, updating the SYNOPSIS to

- 98923 be consistent with the normative text.
- 98924 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/21 is applied, updating the
98925 DESCRIPTION to describe the behavior when files to be linked are symbolic links and the
98926 system is not capable of making hard links to symbolic links.
- 98927 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/22 is applied, updating the OPTIONS
98928 section to describe the behavior for how multiple **-odelete=pattern** options are to be handled.
- 98929 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/23 is applied, updating the **write** option
98930 within the OPTIONS section.
- 98931 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/24 is applied, adding a paragraph into
98932 the OPTIONS section that states that specifying more than one of the mutually-exclusive options
98933 (**-H** and **-L**) is not considered an error and that the last option specified will determine the
98934 behavior of the utility.
- 98935 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/25 is applied, removing the *ctime*
98936 paragraph within the EXTENDED DESCRIPTION. There is a contradiction in the definition of
98937 the *ctime* keyword for the *pax* extended header, in that the *st_ctime* member of the **stat** structure
98938 does not refer to a file creation time. No field in the standard **stat** structure from **<sys/stat.h>**
98939 includes a file creation time.
- 98940 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/26 is applied, making it clear that *typeflag*
98941 1 (**ustar** Interchange Format) applies not only to files that are hard-linked, but also to files that
98942 are aliased via symlinks.
- 98943 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/27 is applied, clarifying the *cpio c_nlink*
98944 field.
- 98945 **Issue 7**
- 98946 Austin Group Interpretations 1003.1-2001 #011, #036, #086, and #109 are applied. +
- 98947 SD5-XCU-ERN-2 is applied, making **-c** and **-n** mutually-exclusive in the SYNOPSIS.
- 98948 SD5-XCU-ERN-3 is applied, revising the default behavior of **-H** and **-L**.
- 98949 SD5-XCU-ERN-5, SD5-XCU-ERN-6, SD5-XCU-ERN-7, SD5-XCU-ERN-60 are applied.
- 98950 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS. -

98951 **NAME**

98952 pr — print files

98953 **SYNOPSIS**

98954 pr [+page] [-column] [-adFmrt] [-e[*char*][*gap*]] [-h *header*] [-i[*char*][*gap*]]
 98955 XSI [-l *lines*] [-n[*char*][*width*]] [-o *offset*] [-s[*char*]] [-w *width*] [-fp]
 98956 [*file...*]

98957 **DESCRIPTION**

98958 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be
 98959 read, formatted, and written to standard output. By default, the input shall be separated into
 98960 66-line pages, each with:

- 98961 • A 5-line header that includes the page number, date, time, and the pathname of the file
- 98962 • A 5-line trailer consisting of blank lines

98963 If standard output is associated with a terminal, diagnostic messages shall be deferred until the
 98964 *pr* utility has completed processing.

98965 When options specifying multi-column output are specified, output text columns shall be of
 98966 equal width; input lines that do not fit into a text column shall be truncated. By default, text
 98967 columns shall be separated with at least one <blank>.

98968 **OPTIONS**

98969 The *pr* utility shall conform to XBD [Section 12.2](#) (on page 201), except that: the *page* option has a
 98970 '+' delimiter; *page* and *column* can be multi-digit numbers; some of the option-arguments are
 98971 optional; and some of the option-arguments cannot be specified as separate arguments from the
 98972 preceding option letter. In particular, the *-s* option does not allow the option letter to be
 98973 separated from its argument, and the options *-e*, *-i*, and *-n* require that both arguments, if
 98974 present, not be separated from the option letter.

98975 The following options shall be supported. In the following option descriptions, *column*, *lines*,
 98976 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

98977 **+page** Begin output at page number *page* of the formatted input.

98978 **-column** Produce multi-column output that is arranged in *column* columns (the default shall
 98979 be 1) and is written down each column in the order in which the text is received
 98980 from the input file. This option should not be used with *-m*. The options *-e* and *-i*
 98981 shall be assumed for multiple text-column output. Whether or not text columns are
 98982 produced with identical vertical lengths is unspecified, but a text column shall
 98983 never exceed the length of the page (see the *-l* option). When used with *-t*, use the
 98984 minimum number of lines to write the output.

98985 **-a** Modify the effect of the *-column* option so that the columns are filled across the
 98986 page in a round-robin order (for example, when *column* is 2, the first input line
 98987 heads column 1, the second heads column 2, the third is the second line in column
 98988 1, and so on).

98989 **-d** Produce output that is double-spaced; append an extra <newline> following every
 98990 <newline> found in the input.

98991 **-e[*char*][*gap*]**

98992 Expand each input <tab> to the next greater column position specified by the
 98993 formula $n*gap+1$, where *n* is an integer > 0. If *gap* is zero or is omitted, it shall
 98994 default to 8. All <tab>s in the input shall be expanded into the appropriate number
 98995 of <space>s. If any non-digit character, *char*, is specified, it shall be used as the

98996		input <tab>. If the first character of the -e option-argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> .
98997		
98998	XSI	-f Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s. Pause before beginning the first page if the standard output is associated with a terminal.
98999		
99000		
99001		-F Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s.
99002		
99003		-h header Use the string <i>header</i> to replace the contents of the <i>file</i> operand in the page header.
99004		-i[char][gap] In output, replace <space>s with <tab>s wherever one or more adjacent <space>s reach column positions <i>gap</i> +1, 2* <i>gap</i> +1, 3* <i>gap</i> +1, and so on. If <i>gap</i> is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, <i>char</i> , is specified, it shall be used as the output <tab>. If the first character of the -i option-argument is a digit, the entire option-argument shall be assumed to be <i>gap</i> .
99005		
99006		
99007		
99008		
99009		
99010		-l lines Override the 66-line default and reset the page length to <i>lines</i> . If <i>lines</i> is not greater than the sum of both the header and trailer depths (in lines), the <i>pr</i> utility shall suppress both the header and trailer, as if the -t option were in effect.
99011		
99012		
99013		-m Merge files. Standard output shall be formatted so the <i>pr</i> utility writes one line from each file specified by a <i>file</i> operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine <i>file</i> operands.
99014		
99015		
99016		
99017		-n[char][width]
99018		Provide <i>width</i> -digit line numbering (default for <i>width</i> shall be 5). The number shall occupy the first <i>width</i> column positions of each text column of default output or each line of -m output. If <i>char</i> (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for <i>char</i> is a <tab>).
99019		
99020		
99021		
99022		
99023		-o offset Each line of output shall be preceded by offset <space>s. If the -o option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the -w option below).
99024		
99025		
99026		-p Pause before beginning each page if the standard output is directed to a terminal (<i>pr</i> shall write an <alert> to standard error and wait for a <carriage-return> to be read on <i>/dev/tty</i>).
99027		
99028		
99029		-r Write no diagnostic reports on failure to open files.
99030		-s[char] Separate text columns by the single character <i>char</i> instead of by the appropriate number of <space>s (default for <i>char</i> shall be <tab>).
99031		
99032		-t Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.
99033		
99034		
99035		-w width Set the width of the line to <i>width</i> column positions for multiple text-column output only. If the -w option is not specified and the -s option is not specified, the default width shall be 72. If the -w option is not specified and the -s option is specified, the default width shall be 512.
99036		
99037		
99038		
99039		For single column output, input lines shall not be truncated.

99040 **OPERANDS**

99041 The following operand shall be supported:

99042 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*
99043 operand is '-', the standard input shall be used.99044 **STDIN**99045 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
99046 See the INPUT FILES section.99047 **INPUT FILES**

99048 The input files shall be text files.

99049 The file */dev/tty* shall be used to read responses required by the **-p** option.99050 **ENVIRONMENT VARIABLES**99051 The following environment variables shall affect the execution of *pr*:99052 *LANG* Provide a default value for the internationalization variables that are unset or null. |
99053 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
99054 used to determine the values of locale categories.)99055 *LC_ALL* If set to a non-empty string value, override the values of all the other
99056 internationalization variables.99057 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
99058 characters (for example, single-byte as opposed to multi-byte characters in
99059 arguments and input files) and which characters are defined as printable (character
99060 class **print**). Non-printable characters are still written to standard output, but are
99061 not counted for the purpose for column-width and line-length calculations.99062 *LC_MESSAGES*
99063 Determine the locale that should be used to affect the format and contents of
99064 diagnostic messages written to standard error.99065 *LC_TIME* Determine the format of the date and time for use in writing header lines.99066 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.99067 *TZ* Determine the timezone used to calculate date and time strings written in header
99068 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.99069 **ASYNCHRONOUS EVENTS**99070 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
99071 messages to the screen before terminating.99072 **STDOUT**99073 The *pr* utility output shall be a paginated version of the original file (or files). This pagination
99074 shall be accomplished using either `<form-feed>`s or a sequence of `<newline>`s, as controlled by
99075 XSI the **-F** or **-f** option. Page headers shall be generated unless the **-t** option is specified. The page
99076 headers shall be of the form:99077 `"\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>`99078 In the POSIX locale, the `<output of date>` field, representing the date and time of last modification
99079 of the input file (or the current date and time if the input file is standard input), shall be
99080 equivalent to the output of the following command as it would appear if executed at the given
99081 time:99082 `date "+%b %e %H:%M %Y"`99083 without the trailing `<newline>`, if the page being written is from standard input. If the page
99084 being written is not from standard input, in the POSIX locale, the same format shall be used, but

99085 the time used shall be the modification time of the file corresponding to *file* instead of the current
 99086 time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format and
 99087 order of presentation of this field may be used.

99088 If the standard input is used instead of a *file* operand, the *<file>* field shall be replaced by a null
 99089 string.

99090 If the **-h** option is specified, the *<file>* field shall be replaced by the *header* argument.

99091 **STDERR**

99092 The standard error shall be used for diagnostic messages and for alerting the terminal when **-p**
 99093 is specified.

99094 **OUTPUT FILES**

99095 None.

99096 **EXTENDED DESCRIPTION**

99097 None.

99098 **EXIT STATUS**

99099 The following exit values shall be returned:

99100 0 Successful completion.

99101 >0 An error occurred.

99102 **CONSEQUENCES OF ERRORS**

99103 Default.

99104 **APPLICATION USAGE**

99105 A conforming application must protect its first operand, if it starts with a plus sign, by preceding
 99106 it with the **--** argument that denotes the end of the options. For example, *pr+x* could be
 99107 interpreted as an invalid page number or a *file* operand.

99108 **EXAMPLES**

- 99109 1. Print a numbered list of all files in the current directory:

```
99110 ls -a | pr -n -h "Files in $(pwd)."
```

- 99111 2. Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
99112 pr -3d -h "file list" file1 file2
```

- 99113 3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, ...:

```
99114 pr -e9 -t <file1 >file2
```

99115 **RATIONALE**

99116 This utility is one of those that does not follow the Utility Syntax Guidelines because of its
 99117 historical origins. The standard developers could have added new options that obeyed the
 99118 guidelines (and marked the old options obsolescent) or devised an entirely new utility; there are
 99119 examples of both actions in this volume of POSIX.1-200x. Because of its widespread use by
 99120 historical applications, the standard developers decided to exempt this version of *pr* from many
 99121 of the guidelines.

99122 Implementations are required to accept option-arguments to the **-h**, **-l**, **-o**, and **-w** options
 99123 whether presented as part of the same argument or as a separate argument to *pr*, as suggested by
 99124 the Utility Syntax Guidelines. The **-n** and **-s** options, however, are specified as in historical
 99125 practice because they are frequently specified without their optional arguments. If a *<blank>*
 99126 were allowed before the option-argument in these cases, a *file* operand could mistakenly be
 99127 interpreted as an option-argument in historical applications.

99128 The text about the minimum number of lines in multi-column output was included to ensure

99129 that a best effort is made in balancing the length of the columns. There are known historical
 99130 implementations in which, for example, 60-line files are listed by *pr -2* as one column of 56 lines
 99131 and a second of 4. Although this is not a problem when a full page with headers and trailers is
 99132 produced, it would be relatively useless when used with *-t*.

99133 Historical implementations of the *pr* utility have differed in the action taken for the *-f* option.
 99134 BSD uses it as described here for the *-F* option; System V uses it to change trailing *<newline>*s
 99135 on each page to a *<form-feed>* and, if standard output is a TTY device, sends an *<alert>* to
 99136 standard error and reads a line from */dev/tty* before the first page. There were strong arguments
 99137 from both sides of this issue concerning historical practice and as a result the *-F* option was
 99138 added. XSI-conformant systems support the System V historical actions for the *-f* option.

99139 The *<output of date>* field in the *-l* format is specified only for the POSIX locale. As noted, the
 99140 format can be different in other locales. No mechanism for defining this is present in this volume
 99141 of POSIX.1-200x, as the appropriate vehicle is a message catalog; that is, the format should be
 99142 specified as a “message”.

99143 FUTURE DIRECTIONS

99144 None.

99145 SEE ALSO

99146 *expand*, *lp*

99147 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

99148 CHANGE HISTORY

99149 First released in Issue 2.

99150 Issue 6

99151 The following new requirements on POSIX implementations derive from alignment with the
 99152 Single UNIX Specification:

- 99153 • The *-p* option is added.

99154 The normative text is reworded to avoid use of the term “must” for application requirements.

99155 Issue 7

99156 PASC Interpretation 1003.2-92 #151 (SD5-XCU-ERN-44) is applied.

99157 Austin Group Interpretation 1003.1-2001 #093 is applied.

99158 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

99159 **NAME**99160 `printf` — write formatted output99161 **SYNOPSIS**99162 `printf format [argument...]`99163 **DESCRIPTION**99164 The *printf* utility shall write formatted operands to the standard output. The *argument* operands
99165 shall be formatted under control of the *format* operand.99166 **OPTIONS**

99167 None.

99168 **OPERANDS**

99169 The following operands shall be supported:

99170 *format* A string describing the format to use to write the remaining operands. See the
99171 EXTENDED DESCRIPTION section.99172 *argument* The strings to be written to standard output, under the control of *format*. See the
99173 EXTENDED DESCRIPTION section.99174 **STDIN**

99175 Not used.

99176 **INPUT FILES**

99177 None.

99178 **ENVIRONMENT VARIABLES**99179 The following environment variables shall affect the execution of *printf*:99180 *LANG* Provide a default value for the internationalization variables that are unset or null. |
99181 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
99182 used to determine the values of locale categories.)99183 *LC_ALL* If set to a non-empty string value, override the values of all the other
99184 internationalization variables.99185 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
99186 characters (for example, single-byte as opposed to multi-byte characters in
99187 arguments).99188 *LC_MESSAGES*99189 Determine the locale that should be used to affect the format and contents of
99190 diagnostic messages written to standard error.99191 *LC_NUMERIC*99192 Determine the locale for numeric formatting. It shall affect the format of numbers
99193 written using the *e*, *E*, *f*, *g*, and *G* conversion specifier characters (if supported).99194 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.99195 **ASYNCHRONOUS EVENTS**

99196 Default.

99197 **STDOUT**

99198 See the EXTENDED DESCRIPTION section.

99199 **STDERR**
 99200 The standard error shall be used only for diagnostic messages.

99201 **OUTPUT FILES**
 99202 None.

99203 EXTENDED DESCRIPTION

99204 The *format* operand shall be used as the *format* string described in XBD Chapter 5 (on page 107)
 99205 with the following exceptions:

- 99206 1. A <space> in the format string, in any context other than a flag of a conversion
 99207 specification, shall be treated as an ordinary character that is copied to the output.
- 99208 2. A ' Δ ' character in the format string shall be treated as a ' Δ ' character, not as a <space>.
- 99209 3. In addition to the escape sequences shown in XBD Chapter 5 (on page 107) ('\\', '\a',
 99210 '\b', '\f', '\n', '\r', '\t', '\v'), "\ddd", where *ddd* is a one, two, or three-digit
 99211 octal number, shall be written as a byte with the numeric value specified by the octal
 99212 number.
- 99213 4. The implementation shall not precede or follow output from the *d* or *u* conversion
 99214 specifiers with <blank>s not specified by the *format* operand.
- 99215 5. The implementation shall not precede output from the *o* conversion specifier with zeros
 99216 not specified by the *format* operand.
- 99217 6. The *e*, *E*, *f*, *g*, and *G* conversion specifiers need not be supported.
- 99218 7. An additional conversion specifier character, *b*, shall be supported as follows. The
 99219 argument shall be taken to be a string that may contain backslash-escape sequences. The
 99220 following backslash-escape sequences shall be supported:
 - 99221 — The escape sequences listed in XBD Chapter 5 (on page 107) ('\\', '\a', '\b',
 99222 '\f', '\n', '\r', '\t', '\v'), which shall be converted to the characters they
 99223 represent
 - 99224 — "\0ddd", where *ddd* is a zero, one, two, or three-digit octal number that shall be
 99225 converted to a byte with the numeric value specified by the octal number
 - 99226 — '\c', which shall not be written and shall cause *printf* to ignore any remaining
 99227 characters in the string operand containing it, any remaining string operands, and
 99228 any additional characters in the *format* operand

99229 The interpretation of a backslash followed by any other sequence of characters is
 99230 unspecified.

99231 Bytes from the converted string shall be written until the end of the string or the number
 99232 of bytes indicated by the precision specification is reached. If the precision is omitted, it
 99233 shall be taken to be infinite, so all bytes up to the end of the converted string shall be
 99234 written.

- 99235 8. For each conversion specification that consumes an argument, the next argument operand
 99236 shall be evaluated and converted to the appropriate type for the conversion as specified
 99237 below.
- 99238 9. The *format* operand shall be reused as often as necessary to satisfy the argument
 99239 operands. Any extra *c* or *s* conversion specifiers shall be evaluated as if a null string
 99240 argument were supplied; other extra conversion specifications shall be evaluated as if a
 99241 zero argument were supplied. If the *format* operand contains no conversion specifications
 99242 and *argument* operands are present, the results are unspecified.

- 99243 10. If a character sequence in the *format* operand begins with a '%' character, but does not
99244 form a valid conversion specification, the behavior is unspecified.

99245 The *argument* operands shall be treated as strings if the corresponding conversion specifier is b,
99246 c, or s; otherwise, it shall be evaluated as a C constant, as described by the ISO C standard, with
99247 the following extensions:

- 99248 • A leading plus or minus sign shall be allowed.
- 99249 • If the leading character is a single-quote or double-quote, the value shall be the numeric
99250 value in the underlying codeset of the character following the single-quote or double-
99251 quote.

99252 If an argument operand cannot be completely converted into an internal value appropriate to
99253 the corresponding conversion specification, a diagnostic message shall be written to standard
99254 error and the utility shall not exit with a zero exit status, but shall continue processing any
99255 remaining operands and shall write the value accumulated at the time the error was detected to
99256 standard output.

99257 It is not considered an error if an argument operand is not completely used for a c or s
99258 conversion or if a string operand's first or second character is used to get the numeric value of a
99259 character.

99260 EXIT STATUS

99261 The following exit values shall be returned:

- 99262 0 Successful completion.
- 99263 >0 An error occurred.

99264 CONSEQUENCES OF ERRORS

99265 Default.

99266 APPLICATION USAGE

99267 The floating-point formatting conversion specifications of *printf()* are not required because all
99268 arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations
99269 and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-
99270 point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility
99271 cannot really be used to format *bc* output; it does not support arbitrary precision.)
99272 Implementations are encouraged to support the floating-point conversions as an extension.

99273 Note that this *printf* utility, like the *printf()* function defined in the System Interfaces volume of
99274 POSIX.1-200x on which it is based, makes no special provision for dealing with multi-byte
99275 characters when using the %c conversion specification or when a precision is specified in a %b or
99276 %s conversion specification. Applications should be extremely cautious using either of these
99277 features when there are multi-byte characters in the character set.

99278 No provision is made in this volume of POSIX.1-200x which allows field widths and precisions
99279 to be specified as '*' since the '*' can be replaced directly in the *format* operand using shell
99280 variable substitution. Implementations can also provide this feature as an extension if they so
99281 choose.

99282 Hexadecimal character constants as defined in the ISO C standard are not recognized in the
99283 *format* operand because there is no consistent way to detect the end of the constant. Octal
99284 character constants are limited to, at most, three octal digits, but hexadecimal character constants
99285 are only terminated by a non-hex-digit character. In the ISO C standard, the "##" concatenation
99286 operator can be used to terminate a constant and follow it with a hexadecimal character to be
99287 written. In the shell, concatenation occurs before the *printf* utility has a chance to parse the end
99288 of the hexadecimal constant.

99289 The %b conversion specification is not part of the ISO C standard; it has been added here as a

99290 portable way to process backslash escapes expanded in string operands as provided by the *echo*
 99291 utility. See also the APPLICATION USAGE section of *echo* (on page 2550) for ways to use *printf*
 99292 as a replacement for all of the traditional versions of the *echo* utility.

99293 If an argument cannot be parsed correctly for the corresponding conversion specification, the
 99294 *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end
 99295 of an argument being used for a numeric conversion shall be reported as errors.

99296 EXAMPLES

99297 To alert the user and then print and read a series of prompts:

```
99298 printf "\aPlease fill in the following: \nName: "  
99299 read name  
99300 printf "Phone number: "  
99301 read phone
```

99302 To read out a list of right and wrong answers from a file, calculate the percentage correctly, and
 99303 print them out. The numbers are right-justified and separated by a single <tab>. The percentage
 99304 is written to one decimal place of accuracy:

```
99305 while read right wrong ; do  
99306     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)  
99307     printf "%2d right\t%2d wrong\t(%s%%)\n" \  
99308         $right $wrong $percent  
99309 done < database_file
```

99310 The command:

```
99311 printf "%5d%4d\n" 1 21 321 4321 54321
```

99312 produces:

```
99313     1  21  
99314     3214321  
99315 54321  0
```

99316 Note that the *format* operand is used three times to print all of the given strings and that a '0'
 99317 was supplied by *printf* to satisfy the last %4d conversion specification.

99318 The *printf* utility is required to notify the user when conversion errors are detected while
 99319 producing numeric output; thus, the following results would be expected on an implementation
 99320 with 32-bit twos-complement integers when %d is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
5a	5	printf: "5a" not completely converted
9999999999	2147483647	printf: "9999999999" arithmetic overflow
-9999999999	-2147483648	printf: "-9999999999" arithmetic overflow
ABC	0	printf: "ABC" expected numeric value

99327 The diagnostic message format is not specified, but these examples convey the type of
 99328 information that should be reported. Note that the value shown on standard output is what
 99329 would be expected as the return value from the *strtol()* function as defined in the System
 99330 Interfaces volume of POSIX.1-200x. A similar correspondence exists between %u and *strtoul()*
 99331 and %e, %f, and %g (if the implementation supports floating-point conversions) and *strtod()*.

99332 In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

```
99333 printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"
```

99334 produces:

- 99335 3 Numeric value of constant 3
- 99336 3 Numeric value of constant 3
- 99337 -3 Numeric value of constant -3
- 99338 51 Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset
- 99339 43 Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset
- 99340 45 Numeric value of the character '-' in the ISO/IEC 646:1991 standard codeset

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the **wchar_t** representation of the character as described in the System Interfaces volume of POSIX.1-200x.

RATIONALE

The *printf* utility was added to provide functionality that has historically been provided by *echo*. However, due to irreconcilable differences in the various versions of *echo* extant, the version has few special features, leaving those to this new *printf* utility, which is based on one in the Ninth Edition system.

The EXTENDED DESCRIPTION section almost exactly matches the *printf()* function in the ISO C standard, although it is described in terms of the file format notation in XBD Chapter 5 (on page 107).

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *bc*, *echo*

XBD Chapter 5 (on page 107), Chapter 8 (on page 159)

XSH *printf*

CHANGE HISTORY

First released in Issue 4.

Issue 7

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

99362 **NAME**
 99363 prs — print an SCCS file (**DEVELOPMENT**)

99364 **SYNOPSIS**

```
99365 XSI prs [-a] [-d dataspec] [-r[SID]] file...
99366 prs [-e|-l] -c cutoff [-d dataspec] file...
99367 prs [-e|-l] -r[SID] [-d dataspec] file...
```

99368 **DESCRIPTION**

99369 The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied
 99370 format.

99371 **OPTIONS**

99372 The *prs* utility shall conform to XBD Section 12.2 (on page 201), except that the **-r** option has an
 99373 optional option-argument. This optional option-argument cannot be presented as a separate
 99374 argument. The following options shall be supported:

99375 **-d *dataspec*** Specify the output data specification. The *dataspec* shall be a string consisting of
 99376 SCCS file *data keywords* (see [Data Keywords](#), on page 2972) interspersed with
 99377 optional user-supplied text.

99378 **-r[*SID*]** Specify the SCCS identification string (SID) of a delta for which information is
 99379 desired. If no *SID* option-argument is specified, the SID of the most recently
 99380 created delta shall be assumed.

99381 **-e** Request information for all deltas created earlier than and including the delta
 99382 designated via the **-r** option or the date-time given by the **-c** option.

99383 **-l** Request information for all deltas created later than and including the delta
 99384 designated via the **-r** option or the date-time given by the **-c** option.

99385 **-c *cutoff*** Indicate the *cutoff* date-time, in the form:

99386 `YY[MM[DD[HH[MM[SS]]]]]`

99387 For the YY component, values in the range [69,99] shall refer to years 1969 to 1999
 99388 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

99389 **Note:** It is expected that in a future version of this standard the default century inferred
 99390 from a 2-digit year will change. (This would apply to all commands accepting a
 99391 2-digit year as input.)

99392 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
 99393 date-time shall be included in the output. Units omitted from the date-time default
 99394 to their maximum possible values; for example, **-c 7502** is equivalent to
 99395 **-c 750228235959**.

99396 **-a** Request writing of information for both removed—that is, *delta type=R* (see
 99397 *rm~~del~~*)—and existing—that is, *delta type=D_r*—deltas. If the **-a** option is not
 99398 specified, information for existing deltas only shall be provided.

99399 **OPERANDS**

99400 The following operand shall be supported:

99401 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs*
 99402 utility shall behave as though each file in the directory were specified as a named
 99403 file, except that non-SCCS files (last component of the pathname does not begin
 99404 with **s.**) and unreadable files shall be silently ignored.

99405 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;
 99406 each line of the standard input shall be taken to be the name of an SCCS file to be
 99407 processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

99408
 99409 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each
 99410 line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

99411
 99412 Any SCCS files displayed are files of an unspecified format.

ENVIRONMENT VARIABLES

99413
 99414 The following environment variables shall affect the execution of *prs*:

99415 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 99416 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 99417 used to determine the values of locale categories.)

99418 *LC_ALL* If set to a non-empty string value, override the values of all the other
 99419 internationalization variables.

99420 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 99421 characters (for example, single-byte as opposed to multi-byte characters in
 99422 arguments and input files).

99423 *LC_MESSAGES*
 99424 Determine the locale that should be used to affect the format and contents of
 99425 diagnostic messages written to standard error.

99426 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

99427
 99428 Default.

STDOUT

99429
 99430 The standard output shall be a text file whose format is dependent on the data keywords
 99431 specified with the *-d* option.

Data Keywords

99432
 99433 Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an
 99434 SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple
 99435 times.

99436 The information written by *prs* shall consist of:

- 99437 1. The user-supplied text
- 99438 2. Appropriate values (extracted from the SCCS file) substituted for the recognized data
 99439 keywords in the order of appearance in the *dataspec*

99440 The format of a data keyword value shall either be simple (*'S'*), in which keyword substitution
 99441 is direct, or multi-line (*'M'*).

99442 User-supplied text shall be any text other than recognized data keywords. A <tab> shall be
 99443 specified by *'\t'* and <newline> by *'\n'*. When the *-r* option is not specified, the default
 99444 *dataspec* shall be:

99445 *:PN::\n\n*

99446 and the following *dataspec* shall be used for each selected delta:

99447 *:Dt:\t:DL:\nMRS:\n:MR:COMMENTS:\n:C:*

99448

99449

99450

99451

99452

99453

99454

99455

99456

99457

99458

99459

99460

99461

99462

99463

99464

99465

99466

99467

99468

99469

99470

99471

99472

99473

99474

99475

99476

99477

99478

99479

99480

99481

99482

99483

99484

99485

99486

99487

99488

99489

99490

99491

99492

99493

99494

99495

99496

99497

99498

99499

99500

SCCS File Data Keywords				
Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn***	S
:Ld:	Lines deleted by Delta	"	nnnnn***	S
:Lu:	Lines unchanged by Delta	"	nnnnn***	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	See below**	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year delta created	"	nn	S
:Dm:	Month delta created	"	nn	S
:Dd:	Day delta created	"	nn	S
:T:	Time delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour delta created	"	nn	S
:Tm:	Minutes delta created	"	nn	S
:Ts:	Seconds delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta sequence number	"	nnnn	S
:DI:	Sequence number of deltas included, excluded, or ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (sequence #)	"	:DS: :DS: ...	S
:Dx:	Deltas excluded (sequence #)	"	:DS: :DS: ...	S
:Dg:	Deltas ignored (sequence #)	"	:DS: :DS: ...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation program name	"	text	S
:KF:	Keyword error, warning flag	"	yes or no	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of <i>what</i> string	N/A	:Z::M:\t:I:	S
:A:	A form of <i>what</i> string	N/A	:Z::Y: :M: :I::Z:	S

99501
99502
99503
99504
99505
99506
99507
99508
99509
99510
99511
99512
99513
99514
99515
99516
99517
99518
99519
99520
99521
99522
99523
99524
99525
99526
99527
99528
99529
99530
99531
99532
99533
99534
99535
99536
99537
99538
99539
99540
99541
99542

SCCS File Data Keywords				
Keyword	Data Item	File Section	Value	Format
:Z:	<i>what</i> string delimiter	N/A	@ (#)	S
:F:	SCCS filename	N/A	<i>text</i>	S
:PN:	SCCS file pathname	N/A	<i>text</i>	S

* **:Dt:=:DT: :I: :D: :T: :P: :DS: :DP:**

** **:R::L::B::S:** if the delta is a branch delta (**:BF:=yes**)

:R::L: if the delta is not a branch delta (**:BF:=no**)

*** The line statistics are capped at 99999. For example, if 100000 lines were unchanged in a certain revision, **:Lu:** shall produce the value 99999.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. The following example:

```
prs -d "User Names for :F: are:\n:UN:" s.file
```

might write to standard output:

```
User Names for s.file are:
```

```
xyz
```

```
131
```

```
abc
```

2. The following example:

```
prs -d "Delta for pgm :M:: :I: - :D: By :P:" -r s.file
```

might write to standard output:

```
Delta for pgm main.c: 3.7 - 77/12/01 By cas
```

3. As a special case:

```
prs s.file
```

might write to standard output:

```
s.file:
```

```
<blank line>
```

```
D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
```

99543 MRS :
 99544 b178-12345
 99545 b179-54321
 99546 COMMENTS :
 99547 this is the comment line for s.file initial delta
 99548 <blank line>

99549 for each delta table entry of the **D** type. The only option allowed to be used with this
 99550 special case is the **-a** option.

RATIONALE

99551 None.
 99552

FUTURE DIRECTIONS

99553 None.
 99554

SEE ALSO

99555 *admin, delta, get, what*

99556 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

CHANGE HISTORY

99558 First released in Issue 2.
 99559

Issue 5

99560 The phrase “in which keyword substitution is followed by a <newline>” is deleted from the end
 99561 of the second paragraph of [Data Keywords](#) (on page 2972).
 99562

99563 The interpretation of the *YY* component of the **-c cutoff** argument is noted.

Issue 6

99564 The normative text is reworded to emphasize the term “shall” for implementation requirements.
 99565

99566 The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a
 99567 note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.

99568 The Open Group Interpretation PIN4C.00009 is applied.

Issue 7

99569 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
 99570

99571 NAME

99572 ps — report process status

99573 SYNOPSIS

```
99574 XSI ps [-aA] [-defl] [-g grouplist] [-G grouplist]
99575 [-n namelist] [-o format]... [-p proclist] [-t termlist]
99576 [-u userlist] [-U userlist]
```

99577 DESCRIPTION

99578 The *ps* utility shall write information about processes, subject to having the appropriate
 99579 privileges to obtain information about those processes.

99580 By default, *ps* shall select all processes with the same effective user ID as the current user and the
 99581 same controlling terminal as the invoker.

99582 OPTIONS

99583 The *ps* utility shall conform to XBD Section 12.2 (on page 201).

99584 The following options shall be supported:

99585 **-a** Write information for all processes associated with terminals. Implementations
 99586 may omit session leaders from this list.

99587 **-A** Write information for all processes.

99588 XSI **-d** Write information for all processes, except session leaders.

99589 XSI **-e** Write information for all processes. (Equivalent to **-A**.)

99590 XSI **-f** Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)

99591 XSI **-g grouplist** Write information for processes whose session leaders are given in *grouplist*. The
 99592 application shall ensure that the *grouplist* is a single argument in the form of a
 99593 <blank> or comma-separated list.

99594 **-G grouplist** Write information for processes whose real group ID numbers are given in
 99595 *grouplist*. The application shall ensure that the *grouplist* is a single argument in the
 99596 form of a <blank> or comma-separated list.

99597 XSI **-l** Generate a **long** listing. (See STDOUT for the contents of a **long** listing.)

99598 XSI **-n namelist** Specify the name of an alternative system *namelist* file in place of the default. The
 99599 name of the default file and the format of a *namelist* file are unspecified.

99600 **-o format** Write information according to the format specification given in *format*. This is
 99601 fully described in the STDOUT section. Multiple **-o** options can be specified; the
 99602 format specification shall be interpreted as the <space>-separated concatenation of
 99603 all the *format* option-arguments.

99604 **-p proclist** Write information for processes whose process ID numbers are given in *proclist*.
 99605 The application shall ensure that the *proclist* is a single argument in the form of a
 99606 <blank> or comma-separated list.

99607 **-t termlist** Write information for processes associated with terminals given in *termlist*. The
 99608 application shall ensure that the *termlist* is a single argument in the form of a
 99609 <blank> or comma-separated list. Terminal identifiers shall be given in an
 99610 implementation-defined format. On XSI-conformant systems, they shall be given
 99611 in one of two forms: the device's filename (for example, **tty04**) or, if the device's
 99612 filename starts with **tty**, just the identifier following the characters **tty** (for example,
 99613 "04").

99614 XSI **-u *userlist*** Write information for processes whose user ID numbers or login names are given
 99615 in *userlist*. The application shall ensure that the *userlist* is a single argument in the
 99616 form of a <blank> or comma-separated list. In the listing, the numerical user ID
 99617 shall be written unless the **-f** option is used, in which case the login name shall be
 99618 written.

99619 **-U *userlist*** Write information for processes whose real user ID numbers or login names are
 99620 given in *userlist*. The application shall ensure that the *userlist* is a single argument
 99621 in the form of a <blank> or comma-separated list.

99622 With the exception of **-o *format***, all of the options shown are used to select processes. If any are
 99623 specified, the default list shall be ignored and *ps* shall select the processes represented by the
 99624 inclusive OR of all the selection-criteria options.

99625 OPERANDS

99626 None.

99627 STDIN

99628 Not used.

99629 INPUT FILES

99630 None.

99631 ENVIRONMENT VARIABLES

99632 The following environment variables shall affect the execution of *ps*:

99633 **COLUMNS** Override the system-selected horizontal display line size, used to determine the |
 99634 number of text columns to display. See XBD [Chapter 8](#) (on page 159) for valid |
 99635 values and results when it is unset or null.

99636 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 99637 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables |
 99638 used to determine the values of locale categories.)

99639 **LC_ALL** If set to a non-empty string value, override the values of all the other
 99640 internationalization variables.

99641 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 99642 characters (for example, single-byte as opposed to multi-byte characters in
 99643 arguments).

99644 **LC_MESSAGES**
 99645 Determine the locale that should be used to affect the format and contents of
 99646 diagnostic messages written to standard error and informative messages written to
 99647 standard output.

99648 **LC_TIME** Determine the format and contents of the date and time strings displayed.

99649 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

99650 **TZ** Determine the timezone used to calculate date and time strings displayed. If **TZ** is
 99651 unset or null, an unspecified default timezone shall be used.

99652 ASYNCHRONOUS EVENTS

99653 Default.

99654 STDOUT

99655 When the **-o** option is not specified, the standard output format is unspecified.

99656	XSI	On XSI-conformant systems, the output format shall be as follows. The column headings and descriptions of the columns in a <i>ps</i> listing are given below. The precise meanings of these fields are implementation-defined. The letters 'f' and 'l' (below) indicate the option (full or long) that shall cause the corresponding heading to appear; all means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes are listed.
99657		
99658		
99659		
99660		
99661		
99662		F (l) Flags (octal and additive) associated with the process.
99663		S (l) The state of the process.
99664		UID (f,l) The user ID number of the process owner; the login name is printed under the -f option.
99665		
99666		PID (all) The process ID of the process; it is possible to kill a process if this datum is known.
99667		
99668		PPID (f,l) The process ID of the parent process.
99669		C (f,l) Processor utilization for scheduling.
99670		PRI (l) The priority of the process; higher numbers mean lower priority.
99671		NI (l) Nice value; used in priority computation.
99672		ADDR (l) The address of the process.
99673		SZ (l) The size in blocks of the core image of the process.
99674		WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.
99675		
99676		STIME (f) Starting time of the process.
99677		TTY (all) The controlling terminal for the process.
99678		TIME (all) The cumulative execution time for the process.
99679		CMD (all) The command name; the full command name and its arguments are written under the -f option.
99680		
99681		A process that has exited and has a parent, but has not yet been waited for by the parent, shall be marked defunct .
99682		
99683		Under the option -f , <i>ps</i> tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the option -f , is written in square brackets.
99684		
99685		
99686		The -o option allows the output format to be specified under user control.
99687		The application shall ensure that the format specification is a list of names presented as a single argument, <blank> or comma-separated. Each variable has a default header. The default header can be overridden by appending an equals sign and the new text of the header. The rest of the characters in the argument shall be used as the header text. The fields specified shall be written in the order specified on the command line, and should be arranged in columns in the output. The field widths shall be selected by the system to be at least as wide as the header text (default or overridden value). If the header text is null, such as -o user= , the field width shall be at least as wide as the default header text. If all header text fields are null, no header line shall be written.
99688		
99689		
99690		
99691		
99692		
99693		
99694		
99695		
99696		The following names are recognized in the POSIX locale:
99697		ruser The real user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
99698		
99699		user The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
99700		
99701		rgroup The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
99702		

99703	group	The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
99704		
99705	pid	The decimal value of the process ID.
99706	ppid	The decimal value of the parent process ID.
99707	pgid	The decimal value of the process group ID.
99708	pcpu	The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of “recently” in this context is unspecified. The CPU time available is determined in an unspecified manner.
99709		
99710		
99711	vsz	The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.
99712	nice	The decimal value of the nice value of the process; see <i>nice</i> .
99713	etime	In the POSIX locale, the elapsed time since the process was started, in the form:
99714		[[<i>dd</i> -] <i>hh</i> :] <i>mm</i> : <i>ss</i>
99715		where <i>dd</i> shall represent the number of days, <i>hh</i> the number of hours, <i>mm</i> the number of minutes, and <i>ss</i> the number of seconds. The <i>dd</i> field shall be a decimal integer. The <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be two-digit decimal integers padded on the left with zeros.
99716		
99717		
99718	time	In the POSIX locale, the cumulative CPU time of the process in the form:
99719		[[<i>dd</i> -] <i>hh</i> : <i>mm</i> :] <i>ss</i>
99720		The <i>dd</i> , <i>hh</i> , <i>mm</i> , and <i>ss</i> fields shall be as described in the etime specifier.
99721	tty	The name of the controlling terminal of the process (if any) in the same format used by the <i>who</i> utility.
99722		
99723	comm	The name of the command being executed (<i>argv</i> [0] value) as a string.
99724	args	The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation-defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of <i>ps</i> .
99725		
99726		
99727		
99728		
99729		
99730		
99731		Any field need not be meaningful in all implementations. In such a case a hyphen (‘-’) should be output in place of the field value.
99732		
99733		Only comm and args shall be allowed to contain <blank>s; all others shall not. Any implementation-defined variables shall be specified in the system documentation along with the default header and indicating whether the field may contain <blank>s.
99734		
99735		
99736		The following table specifies the default header to be used in the POSIX locale corresponding to each format specifier.
99737		

Table 4-17 Variable Names and Default Headers in *ps*

Format Specifier	Default Header	Format Specifier	Default Header
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ
pid	PID		

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Things can change while *ps* is running; the snapshot it gives is only true for an instant, and might not be accurate by the time it is displayed.

The **args** format specifier is allowed to produce a truncated version of the command arguments. In some implementations, this information is no longer available when the *ps* utility is executed.

If the field width is too narrow to display a textual ID, the system may use a numeric version. Normally, the system would be expected to choose large enough field widths, but if a large number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on one line. One way to ensure adequate width for the textual IDs is to override the default header for a field to make it larger than most or all user or group names.

There is no special quoting mechanism for header text. The header text is the rest of the argument. If multiple header changes are needed, multiple **-o** options can be used, such as:

```
ps -o "user=User Name" -o pid=Process\ ID
```

On some implementations, especially multi-level secure systems, *ps* may be severely restricted and produce information only about child processes owned by the user.

EXAMPLES

The command:

```
ps -o user,pid,ppid=MOM -o args
```

writes at least the following in the POSIX locale:

```
USER    PID    MOM    COMMAND
helene  34     12    ps -o uid,pid,ppid=MOM -o args
```

The contents of the **COMMAND** field need not be the same in all implementations, due to

99782 possible truncation.

99783 RATIONALE

99784 There is very little commonality between BSD and System V implementations of *ps*. Many
 99785 options conflict or have subtly different usages. The standard developers attempted to select a
 99786 set of options for the base standard that were useful on a wide range of systems and selected
 99787 options that either can be implemented on both BSD and System V-based systems without
 99788 breaking the current implementations or where the options are sufficiently similar that any
 99789 changes would not be unduly problematic for users or implementors.

99790 It is recognized that on some implementations, especially multi-level secure systems, *ps* may be
 99791 nearly useless. The default output has therefore been chosen such that it does not break
 99792 historical implementations and also is likely to provide at least some useful information on most
 99793 systems.

99794 The major change is the addition of the format specification capability. The motivation for this
 99795 invention is to provide a mechanism for users to access a wider range of system information, if
 99796 the system permits it, in a portable manner. The fields chosen to appear in this volume of
 99797 POSIX.1-200x were arrived at after considering what concepts were likely to be both reasonably
 99798 useful to the “average” user and had a reasonable chance of being implemented on a wide range
 99799 of systems. Again it is recognized that not all systems are able to provide all the information
 99800 and, conversely, some may wish to provide more. It is hoped that the approach adopted will be
 99801 sufficiently flexible and extensible to accommodate most systems. Implementations may be
 99802 expected to introduce new format specifiers.

99803 The default output should consist of a short listing containing the process ID, terminal name,
 99804 cumulative execution time, and command name of each process.

99805 The preference of the standard developers would have been to make the format specification an
 99806 operand of the *ps* command. Unfortunately, BSD usage precluded this.

99807 At one time a format was included to display the environment array of the process. This was
 99808 deleted because there is no portable way to display it.

99809 The **-A** option is equivalent to the BSD **-g** and the SVID **-e**. Because the two systems differed, a
 99810 mnemonic compromise was selected.

99811 The **-a** option is described with some optional behavior because the SVID omits session leaders,
 99812 but BSD does not.

99813 In an early proposal, format specifiers appeared for priority and start time. The former was not
 99814 defined adequately in this volume of POSIX.1-200x and was removed in deference to the defined
 99815 nice value; the latter because elapsed time was considered to be more useful.

99816 In a new BSD version of *ps*, a **-O** option can be used to write all of the default information,
 99817 followed by additional format specifiers. This was not adopted because the default output is
 99818 implementation-defined. Nevertheless, this is a useful option that should be reserved for that
 99819 purpose. In the **-o** option for the POSIX Shell and Utilities *ps*, the format is the concatenation of
 99820 each **-o**. Therefore, the user can have an alias or function that defines the beginning of their
 99821 desired format and add more fields to the end of the output in certain cases where that would be
 99822 useful.

99823 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
 99824 require that they all use the same format.

99825 The **pcpu** field indicates that the CPU time available is determined in an unspecified manner.
 99826 This is because it is difficult to express an algorithm that is useful across all possible machine
 99827 architectures. Historical counterparts to this value have attempted to show percentage of use in
 99828 the recent past, such as the preceding minute. Frequently, these values for all processes did not
 99829 add up to 100%. Implementations are encouraged to provide data in this field to users that will

99830 help them identify processes currently affecting the performance of the system.

99831 **FUTURE DIRECTIONS**

99832 None.

99833 **SEE ALSO**

99834 *kill, nice, renice*

99835 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

99836 **CHANGE HISTORY**

99837 First released in Issue 2.

99838 **Issue 6**

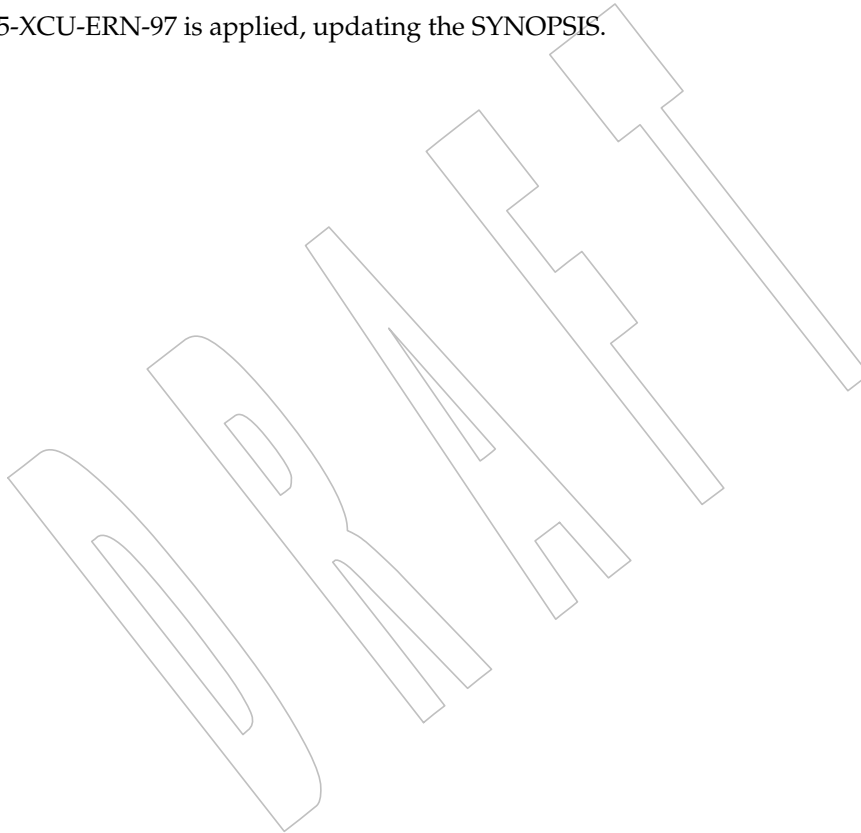
99839 This utility is marked as part of the User Portability Utilities option.

99840 The normative text is reworded to avoid use of the term “must” for application requirements.

99841 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

99842 **Issue 7**

99843 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



99844 **NAME**
 99845 `pwd` — return working directory name

99846 **SYNOPSIS**
 99847 `pwd [-L|-P]`

99848 **DESCRIPTION**
 99849 The *pwd* utility shall write to standard output an absolute pathname of the current working
 99850 directory, which does not contain the filenames dot or dot-dot.

99851 **OPTIONS**
 99852 The *pwd* utility shall conform to XBD [Section 12.2](#) (on page 201).

99853 The following options shall be supported by the implementation:

99854 **-L** If the *PWD* environment variable contains an absolute pathname of the current
 99855 directory that does not contain the filenames dot or dot-dot, *pwd* shall write this
 99856 pathname to standard output. Otherwise, the **-L** option shall behave as the **-P**
 99857 option.

99858 **-P** The absolute pathname written shall not contain filenames that, in the context of
 99859 the pathname, refer to files of type symbolic link.

99860 If both **-L** and **-P** are specified, the last one shall apply. If neither **-L** nor **-P** is specified, the *pwd*
 99861 utility shall behave as if **-L** had been specified.

99862 **OPERANDS**
 99863 None.

99864 **STDIN**
 99865 Not used.

99866 **INPUT FILES**
 99867 None.

99868 **ENVIRONMENT VARIABLES**
 99869 The following environment variables shall affect the execution of *pwd*:

99870 **LANG** Provide a default value for the internationalization variables that are unset or null.
 99871 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables
 99872 used to determine the values of locale categories.)

99873 **LC_ALL** If set to a non-empty string value, override the values of all the other
 99874 internationalization variables.

99875 **LC_MESSAGES**
 99876 Determine the locale that should be used to affect the format and contents of
 99877 diagnostic messages written to standard error.

99878 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

99879 **PWD** An absolute pathname of the current working directory. If an application sets or
 99880 unsets the value of *PWD*, the behavior of *pwd* is unspecified.

99881 **ASYNCHRONOUS EVENTS**
 99882 Default.

99883 STDOUT

99884 The *pwd* utility output is an absolute pathname of the current working directory:

99885 "%s\n", <directory pathname>

99886 STDERR

99887 The standard error shall be used only for diagnostic messages.

99888 OUTPUT FILES

99889 None.

99890 EXTENDED DESCRIPTION

99891 None.

99892 EXIT STATUS

99893 The following exit values shall be returned:

99894 0 Successful completion.

99895 >0 An error occurred.

99896 CONSEQUENCES OF ERRORS

99897 If an error is detected, output shall not be written to standard output, a diagnostic message shall
99898 be written to standard error, and the exit status is not zero.

99899 APPLICATION USAGE

99900 None.

99901 EXAMPLES

99902 None.

99903 RATIONALE

99904 Some implementations have historically provided *pwd* as a shell special built-in command.

99905 In most utilities, if an error occurs, partial output may be written to standard output. This does
99906 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical
99907 shell scripts without checking the exit status, it is important that the historical behavior is
99908 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any
99909 partial output being written to standard output.

99910 An earlier version of this standard stated that the *PWD* environment variable was affected when
99911 the *-P* option was in effect. This was incorrect; conforming implementations do not do this.

99912 FUTURE DIRECTIONS

99913 None.

99914 SEE ALSO

99915 *cd*

99916 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

99917 XSH *getcwd()*

99918 CHANGE HISTORY

99919 First released in Issue 2.

99920 Issue 6

99921 The *-P* and *-L* options are added to describe actions relating to symbolic links as specified in the
99922 IEEE P1003.2b draft standard.

99923

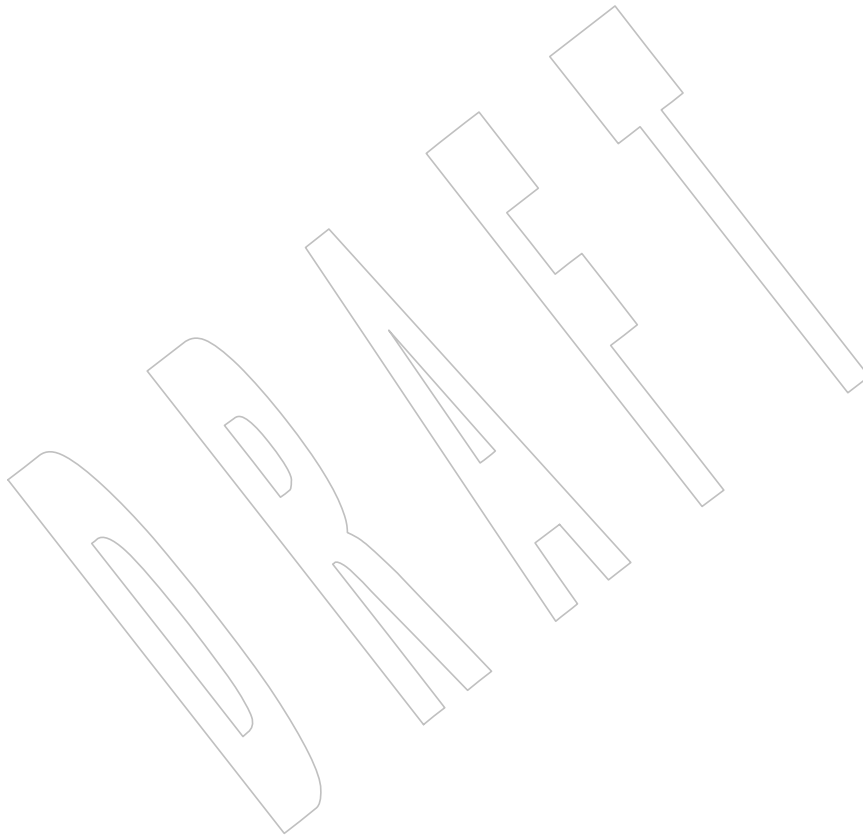
99924

99925

Issue 7

Austin Group Interpretation 1003.1-2001 #097 is applied.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



99926 NAME

99927 qalter — alter batch job

99928 SYNOPSIS

```
99929 OB BE qalter [-a date_time] [-A account_string] [-c interval] [-e path_name]
99930 [-h hold_list] [-j join_list] [-k keep_list] [-l resource_list]
99931 [-m mail_options] [-M mail_list] [-N name] [-o path_name]
99932 [-p priority] [-r y|n] [-S path_name_list] [-u user_list]
99933 job_identifier...
```

99934 DESCRIPTION

99935 The attributes of a batch job are altered by a request to the batch server that manages the batch
 99936 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes
 99937 of one or more batch jobs.

99938 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which
 99939 a batch *job_identifier* is presented to the utility.

99940 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch
 99941 *job_identifiers* are presented to the utility.

99942 If the *qalter* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 99943 process the remaining batch *job_identifiers*, if any.

99944 For each batch *job_identifier* for which the *qalter* utility succeeds, each attribute of the identified
 99945 batch job shall be altered as indicated by all the options presented to the utility.

99946 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any
 99947 attribute of the batch job.

99948 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other
 99949 than those required by the options and option-arguments presented to the utility.

99950 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that
 99951 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job
 99952 corresponding to each successfully processed batch *job_identifier*. An attempt to alter the
 99953 attributes of a batch job in the RUNNING state is implementation-defined.

99954 OPTIONS

99955 The *qalter* utility shall conform to XBD [Section 12.2](#) (on page 201).

99956 The following options shall be supported by the implementation:

99957 **-a date_time** Redefine the time at which the batch job becomes eligible for execution.

99958 The *date_time* argument shall be in the same form and represent the same time as
 99959 for the *touch* utility. The time so represented shall be set into the *Execution_Time*
 99960 attribute of the batch job. If the time specified is earlier than the current time, the
 99961 **-a** option shall have no effect.

99962 **-A account_string**

99963 Redefine the account to which the resource consumption of the batch job should be
 99964 charged.

99965 The syntax of the *account_string* option-argument is unspecified.

99966 The *qalter* utility shall set the *Account_Name* attribute of the batch job to the value
 99967 of the *account_string* option-argument.

99968 **-c interval** Redefine whether the batch job should be checkpointed, and if so, how often.

99969 The *qalter* utility shall accept a value for the interval option-argument that is one of

99970 the following:

99971 n No checkpointing is to be performed on the batch job

99972 (NO_CHECKPOINT).

99973 s Checkpointing is to be performed only when the batch server is shut

99974 down (CHECKPOINT_AT_SHUTDOWN).

99975 c Automatic periodic checkpointing is to be performed at the

99976 *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU

99977 minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

99978 c=*minutes* Automatic periodic checkpointing is to be performed every *minutes*

99979 of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is

99980 greater. The *minutes* argument shall conform to the syntax for

99981 unsigned integers and shall be greater than zero.

99982 An implementation may define other checkpoint intervals. The conformance

99983 document for an implementation shall describe any alternative checkpoint

99984 intervals, how they are specified, their internal behavior, and how they affect the

99985 behavior of the utility.

99986 The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the

99987 *interval* option-argument.

99988 **-e path_name** Redefine the path to be used for the standard error stream of the batch job.

99989 The *qalter* utility shall accept a *path_name* option-argument that conforms to the

99990 syntax of the *path_name* element defined in the System Interfaces volume of

99991 POSIX.1-200x, which can be preceded by a host name element of the form

99992 *hostname:*.

99993 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*

99994 utility shall set the *Error_Path* attribute of the batch job to the value of the

99995 *path_name* option-argument, including the host name element, if present.

99996 If the *path_name* option-argument constitutes a relative pathname and no host

99997 name element is specified, the *qalter* utility shall set the *Error_Path* attribute of the

99998 batch job to the value of the absolute pathname derived by expanding the

99999 *path_name* option-argument relative to the current directory of the process that

100000 executes the *qalter* utility.

100001 If the *path_name* option-argument constitutes a relative pathname and a host name

100002 element is specified, the *qalter* utility shall set the *Error_Path* attribute of the batch

100003 job to the value of the option-argument without expansion.

100004 If the *path_name* option-argument does not include a host name element, the *qalter*

100005 utility shall prefix the pathname in the *Error_Path* attribute with *hostname:*, where

100006 *hostname* is the name of the host upon which the *qalter* utility is being executed.

100007 **-h hold_list** Redefine the types of holds, if any, on the batch job. The *qalter* **-h** option shall

100008 accept a value for the *hold_list* option-argument that is a string of alphanumeric

100009 characters in the portable character set.

100010 The *qalter* utility shall accept a value for the *hold_list* option-argument that is a

100011 string of one or more of the characters 'u', 's', or 'o', or the single character

100012 'n'. For each unique character in the *hold_list* option-argument, the *qalter* utility

100013 shall add a value to the *Hold_Types* attribute of the batch job as follows, each

100014

- 100015 representing a different hold type:
- 100016 u USER
- 100017 s SYSTEM
- 100018 o OPERATOR
- 100019 If any of these characters are duplicated in the *hold_list* option-argument, the
- 100020 duplicates shall be ignored. An existing *Hold_Types* attribute can be cleared by the
- 100021 hold type:
- 100022 n NO_HOLD
- 100023 The *qalter* utility shall consider it an error if any hold type other than 'n' is
- 100024 combined with hold type 'n'. Strictly conforming applications shall not repeat
- 100025 any of the characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument.
- 100026 The *qalter* utility shall permit the repetition of characters, but shall not assign
- 100027 additional meaning to the repeated characters. An implementation may define
- 100028 other hold types. The conformance document for an implementation shall describe
- 100029 any additional hold types, how they are specified, their internal behavior, and how
- 100030 they affect the behavior of the utility.
- 100031 **-j** *join_list* Redefine which streams of the batch job are to be merged. The *qalter* **-j** option shall
- 100032 accept a value for the *join_list* option-argument that is a string of alphanumeric
- 100033 characters in the portable character set.
- 100034 The *qalter* utility shall accept a *join_list* option-argument that consists of one or
- 100035 more of the characters 'e' and 'o', or the single character 'n'.
- 100036 All of the other batch job output streams specified shall be merged into the output
- 100037 stream represented by the character listed first in the *join_list* option-argument.
- 100038 For each unique character in the *join_list* option-argument, the *qalter* utility shall
- 100039 add a value to the *Join_Path* attribute of the batch job as follows, each representing
- 100040 a different batch job stream to join:
- 100041 e The standard error of the batch job (JOIN_STD_ERROR).
- 100042 o The standard output of the batch job (JOIN_STD_OUTPUT).
- 100043 An existing *Join_Path* attribute can be cleared by the join type:
- 100044 n NO_JOIN
- 100045 If 'n' is specified, then no files are joined. The *qalter* utility shall consider it an
- 100046 error if any join type other than 'n' is combined with join type 'n'.
- 100047 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
- 100048 'n' within the *join_list* option-argument. The *qalter* utility shall permit the
- 100049 repetition of characters, but shall not assign additional meaning to the repeated
- 100050 characters.
- 100051 An implementation may define other join types. The conformance document for an
- 100052 implementation shall describe any additional batch job streams, how they are
- 100053 specified, their internal behavior, and how they affect the behavior of the utility.
- 100054 **-k** *keep_list* Redefine which output of the batch job to retain on the execution host.
- 100055 The *qalter* **-k** option shall accept a value for the *keep_list* option-argument that is a
- 100056 string of alphanumeric characters in the portable character set.
- 100057 The *qalter* utility shall accept a *keep_list* option-argument that consists of one or
- 100058 more of the characters 'e' and 'o', or the single character 'n'.

For each unique character in the *keep_list* option-argument, the *qalter* utility shall add a value to the *Keep_Files* attribute of the batch job as follows, each representing a different batch job stream to keep:

e The standard error of the batch job (KEEP_STD_ERROR).

o The standard output of the batch job (KEEP_STD_OUTPUT).

If both 'e' and 'o' are specified, then both files are retained. An existing *Keep_Files* attribute can be cleared by the keep type:

n NO_KEEP

If 'n' is specified, then no files are retained. The *qalter* utility shall consider it an error if any keep type other than 'n' is combined with keep type 'n'.

Strictly conforming applications shall not repeat any of the characters 'e', 'o', or 'n' within the *keep_list* option-argument. The *qalter* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters. An implementation may define other keep types. The conformance document for an implementation shall describe any additional keep types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

-l *resource_list*

Redefine the resources that are allowed or required by the batch job.

The *qalter* utility shall accept a *resource_list* option-argument that conforms to the following syntax:

```
resource=value[ , resource=value, , ... ]
```

The *qalter* utility shall set one entry in the value of the *Resource_List* attribute of the batch job for each resource listed in the *resource_list* option-argument.

Because the list of supported resource names might vary by batch server, the *qalter* utility shall rely on the batch server to validate the resource names and associated values. See [Section 3.3.3](#) (on page 2341) for a means of removing *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

-m *mail_options*

Redefine the points in the execution of the batch job at which the batch server is to send mail about a change in the state of the batch job.

The *qalter* **-m** option shall accept a value for the *mail_options* option-argument that is a string of alphanumeric characters in the portable character set.

The *qalter* utility shall accept a value for the *mail_options* option-argument that is a string of one or more of the characters 'e', 'b', and 'a', or the single character 'n'. For each unique character in the *mail_options* option-argument, the *qalter* utility shall add a value to the *Mail_Users* attribute of the batch job as follows, each representing a different time during the life of a batch job at which to send mail:

e MAIL_AT_EXIT

b MAIL_AT_BEGINNING

a MAIL_AT_ABORT

If any of these characters are duplicated in the *mail_options* option-argument, the duplicates shall be ignored.

- 100103 An existing *Mail_Points* attribute can be cleared by the mail type:
- 100104 n NO_MAIL
- 100105 If 'n' is specified, then mail is not sent. The *qalter* utility shall consider it an error
100106 if any mail type other than 'n' is combined with mail type 'n'. Strictly
100107 conforming applications shall not repeat any of the characters 'e', 'b', 'a', or
100108 'n' within the *mail_options* option-argument. The *qalter* utility shall permit the
100109 repetition of characters but shall not assign additional meaning to the repeated
100110 characters.
- 100111 An implementation may define other mail types. The conformance document for
100112 an implementation shall describe any additional mail types, how they are
100113 specified, their internal behavior, and how they affect the behavior of the utility.
- 100114 **-M** *mail_list* Redefine the list of users to which the batch server that executes the batch job is to
100115 send mail, if the batch server sends mail about the batch job.
- 100116 The syntax of the *mail_list* option-argument is unspecified. If the implementation
100117 of the *qalter* utility uses a name service to locate users, the utility shall accept the
100118 syntax used by the name service.
- 100119 If the implementation of the *qalter* utility does not use a name service to locate
100120 users, the implementation shall accept the following syntax for user names:
- 100121 `mail_address[, mail_address , ...]`
- 100122 The interpretation of *mail_address* is implementation-defined.
- 100123 The *qalter* utility shall set the *Mail_Users* attribute of the batch job to the value of
100124 the *mail_list* option-argument.
- 100125 **-N** *name* Redefine the name of the batch job.
- 100126 The *qalter* **-N** option shall accept a value for the *name* option-argument that is a
100127 string of up to 15 alphanumeric characters in the portable character set where the
100128 first character is alphabetic.
- 100129 The syntax of the *name* option-argument is unspecified.
- 100130 The *qalter* utility shall set the *Job_Name* attribute of the batch job to the value of the
100131 *name* option-argument.
- 100132 **-o** *path_name* Redefine the path for the standard output of the batch job.
- 100133 The *qalter* utility shall accept a *path_name* option-argument that conforms to the
100134 syntax of the *path_name* element defined in the System Interfaces volume of
100135 POSIX.1-200x, which can be preceded by a host name element of the form
100136 *hostname*:.
100137
- 100138 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*
100139 utility shall set the *Output_Path* attribute of the batch job to the value of the
100140 *path_name* option-argument.
- 100141 If the *path_name* option-argument constitutes a relative pathname and no host
100142 name element is specified, the *qalter* utility shall set the *Output_Path* attribute of the
100143 batch job to the absolute pathname derived by expanding the *path_name* option-
100144 argument relative to the current directory of the process that executes the *qalter*
100145 utility.
- 100146 If the *path_name* option-argument constitutes a relative pathname and a host name
100147 element is specified, the *qalter* utility shall set the *Output_Path* attribute of the batch

- 100148 job to the value of the *path_name* option-argument without any expansion of the
100149 pathname.
- 100150 If the *path_name* option-argument does not include a host name element, the *qalter*
100151 utility shall prefix the pathname in the *Output_Path* attribute with *hostname:*, where
100152 *hostname* is the name of the host upon which the *qalter* utility is being executed.
- 100153 **-p priority** Redefine the priority of the batch job.
- 100154 The *qalter* utility shall accept a value for the priority option-argument that
100155 conforms to the syntax for signed decimal integers, and which is not less than
100156 $-1\ 024$ and not greater than $1\ 023$.
- 100157 The *qalter* utility shall set the *Priority* attribute of the batch job to the value of the
100158 *priority* option-argument.
- 100159 **-r y | n** Redefine whether the batch job is rerunnable.
- 100160 If the value of the option-argument is 'y', the *qalter* utility shall set the *Rerunable*
100161 attribute of the batch job to TRUE.
- 100162 If the value of the option-argument is 'n', the *qalter* utility shall set the *Rerunable*
100163 attribute of the batch job to FALSE.
- 100164 The *qalter* utility shall consider it an error if any character other than 'y' or 'n' is
100165 specified in the option-argument.
- 100166 **-S path_name_list**
- 100167 Redefine the shell that interprets the script at the destination system.
- 100168 The *qalter* utility shall accept a *path_name_list* option-argument that conforms to the
100169 following syntax:
- 100170 `pathname[@host][,pathname[@host],...]`
- 100171 The *qalter* utility shall accept only one pathname that is missing a corresponding
100172 host name. The *qalter* utility shall allow only one pathname per named host.
- 100173 The *qalter* utility shall add a value to the *Shell_Path_List* attribute of the batch job
100174 for each entry in the *path_name_list* option-argument. See [Section 3.3.3](#) (on page
100175 2341) for a means of removing *keyword=value* (and *value@keyword*) pairs and other
100176 general rules for list-oriented batch job attributes.
- 100177 **-u user_list** Redefine the user name under which the batch job is to run at the destination
100178 system.
- 100179 The *qalter* utility shall accept a *user_list* option-argument that conforms to the
100180 following syntax:
- 100181 `username[@host][,username[@host],...]`
- 100182 The *qalter* utility shall accept only one user name that is missing a corresponding
100183 host name. The *qalter* utility shall accept only one user name per named host.
- 100184 The *qalter* utility shall add a value to the *User_List* attribute of the batch job for each
100185 entry in the *user_list* option-argument. See [Section 3.3.3](#) (on page 2341) for a means
100186 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for
100187 list-oriented batch job attributes.
- 100188 **OPERANDS**
- 100189 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch
100190 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100191 **STDIN**

100192 Not used.

100193 **INPUT FILES**

100194 None.

100195 **ENVIRONMENT VARIABLES**100196 The following environment variables shall affect the execution of *qalter*:

100197 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100198 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 100199 used to determine the values of locale categories.)

100200 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100201 internationalization variables.

100202 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100203 characters (for example, single-byte as opposed to multi-byte characters in
 100204 arguments).

100205 *LC_MESSAGES*
 100206 Determine the locale that should be used to affect the format and contents of
 100207 diagnostic messages written to standard error.

100208 *LOGNAME* Determine the login name of the user.

100209 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
 100210 unset or null, an unspecified default timezone shall be used.

100211 **ASYNCHRONOUS EVENTS**

100212 Default.

100213 **STDOUT**

100214 None.

100215 **STDERR**

100216 The standard error shall be used only for diagnostic messages.

100217 **OUTPUT FILES**

100218 None.

100219 **EXTENDED DESCRIPTION**

100220 None.

100221 **EXIT STATUS**

100222 The following exit values shall be returned:

100223 0 Successful completion.

100224 >0 An error occurred.

100225 **CONSEQUENCES OF ERRORS**

100226 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic
 100227 message to standard error when the error reply received from a batch server indicates that the
 100228 batch *job_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to locate
 100229 the batch job on other batch servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qalter* utility allows users to change the attributes of a batch job.

As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the batch job insofar as an altered job retains its place in the queue with some traditional selection algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and *qsub* utilities.

The result of an attempt on the part of a user to alter a batch job in a RUNNING state is implementation-defined because a batch job in the RUNNING state will already have opened its output files and otherwise performed any actions indicated by the options in effect at the time the batch job began execution.

The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few exceptions: **-V**, **-v**, and **-q**. The **-V** and **-v** are inappropriate for the *qalter* utility, since they capture potentially transient environment information from the submitting process. The **-q** option would specify a new queue, which would largely negate the previously stated advantage of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

Each of the following paragraphs provides the rationale for a *qalter* option.

Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

The **-a** option allows users to alter the date and time at which a batch job becomes eligible to run.

The **-A** option allows users to change the account that will be charged for the resources consumed by the batch job. Support for the **-A** option is mandatory for conforming implementations of *qalter*, even though support of accounting is optional for servers. Whether or not to support accounting is left to the implementor of the server, but mandatory support of the **-A** option assures users of a consistent interface and allows them to control accounting on servers that support accounting.

The **-c** option allows users to alter the checkpointing interval of a batch job. A checkpointing system, which is not defined by POSIX.1-200x, allows recovery of a batch job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume expensive computing time or must meet a critical schedule. Users should be allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely completion of the batch job; therefore, this volume of POSIX.1-200x provides the checkpointing interval option. Support for checkpointing is optional for servers.

The **-e** option allows users to alter the name and location of the standard error stream written by a batch job. However, the path of the standard error stream is meaningless if the value of the *Join_Path* attribute of the batch job is TRUE.

The **-h** option allows users to set the hold type in the *Hold_Types* attribute of a batch job. The *qhold* and *qrls* utilities add or remove hold types to the *Hold_Types* attribute, respectively. The **-h** option has been modified to allow for implementation-defined hold types.

The **-j** option allows users to alter the decision to join (merge) the standard error stream of the batch job with the standard output stream of the batch job.

The **-l** option allows users to change the resource limits imposed on a batch job.

The **-m** option allows users to modify the list of points in the life of a batch job at which the designated users will receive mail notification.

- 100277 The **-M** option allows users to alter the list of users who will receive notification about events in
100278 the life of a batch job.
- 100279 The **-N** option allows users to change the name of a batch job.
- 100280 The **-o** option allows users to alter the name and path to which the standard output stream of
100281 the batch job will be written.
- 100282 The **-P** option allows users to modify the priority of a batch job. Support for priority is optional
100283 for batch servers.
- 100284 The **-r** option allows users to alter the rerunability status of a batch job.
- 100285 The **-S** option allows users to change the name and location of the shell image that will be
100286 invoked to interpret the script of the batch job. This option has been modified to allow a list of
100287 shell name and locations associated with different hosts.
- 100288 The **-u** option allows users to change the user identifier under which the batch job will execute.
- 100289 The *job_identifier* operand syntax is provided so that the user can differentiate between the
100290 originating and destination (or executing) batch server. These may or may not be the same. The
100291 *.server_name* portion identifies the originating batch server, while the *@server* portion identifies
100292 the destination batch server.
- 100293 Historically, the *qalter* utility has been a component of the Network Queuing System (NQS), the
100294 existing practice from which this utility has been derived.

100295 FUTURE DIRECTIONS

100296 The *qalter* utility may be removed in a future version.

100297 SEE ALSO

100298 [Chapter 3](#) (on page 2319), *qdel*, *qhold*, *qmove*, *qrls*, *qsub*, *touch*

100299 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

100300 CHANGE HISTORY

100301 Derived from IEEE Std 1003.2d-1994.

100302 Issue 6

100303 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

100304 IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the **-a** option.

100305 Issue 7

100306 The *qalter* utility is marked obsolescent.

100307 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100308 **NAME**

100309 qdel — delete batch jobs

100310 **SYNOPSIS**100311 OB BE `qdel job_identifier...`100312 **DESCRIPTION**100313 A batch job is deleted by sending a request to the batch server that manages the batch job. A
100314 batch job that has been deleted is no longer subject to management by batch services.100315 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or
100316 more batch jobs.100317 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch
100318 *job_identifier* is presented to the utility.100319 The *qdel* utility shall delete batch jobs in the order in which their batch *job_identifiers* are
100320 presented to the utility.100321 If the *qdel* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
100322 process the remaining batch *job_identifiers*, if any.100323 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that
100324 manages the batch job.100325 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed
100326 batch *job_identifier* has been deleted.100327 **OPTIONS**

100328 None.

100329 **OPERANDS**100330 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch
100331 *job_identifier* (see [Section 3.3.1](#), on page 2340).100332 **STDIN**

100333 Not used.

100334 **INPUT FILES**

100335 None.

100336 **ENVIRONMENT VARIABLES**100337 The following environment variables shall affect the execution of *qdel*:100338 *LANG* Provide a default value for the internationalization variables that are unset or null. |
100339 (See [XBD Section 8.2](#) (on page 160) the precedence of internationalization variables |
100340 used to determine the values of locale categories.)100341 *LC_ALL* If set to a non-empty string value, override the values of all the other
100342 internationalization variables.100343 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
100344 characters (for example, single-byte as opposed to multi-byte characters in
100345 arguments).100346 *LC_MESSAGES*100347 Determine the locale that should be used to affect the format and contents of
100348 diagnostic messages written to standard error.

100349 *LOGNAME* Determine the login name of the user.

100350 **ASYNCHRONOUS EVENTS**

100351 Default.

100352 **STDOUT**

100353 An implementation of the *qdel* utility may write informative messages to standard output.

100354 **STDERR**

100355 The standard error shall be used only for diagnostic messages.

100356 **OUTPUT FILES**

100357 None.

100358 **EXTENDED DESCRIPTION**

100359 None.

100360 **EXIT STATUS**

100361 The following exit values shall be returned:

100362 0 Successful completion.

100363 >0 An error occurred.

100364 **CONSEQUENCES OF ERRORS**

100365 In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic
 100366 message to standard error when the error reply received from a batch server indicates that the
 100367 batch *job_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the
 100368 diagnostic message while attempting to locate the job on other servers is implementation-
 100369 defined.

100370 **APPLICATION USAGE**

100371 None.

100372 **EXAMPLES**

100373 None.

100374 **RATIONALE**

100375 The *qdel* utility allows users and administrators to delete jobs.

100376 The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility
 100377 of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in
 100378 on a remote node, because the *kill* utility does not operate across the network. Second, unlike
 100379 *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job
 100380 identifiers rather than process identifiers, and so this utility can be passed the output of the
 100381 *qselect* utility, thus providing users with a means of deleting a list of jobs.

100382 Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been
 100383 complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted
 100384 are identified individually by their job identifiers.

100385 Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is
 100386 based. However, the *qdel* utility defined in this volume of POSIX.1-200x does not provide an
 100387 option for specifying a signal number to send to the batch job prior to the killing of the process;
 100388 that capability has been subsumed by the *qsig* utility.

100389 A discussion was held about the delays of networking and the possibility that the batch server
 100390 may never respond, due to a down router, down batch server, or other network mishap. The
 100391 DESCRIPTION records this under the words “fails to process any job identifier”. In the broad
 100392 sense, the network problem is also an error, which causes the failure to process the batch job
 100393 identifier.

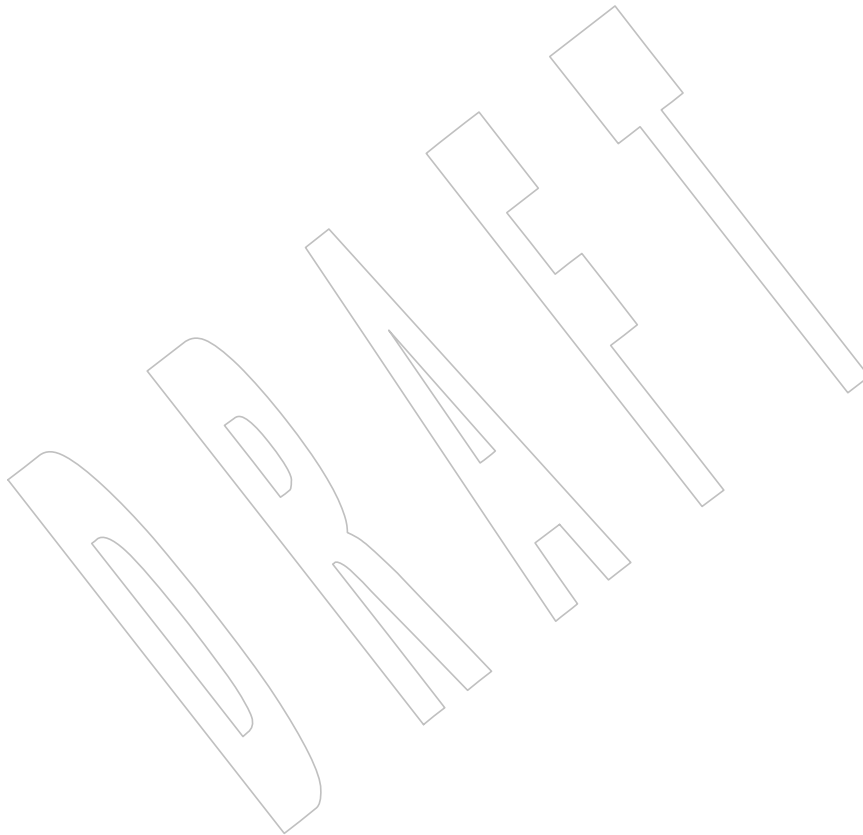
100394 **FUTURE DIRECTIONS**
100395 The *qdel* utility may be removed in a future version.

100396 **SEE ALSO**
100397 [Chapter 3](#) (on page 2319), *kill*, *qselect*, *qsig*
100398 XBD [Chapter 8](#) (on page 159)

100399 **CHANGE HISTORY**
100400 Derived from IEEE Std 1003.2d-1994.

100401 **Issue 6**
100402 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

100403 **Issue 7**
100404 The *qdel* utility is marked obsolescent.
100405 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



100406 **NAME**
 100407 qhold — hold batch jobs

100408 **SYNOPSIS**
 100409 OB BE qhold [-h *hold_list*] *job_identifier*...

100410 DESCRIPTION

100411 A hold is placed on a batch job by a request to the batch server that manages the batch job. A
 100412 batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-
 100413 accessible client of batch services that requests one or more types of hold to be placed on one or
 100414 more batch jobs.

100415 The *qhold* utility shall place holds on those batch jobs for which a batch *job_identifier* is presented
 100416 to the utility.

100417 The *qhold* utility shall place holds on batch jobs in the order in which their batch *job_identifiers*
 100418 are presented to the utility. If the *qhold* utility fails to process any batch *job_identifier* successfully,
 100419 the utility shall proceed to process the remaining batch *job_identifiers*, if any.

100420 The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch
 100421 server that manages the batch job.

100422 The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to
 100423 each successfully processed batch *job_identifier*.

100424 OPTIONS

100425 The *qhold* utility shall conform to XBD [Section 12.2](#) (on page 201).

100426 The following option shall be supported by the implementation:

100427 **-h *hold_list*** Define the types of holds to be placed on the batch job.

100428 The *qhold* **-h** option shall accept a value for the *hold_list* option-argument that is a
 100429 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 100430 [6.1](#), on page 111).

100431 The *qhold* utility shall accept a value for the *hold_list* option-argument that is a
 100432 string of one or more of the characters 'u', 's', or 'o', or the single character
 100433 'n'.

100434 For each unique character in the *hold_list* option-argument, the *qhold* utility shall
 100435 add a value to the *Hold_Types* attribute of the batch job as follows, each
 100436 representing a different hold type:

100437 u USER

100438 s SYSTEM

100439 o OPERATOR

100440 If any of these characters are duplicated in the *hold_list* option-argument, the
 100441 duplicates shall be ignored.

100442 An existing *Hold_Types* attribute can be cleared by the following hold type:

100443 n NO_HOLD

100444 The *qhold* utility shall consider it an error if any hold type other than 'n' is
 100445 combined with hold type 'n'.

100446 Strictly conforming applications shall not repeat any of the characters 'u', 's',

100447 'o', or 'n' within the *hold_list* option-argument. The *qhold* utility shall permit the
 100448 repetition of characters, but shall not assign additional meaning to the repeated
 100449 characters.

100450 An implementation may define other hold types. The conformance document for
 100451 an implementation shall describe any additional hold types, how they are
 100452 specified, their internal behavior, and how they affect the behavior of the utility.

100453 If the **-h** option is not presented to the *qhold* utility, the implementation shall set
 100454 the *Hold_Types* attribute to USER.

100455 OPERANDS

100456 The *qhold* utility shall accept one or more operands that conform to the syntax for a batch
 100457 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100458 STDIN

100459 Not used.

100460 INPUT FILES

100461 None.

100462 ENVIRONMENT VARIABLES

100463 The following environment variables shall affect the execution of *qhold*:

100464 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100465 (See [XBD Section 8.2](#) (on page 160) the precedence of internationalization variables |
 100466 used to determine the values of locale categories.)

100467 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100468 internationalization variables.

100469 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100470 characters (for example, single-byte as opposed to multi-byte characters in
 100471 arguments).

100472 *LC_MESSAGES*
 100473 Determine the locale that should be used to affect the format and contents of
 100474 diagnostic messages written to standard error.

100475 *LOGNAME* Determine the login name of the user.

100476 ASYNCHRONOUS EVENTS

100477 Default.

100478 STDOUT

100479 None.

100480 STDERR

100481 The standard error shall be used only for diagnostic messages.

100482 OUTPUT FILES

100483 None.

100484 EXTENDED DESCRIPTION

100485 None.

100486 EXIT STATUS

100487 The following exit values shall be returned:

100488 0 Successful completion.

100489 >0 An error occurred.

100490 **CONSEQUENCES OF ERRORS**

100491 In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic
 100492 message to standard error when the error reply received from a batch server indicates that the
 100493 batch *job_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output
 100494 the diagnostic message while attempting to locate the job on other servers is implementation-
 100495 defined.

100496 **APPLICATION USAGE**

100497 None.

100498 **EXAMPLES**

100499 None.

100500 **RATIONALE**

100501 The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job
 100502 ineligible for execution.

100503 The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish
 100504 to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired
 100505 using the *qselect* utility.

100506 The **-h** option allows the user to specify the type of hold that is to be placed on the job. This
 100507 option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The
 100508 USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify
 100509 that the user is authorized to set the specified hold for the batch job.

100510 Mail is not required on hold because the administrator has the tools and libraries to build this
 100511 option if he or she wishes.

100512 Historically, the *qhold* utility has been a part of some existing batch systems, although it has not
 100513 traditionally been a part of the NQS.

100514 **FUTURE DIRECTIONS**

100515 The *qhold* utility may be removed in a future version.

100516 **SEE ALSO**

100517 [Chapter 3](#) (on page 2319), *qselect*

100518 [XBD Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

100519 **CHANGE HISTORY**

100520 Derived from IEEE Std 1003.2d-1994.

100521 **Issue 6**

100522 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

100523 **Issue 7**

100524 The *qhold* utility is marked obsolescent.

100525 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100526 **NAME**

100527 qmove — move batch jobs

100528 **SYNOPSIS**100529 OB BE `qmove destination job_identifier...`100530 **DESCRIPTION**

100531 To move a batch job is to remove the batch job from the batch queue in which it resides and
 100532 instantiate the batch job in another batch queue. A batch job is moved by a request to the batch
 100533 server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests
 100534 the movement of one or more batch jobs.

100535 The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch
 100536 *job_identifier* is presented to the utility.

100537 The *qmove* utility shall move batch jobs in the order in which the corresponding batch
 100538 *job_identifiers* are presented to the utility.

100539 If the *qmove* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 100540 process the remaining batch *job_identifiers*, if any.

100541 The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that
 100542 manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all
 100543 successfully processed batch *job_identifiers* have been moved.

100544 **OPTIONS**

100545 None.

100546 **OPERANDS**

100547 The *qmove* utility shall accept one operand that conforms to the syntax for a destination (see
 100548 [Section 3.3.2](#), on page 2341).

100549 The *qmove* utility shall accept one or more operands that conform to the syntax for a batch
 100550 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100551 **STDIN**

100552 Not used.

100553 **INPUT FILES**

100554 None.

100555 **ENVIRONMENT VARIABLES**100556 The following environment variables shall affect the execution of *qmove*:

100557 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100558 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables |
 100559 used to determine the values of locale categories.)

100560 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100561 internationalization variables.

100562 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100563 characters (for example, single-byte as opposed to multi-byte characters in
 100564 arguments).

100565 *LC_MESSAGES*

100566 Determine the locale that should be used to affect the format and contents of
 100567 diagnostic messages written to standard error.

qmove

Utilities

100568 *LOGNAME* Determine the login name of the user.

100569 **ASYNCHRONOUS EVENTS**

100570 Default.

100571 **STDOUT**

100572 None.

100573 **STDERR**

100574 The standard error shall be used only for diagnostic messages.

100575 **OUTPUT FILES**

100576 None.

100577 **EXTENDED DESCRIPTION**

100578 None.

100579 **EXIT STATUS**

100580 The following exit values shall be returned:

100581 0 Successful completion.

100582 >0 An error occurred.

100583 **CONSEQUENCES OF ERRORS**

100584 In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

100589 **APPLICATION USAGE**

100590 None.

100591 **EXAMPLES**

100592 None.

100593 **RATIONALE**

100594 The *qmove* utility allows users to move jobs between queues.

100595 The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails considerably more typing.

100597 Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility, the only option of the *qmove* utility concerns authorization. The **-u** option provides the user with the convenience of changing the user identifier under which the batch job will execute. Minimalism and consistency have taken precedence over convenience; the **-u** option has been deleted because the equivalent capability exists with the **-u** option of the *qalter* utility.

100602 **FUTURE DIRECTIONS**

100603 The *qmove* utility may be removed in a future version.

100604 **SEE ALSO**

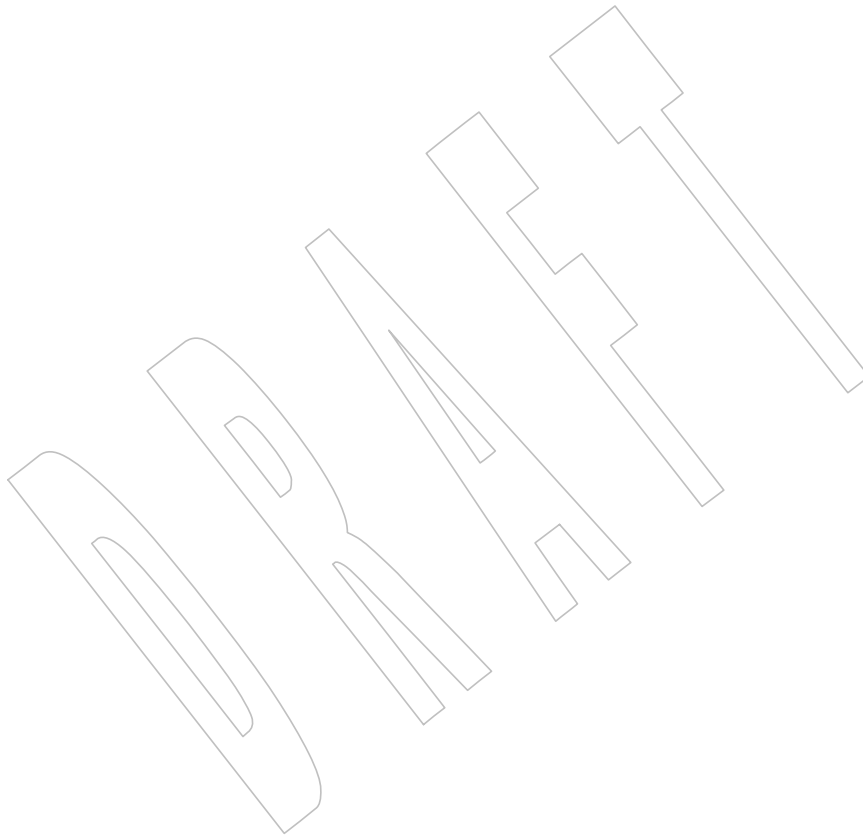
100605 [Chapter 3](#) (on page 2319), *qalter*, *qselect*

100606 [XBD Chapter 8](#) (on page 159)

100607 **CHANGE HISTORY**

100608 Derived from IEEE Std 1003.2d-1994.

- 100609 **Issue 6**
- 100610 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.
- 100611 **Issue 7**
- 100612 The *qmove* utility is marked obsolescent.
- 100613 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



100614 **NAME**

100615 qmsg — send message to batch jobs

100616 **SYNOPSIS**100617 OB BE `qmsg [-EO] message_string job_identifiers...`100618 **DESCRIPTION**

100619 To send a message to a batch job is to request that a server write a message string into one or
 100620 more output files of the batch job. A message is sent to a batch job by a request to the batch
 100621 server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests
 100622 the sending of messages to one or more batch jobs.

100623 The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request*
 100624 to the batch server that manages the batch job. The *qmsg* utility shall not directly write the
 100625 message into the files of the batch job.

100626 The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs,
 100627 for which a batch *job_identifier* is presented to the utility.

100628 The *qmsg* utility shall send *Job Message Requests* for batch jobs in the order in which their batch
 100629 *job_identifiers* are presented to the utility.

100630 If the *qmsg* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 100631 process the remaining batch *job_identifiers*, if any.

100632 The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that
 100633 manages the batch job that corresponds to each successfully processed batch *job_identifier*.

100634 **OPTIONS**100635 The *qmsg* utility shall conform to XBD [Section 12.2](#) (on page 201).

100636 The following options shall be supported by the implementation:

100637 **-E** Specify that the message is written to the standard error of each batch job.100638 The *qmsg* utility shall write the message into the standard error of the batch job.100639 **-O** Specify that the message is written to the standard output of each batch job.100640 The *qmsg* utility shall write the message into the standard output of the batch job.

100641 If neither the **-O** nor the **-E** option is presented to the *qmsg* utility, the utility shall write the
 100642 message into an implementation-defined file. The conformance document for the
 100643 implementation shall describe the name and location of the implementation-defined file. If both
 100644 the **-O** and the **-E** options are presented to the *qmsg* utility, then the utility shall write the
 100645 messages to both standard output and standard error.

100646 **OPERANDS**100647 The *qmsg* utility shall accept a minimum of two operands, *message_string* and one or more batch
100648 *job_identifiers*.

100649 The *message_string* operand shall be the string to be written to one or more output files of the
 100650 batch job followed by a <newline>. If the string contains <blank>s, then the application shall
 100651 ensure that the string is quoted. The *message_string* shall be encoded in the portable character set
 100652 (see XBD [Section 6.1](#), on page 111).

100653 All remaining operands are batch *job_identifiers* that conform to the syntax for a batch
 100654 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100655 **STDIN**

100656 Not used.

100657 **INPUT FILES**

100658 None.

100659 **ENVIRONMENT VARIABLES**100660 The following environment variables shall affect the execution of *qmsg*:

100661 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100662 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 100663 used to determine the values of locale categories.)

100664 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100665 internationalization variables.

100666 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100667 characters (for example, single-byte as opposed to multi-byte characters in
 100668 arguments).

100669 *LC_MESSAGES*
 100670 Determine the locale that should be used to affect the format and contents of
 100671 diagnostic messages written to standard error.

100672 *LOGNAME* Determine the login name of the user.

100673 **ASYNCHRONOUS EVENTS**

100674 Default.

100675 **STDOUT**

100676 None.

100677 **STDERR**

100678 The standard error shall be used only for diagnostic messages.

100679 **OUTPUT FILES**

100680 None.

100681 **EXTENDED DESCRIPTION**

100682 None.

100683 **EXIT STATUS**

100684 The following exit values shall be returned:

100685 0 Successful completion.

100686 >0 An error occurred.

100687 **CONSEQUENCES OF ERRORS**

100688 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic
 100689 message to standard error when the error reply received from a batch server indicates that the
 100690 batch *job_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output
 100691 the diagnostic message while attempting to locate the job on other servers is implementation-
 100692 defined.

100693 **APPLICATION USAGE**

100694 None.

100695 **EXAMPLES**

100696 None.

100697 **RATIONALE**

100698 The *qmsg* utility allows users to write messages into the output files of running jobs. Users,
 100699 including operators and administrators, have a number of occasions when they want to place
 100700 messages in the output files of a batch job. For example, if a disk that is being used by a batch job
 100701 is showing errors, the operator might note this in the standard error stream of the batch job.

100702 The options of the *qmsg* utility provide users with the means of placing the message in the
 100703 output stream of their choice. The default output stream for the message—if the user does not
 100704 designate an output stream—is implementation-defined, since many implementations will
 100705 provide, as an extension to this volume of POSIX.1-200x, a log file that shows the history of
 100706 utility execution.

100707 If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can
 100708 be used to acquire the appropriate list of job identifiers.

100709 The **-E** option allows users to place the message in the standard error stream of the batch job.

100710 The **-O** option allows users to place the message in the standard output stream of the batch job.

100711 Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors
 100712 of an NQS-derived batch system. The utility has been found to be useful enough that it deserves
 100713 to be included in this volume of POSIX.1-200x.

100714 **FUTURE DIRECTIONS**100715 The *qmsg* utility may be removed in a future version.100716 **SEE ALSO**100717 [Chapter 3](#) (on page 2319), *qselect*100718 [XBD Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +100719 **CHANGE HISTORY**

100720 Derived from IEEE Std 1003.2d-1994.

100721 **Issue 6**100722 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.100723 **Issue 7**100724 The *qmsg* utility is marked obsolescent.

100725 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100726 **NAME**

100727 qrerun — rerun batch jobs

100728 **SYNOPSIS**100729 OB BE `qrerun job_identifier...`100730 **DESCRIPTION**

100731 To rerun a batch job is to terminate the session leader of the batch job, delete any associated
 100732 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a
 100733 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible
 100734 batch client that requests the rerunning of one or more batch jobs.

100735 The *qrerun* utility shall rerun those batch jobs for which a batch *job_identifier* is presented to the
 100736 utility.

100737 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job_identifiers* are
 100738 presented to the utility.

100739 If the *qrerun* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
 100740 process the remaining batch *job_identifiers*, if any.

100741 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that
 100742 manages each batch job.

100743 For each successfully processed batch *job_identifier*, the *qrerun* utility shall have rerun the
 100744 corresponding batch job at the time the utility exits.

100745 **OPTIONS**

100746 None.

100747 **OPERANDS**

100748 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch
 100749 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100750 **STDIN**

100751 Not used.

100752 **INPUT FILES**

100753 None.

100754 **ENVIRONMENT VARIABLES**100755 The following environment variables shall affect the execution of *qrerun*:

100756 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100757 (See XBD [Section 8.2](#) (on page 160) the precedence of internationalization variables |
 100758 used to determine the values of locale categories.)

100759 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100760 internationalization variables.

100761 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100762 characters (for example, single-byte as opposed to multi-byte characters in
 100763 arguments).

100764 *LC_MESSAGES*

100765 Determine the locale that should be used to affect the format and contents of
 100766 diagnostic messages written to standard error.

100767 *LOGNAME* Determine the login name of the user.

100768 **ASYNCHRONOUS EVENTS**

100769 Default.

100770 **STDOUT**

100771 None.

100772 **STDERR**

100773 The standard error shall be used only for diagnostic messages.

100774 **OUTPUT FILES**

100775 None.

100776 **EXTENDED DESCRIPTION**

100777 None.

100778 **EXIT STATUS**

100779 The following exit values shall be returned:

100780 0 Successful completion.

100781 >0 An error occurred.

100782 **CONSEQUENCES OF ERRORS**

100783 In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

100788 **APPLICATION USAGE**

100789 None.

100790 **EXAMPLES**

100791 None.

100792 **RATIONALE**

100793 The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.

100794 The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume of POSIX.1-200x to correct user-perceived deficiencies in the existing practice.

100796 **FUTURE DIRECTIONS**

100797 The *qrerun* utility may be removed in a future version.

100798 **SEE ALSO**

100799 [Chapter 3](#) (on page 2319)

100800 [XBD Chapter 8](#) (on page 159)

+

100801 **CHANGE HISTORY**

100802 Derived from IEEE Std 1003.2d-1994.

100803 **Issue 6**

100804 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

100805 **Issue 7**

100806 The *qrerun* utility is marked obsolescent.

100807 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100808 **NAME**
 100809 qrln — release batch jobs

100810 **SYNOPSIS**
 100811 OB BE qrln [-h *hold_list*] *job_identifier*...

100812 DESCRIPTION

100813 A batch job might have one or more holds, which prevent the batch job from executing. A batch
 100814 job from which all the holds have been removed becomes eligible for execution and is said to
 100815 have been released. A batch job hold is removed by sending a request to the batch server that
 100816 manages the batch job. The *qrln* utility is a user-accessible client of batch services that requests
 100817 holds be removed from one or more batch jobs.

100818 The *qrln* utility shall remove one or more holds from those batch jobs for which a batch
 100819 *job_identifier* is presented to the utility.

100820 The *qrln* utility shall remove holds from batch jobs in the order in which their batch *job_identifiers*
 100821 are presented to the utility.

100822 If the *qrln* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
 100823 process the remaining batch *job_identifiers*, if any.

100824 The *qrln* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch
 100825 server that manages the batch job.

100826 The *qrln* utility shall not exit until the holds have been removed from the batch job
 100827 corresponding to each successfully processed batch *job_identifier*.

100828 OPTIONS

100829 The *qrln* utility shall conform to XBD [Section 12.2](#) (on page 201).

100830 The following option shall be supported by the implementation:

100831 **-h *hold_list*** Define the types of holds to be removed from the batch job.

100832 The *qrln* **-h** option shall accept a value for the *hold_list* option-argument that is a
 100833 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 100834 [6.1](#), on page 111).

100835 The *qrln* utility shall accept a value for the *hold_list* option-argument that is a string
 100836 of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

100837 For each unique character in the *hold_list* option-argument, the *qrln* utility shall add
 100838 a value to the *Hold_Types* attribute of the batch job as follows, each representing a
 100839 different hold type:

100840 u USER

100841 s SYSTEM

100842 o OPERATOR

100843 If any of these characters are duplicated in the *hold_list* option-argument, the
 100844 duplicates shall be ignored.

100845 An existing *Hold_Types* attribute can be cleared by the following hold type:

100846 n NO_HOLD

100847 The *qrln* utility shall consider it an error if any hold type other than 'n' is
 100848 combined with hold type 'n'.

100849 Strictly conforming applications shall not repeat any of the characters 'u', 's',
 100850 'o', or 'n' within the *hold_list* option-argument. The *qrls* utility shall permit the
 100851 repetition of characters, but shall not assign additional meaning to the repeated
 100852 characters.

100853 An implementation may define other hold types. The conformance document for
 100854 an implementation shall describe any additional hold types, how they are
 100855 specified, their internal behavior, and how they affect the behavior of the utility.

100856 If the **-h** option is not presented to the *qrls* utility, the implementation shall remove
 100857 the USER hold in the *Hold_Types* attribute.

100858 OPERANDS

100859 The *qrls* utility shall accept one or more operands that conform to the syntax for a batch
 100860 *job_identifier* (see [Section 3.3.1](#), on page 2340).

100861 STDIN

100862 Not used.

100863 INPUT FILES

100864 None.

100865 ENVIRONMENT VARIABLES

100866 The following environment variables shall affect the execution of *qrls*:

100867 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 100868 (See [XBD Section 8.2](#) (on page 160) the precedence of internationalization variables |
 100869 used to determine the values of locale categories.)

100870 *LC_ALL* If set to a non-empty string value, override the values of all the other
 100871 internationalization variables.

100872 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 100873 characters (for example, single-byte as opposed to multi-byte characters in
 100874 arguments).

100875 *LC_MESSAGES*
 100876 Determine the locale that should be used to affect the format and contents of
 100877 diagnostic messages written to standard error.

100878 *LOGNAME* Determine the login name of the user.

100879 ASYNCHRONOUS EVENTS

100880 Default.

100881 STDOUT

100882 None.

100883 STDERR

100884 The standard error shall be used only for diagnostic messages.

100885 OUTPUT FILES

100886 None.

100887 EXTENDED DESCRIPTION

100888 None.

100889 EXIT STATUS

100890 The following exit values shall be returned:

100891 0 Successful completion.

100892 >0 An error occurred.

100893 CONSEQUENCES OF ERRORS

100894 In addition to the default behavior, the *qrls* utility shall not be required to write a diagnostic
 100895 message to standard error when the error reply received from a batch server indicates that the
 100896 batch *job_identifier* does not exist on the server. Whether or not the *qrls* utility waits to output the
 100897 diagnostic message while attempting to locate the job on other servers is implementation-
 100898 defined.

100899 APPLICATION USAGE

100900 None.

100901 EXAMPLES

100902 None.

100903 RATIONALE

100904 The *qrls* utility allows users, operators, and administrators to remove holds from jobs.

100905 The *qrls* utility does not support any job selection options or wildcard arguments. Users may
 100906 acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could
 100907 select all of their held jobs.

100908 The *-h* option allows the user to specify the type of hold that is to be removed. This option
 100909 allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch
 100910 server that manages the batch job will verify whether the user is authorized to remove the
 100911 specified hold for the batch job. If more than one type of hold has been placed on the batch job, a
 100912 user may wish to remove only some of them.

100913 Mail is not required on release because the administrator has the tools and libraries to build this
 100914 option if required.

100915 The *qrls* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of
 100916 POSIX.1-200x as the natural complement to the *qhold* utility.

100917 FUTURE DIRECTIONS

100918 The *qrls* utility may be removed in a future version.

100919 SEE ALSO

100920 [Chapter 3](#) (on page 2319), [qhold](#), [qselect](#)

100921 XBD [Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

100922 CHANGE HISTORY

100923 Derived from IEEE Std 1003.2d-1994.

100924 Issue 6

100925 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

100926 Issue 7

100927 The *qrls* utility is marked obsolescent.

100928 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

100929 **NAME**

100930 qselect — select batch jobs

100931 **SYNOPSIS**

```
100932 OB BE qselect [-a [op]date_time] [-A account_string] [-c [op]interval]
100933 [-h hold_list] [-l resource_list] [-N name] [-p [op]priority]
100934 [-q destination] [-r y|n] [-s states] [-u user_list]
```

100935 **DESCRIPTION**

100936 To select a set of batch jobs is to return the batch *job_identifiers* for each batch job that meets a list
 100937 of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect* utility
 100938 is a user-accessible batch client that requests the selection of batch jobs.

100939 Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch
 100940 *job_identifiers* that meet the criteria specified by the options and option-arguments presented to
 100941 the utility.

100942 The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The
 100943 *qselect* utility shall not exit until the server replies to each request generated.

100944 For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch
 100945 jobs as described in the OPTIONS section.

100946 The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required
 100947 by the options presented to the utility.

100948 When an option is specified with a mandatory or optional *op* component to the option-
 100949 argument, then *op* shall specify a relation between the value of a certain batch job attribute and
 100950 the *value* component of the option-argument. If an *op* is allowable on an option, then the
 100951 description of the option letter indicates the *op* as either mandatory or optional. Acceptable
 100952 strings for the *op* component, and the relation the string indicates, are shown in the following
 100953 list:

100954 .eq. The value represented by the attribute of the batch job is equal to the value represented
 100955 by the option-argument.

100956 .ge. The value represented by the attribute of the batch job is greater than or equal to the
 100957 value represented by the option-argument.

100958 .gt. The value represented by the attribute of the batch job is greater than the value
 100959 represented by the option-argument.

100960 .lt. The value represented by the attribute of the batch job is less than the value represented
 100961 by the option-argument.

100962 .le. The value represented by the attribute of the batch job is less than or equal to the value
 100963 represented by the option-argument.

100964 .ne. The value represented by the attribute of the batch job is not equal to the value
 100965 represented by the option-argument.

100966 **OPTIONS**

100967 The *qselect* utility shall conform to XBD [Section 12.2](#) (on page 201).

100968 The following options shall be supported by the implementation:

100969 **-a** [op]date_time

100970 Restrict selection to a specific time, or a range of times.

- 100971 The *qselect* utility shall select only batch jobs for which the value of the
 100972 *Execution_Time* attribute is related to the Epoch equivalent of the local time
 100973 expressed by the value of the *date_time* component of the option-argument in the
 100974 manner indicated by the value of the *op* component of the option-argument.
- 100975 The *qselect* utility shall accept a *date_time* component of the option-argument that
 100976 conforms to the syntax of the *time* operand of the *touch* utility.
- 100977 If the *op* component of the option-argument is not presented to the *qselect* utility,
 100978 the utility shall select batch jobs for which the *Execution_Time* attribute is equal to
 100979 the *date_time* component of the option-argument.
- 100980 When comparing times, the *qselect* utility shall use the following definitions for the
 100981 *op* component of the option-argument:
- 100982 .eq. The time represented by value of the *Execution_Time* attribute of the batch
 100983 job is equal to the time represented by the *date_time* component of the
 100984 option-argument.
 - 100985 .ge. The time represented by value of the *Execution_Time* attribute of the batch
 100986 job is after or equal to the time represented by the *date_time* component of
 100987 the option-argument.
 - 100988 .gt. The time represented by value of the *Execution_Time* attribute of the batch
 100989 job is after the time represented by the *date_time* component of the option-
 100990 argument.
 - 100991 .lt. The time represented by value of the *Execution_Time* attribute of the batch
 100992 job is before the time represented by the *date_time* component of the
 100993 option-argument.
 - 100994 .le. The time represented by value of the *Execution_Time* attribute of the batch
 100995 job is before or equal to the time represented by the *date_time* component
 100996 of the option-argument.
 - 100997 .ne. The time represented by value of the *Execution_Time* attribute of the batch
 100998 job is not equal to the time represented by the *date_time* component of the
 100999 option-argument.
- 101000 The *qselect* utility shall accept the defined character strings for the *op* component of
 101001 the option-argument.
- 101002 **-A** *account_string*
 101003 Restrict selection to the batch jobs charging a specified account.
- 101004 The *qselect* utility shall select only batch jobs for which the value of the
 101005 *Account_Name* attribute of the batch job matches the value of the *account_string*
 101006 option-argument.
- 101007 The syntax of the *account_string* option-argument is unspecified.
- 101008 **-c** [*op*]*interval*
 101009 Restrict selection to batch jobs within a range of checkpoint intervals.
- 101010 The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint*
 101011 attribute relates to the value of the *interval* component of the option-argument in
 101012 the manner indicated by the value of the *op* component of the option-argument.
- 101013 If the *op* component of the option-argument is omitted, the *qselect* utility shall select
 101014 batch jobs for which the value of the *Checkpoint* attribute is equal to the value of the
 101015 *interval* component of the option-argument.
- 101016 When comparing checkpoint intervals, the *qselect* utility shall use the following

definitions for the *op* component of the option-argument:

.eq. The value of the *Checkpoint* attribute of the batch job equals the value of the *interval* component of the option-argument.

.ge. The value of the *Checkpoint* attribute of the batch job is greater than or equal to the value of the *interval* component option-argument.

.gt. The value of the *Checkpoint* attribute of the batch job is greater than the value of the *interval* component option-argument.

.lt. The value of the *Checkpoint* attribute of the batch job is less than the value of the *interval* component option-argument.

.le. The value of the *Checkpoint* attribute of the batch job is less than or equal to the value of the *interval* component option-argument.

.ne. The value of the *Checkpoint* attribute of the batch job does not equal the value of the *interval* component option-argument.

The *qselect* utility shall accept the defined character strings for the *op* component of the option-argument.

The ordering relationship for the values of the interval option-argument is defined to be:

'n' .gt. 's' .gt. 'c=minutes' .ge. 'c'

When comparing *Checkpoint* attributes with an interval having the value of the single character 'u', only equality or inequality are valid comparisons.

-h hold_list Restrict selection to batch jobs that have a specific type of hold.

The *qselect* utility shall select only batch jobs for which the value of the *Hold_Types* attribute matches the value of the *hold_list* option-argument.

The *qselect* **-h** option shall accept a value for the *hold_list* option-argument that is a string of alphanumeric characters in the portable character set (see XBD Section 6.1, on page 111).

The *qselect* utility shall accept a value for the *hold_list* option-argument that is a string of one or more of the characters 'u', 's', or 'o', or the single character 'n'.

Each unique character in the *hold_list* option-argument of the *qselect* utility is defined as follows, each representing a different hold type:

u USER
s SYSTEM
o OPERATOR

If any of these characters are duplicated in the *hold_list* option-argument, the duplicates shall be ignored.

The *qselect* utility shall consider it an error if any hold type other than 'n' is combined with hold type 'n'.

Strictly conforming applications shall not repeat any of the characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument. The *qselect* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other hold types. The conformance document for

- 101060 an implementation shall describe any additional hold types, how they are
101061 specified, their internal behavior, and how they affect the behavior of the utility.
- 101062 **-l *resource_list***
101063 Restrict selection to batch jobs with specified resource limits and attributes.
- 101064 The *qselect* utility shall accept a *resource_list* option-argument with the following
101065 syntax:
- 101066 *resource_name op value [, , resource_name op value, , ...]*
- 101067 When comparing resource values, the *qselect* utility shall use the following
101068 definitions for the *op* component of the option-argument:
- 101069 .eq. The value of the resource of the same name in the *Resource_List* attribute
101070 of the batch job equals the value of the value component of the option-
101071 argument.
 - 101072 .ge. The value of the resource of the same name in the *Resource_List* attribute
101073 of the batch job is greater than or equal to the value of the *value*
101074 component of the option-argument.
 - 101075 .gt. The value of the resource of the same name in the *Resource_List* attribute
101076 of the batch job is greater than the value of the value component of the
101077 option-argument.
 - 101078 .lt. The value of the resource of the same name in the *Resource_List* attribute
101079 of the batch job is less than the value of the value component of the
101080 option-argument.
 - 101081 .ne. The value of the resource of the same name in the *Resource_List* attribute
101082 of the batch job does not equal the value of the value component of the
101083 option-argument.
 - 101084 .le. The value of the resource of the same name in the *Resource_List* attribute
101085 of the batch job is less than or equal to the value of the *value* component of
101086 the option-argument.
- 101087 When comparing the limit of a *Resource_List* attribute with the *value* component of
101088 the option-argument, if the limit, the value, or both are non-numeric, only equality
101089 or inequality are valid comparisons.
- 101090 The *qselect* utility shall select only batch jobs for which the values of the
101091 *resource_names* listed in the *resource_list* option-argument match the corresponding
101092 limits of the *Resource_List* attribute of the batch job.
- 101093 Limits of *resource_names* present in the *Resource_List* attribute of the batch job that
101094 have no corresponding values in the *resource_list* option-argument shall not be
101095 considered when selecting batch jobs.
- 101096 **-N *name*** Restrict selection to batch jobs with a specified name.
- 101097 The *qselect* utility shall select only batch jobs for which the value of the *Job_Name*
101098 attribute matches the value of the *name* option-argument. The string specified in
101099 the *name* option-argument shall be passed, uninterpreted, to the server. This allows
101100 an implementation to match "wildcard" patterns against batch job names.
- 101101 An implementation shall describe in the conformance document the format it
101102 supports for matching against the *Job_Name* attribute.
- 101103 **-p [*op*]priority**
101104 Restrict selection to batch jobs of the specified priority or range of priorities.

- 101105 The *qselect* utility shall select only batch jobs for which the value of the *Priority*
 101106 attribute of the batch job relates to the value of the *priority* component of the
 101107 option-argument in the manner indicated by the value of the *op* component of the
 101108 option-argument.
- 101109 If the *op* component of the option-argument is omitted, the *qselect* utility shall select
 101110 batch jobs for which the value of the *Priority* attribute of the batch job is equal to
 101111 the value of the *priority* component of the option-argument.
- 101112 When comparing priority values, the *qselect* utility shall use the following
 101113 definitions for the *op* component of the option-argument:
- 101114 .eq. The value of the *Priority* attribute of the batch job equals the value of the
 101115 *priority* component of the option-argument.
 - 101116 .ge. The value of the *Priority* attribute of the batch job is greater than or equal
 101117 to the value of the *priority* component option-argument.
 - 101118 .gt. The value of the *Priority* attribute of the batch job is greater than the value
 101119 of the *priority* component option-argument.
 - 101120 .lt. The value of the *Priority* attribute of the batch job is less than the value of
 101121 the *priority* component option-argument.
 - 101122 .lte. The value of the *Priority* attribute of the batch job is less than or equal to
 101123 the value of the *priority* component option-argument.
 - 101124 .ne. The value of the *Priority* attribute of the batch job does not equal the value
 101125 of the *priority* component option-argument.
- 101126 **-q destination**
 101127 Restrict selection to the specified batch queue or server, or both.
- 101128 The *qselect* utility shall select only batch jobs that are located at the destination
 101129 indicated by the value of the *destination* option-argument.
- 101130 The destination defines a batch queue, a server, or a batch queue at a server.
- 101131 The *qselect* utility shall accept an option-argument for the **-q** option that conforms
 101132 to the syntax for a destination. If the **-q** option is not presented to the *qselect* utility,
 101133 the utility shall select batch jobs from all batch queues at the default batch server.
- 101134 If the option-argument describes only a batch queue, the *qselect* utility shall select
 101135 only batch jobs from the batch queue of the specified name at the default batch
 101136 server. The means by which *qselect* determines the default server is
 101137 implementation-defined.
- 101138 If the option-argument describes only a batch server, the *qselect* utility shall select
 101139 batch jobs from all the batch queues at that batch server.
- 101140 If the option-argument describes both a batch queue and a batch server, the *qselect*
 101141 utility shall select only batch jobs from the specified batch queue at the specified
 101142 server.
- 101143 **-r y | n** Restrict selection to batch jobs with the specified rerunability status.
- 101144 The *qselect* utility shall select only batch jobs for which the value of the *Rerunable*
 101145 attribute of the batch job matches the value of the option-argument.
- 101146 The *qselect* utility shall accept a value for the option-argument that consists of
 101147 either the single character 'y' or the single character 'n'. The character 'y'
 101148 represents the value TRUE, and the character 'n' represents the value FALSE.

101149 **-s states** Restrict selection to batch jobs in the specified states.

101150 The *qselect* utility shall accept an option-argument that consists of any combination
101151 of the characters 'e', 'q', 'r', 'w', 'h', and 't'.

101152 Conforming applications shall not repeat any character in the option-argument.
101153 The *qselect* utility shall permit the repetition of characters in the option-argument,
101154 but shall not assign additional meaning to repeated characters.

101155 The *qselect* utility shall interpret the characters in the *states* option-argument as
101156 follows:

101157 e Represents the EXITING state.

101158 q Represents the QUEUED state.

101159 r Represents the RUNNING state.

101160 t Represents the TRANSITING state.

101161 h Represents the HELD state.

101162 w Represents the WAITING state.

101163 For each character in the *states* option-argument, the *qselect* utility shall select batch
101164 jobs in the corresponding state.

101165 **-u user_list** Restrict selection to batch jobs owned by the specified user names.

101166 The *qselect* utility shall select only the batch jobs of those users specified in the
101167 *user_list* option-argument.

101168 The *qselect* utility shall accept a *user_list* option-argument that conforms to the
101169 following syntax:

101170 *username*[*@host*][, , *username*[*@host*], , . . .]

101171 The *qselect* utility shall accept only one user name that is missing a corresponding
101172 host name. The *qselect* utility shall accept only one user name per named host.

101173 **OPERANDS**

101174 None.

101175 **STDIN**

101176 Not used.

101177 **INPUT FILES**

101178 None.

101179 **ENVIRONMENT VARIABLES**

101180 The following environment variables shall affect the execution of *qselect*:

101181 **LANG** Provide a default value for the internationalization variables that are unset or null. |
101182 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
101183 used to determine the values of locale categories.)

101184 **LC_ALL** If set to a non-empty string value, override the values of all the other
101185 internationalization variables.

101186 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
101187 characters (for example, single-byte as opposed to multi-byte characters in
101188 arguments).

101189 **LC_MESSAGES**

101190 Determine the locale that should be used to affect the format and contents of
101191 diagnostic messages written to standard error.

- 101192 *LOGNAME* Determine the login name of the user.
- 101193 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
101194 unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

- 101195 Default.

STDOUT

- 101197 The *qselect* utility shall write zero or more batch *job_identifiers* to standard output.
- 101198 The *qselect* utility shall separate the batch *job_identifiers* written to standard output by white
101199 space.
- 101200 The *qselect* utility shall write batch *job_identifiers* in the following format:
- 101201 *sequence_number.server_name@server*

STDERR

- 101202 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

- 101203 None.

EXTENDED DESCRIPTION

- 101204 None.

EXIT STATUS

- 101207 The following exit values shall be returned:

- 101208 0 Successful completion.
- 101209 >0 An error occurred.

CONSEQUENCES OF ERRORS

- 101210 Default.

APPLICATION USAGE

- 101211 None.

EXAMPLES

- 101212 The following example shows how a user might use the *qselect* utility in conjunction with the
101213 *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are
101214 already running:

- 101215 `qdel $(qselect -s q)`
- 101216 or:
- 101217 `qselect -s q || xargs qdel`

RATIONALE

- 101218 The *qselect* utility allows users to acquire a list of job identifiers that match user-specified
101219 selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of
101220 the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*. The *qselect* utility
101221 is thus a powerful tool for causing another batch system utility to act upon a set of jobs that
101222 match a list of selection criteria.

- 101223 The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs.
101224 Each option further restricts the selection of jobs. Many of the selection options allow the
101225 specification of a relational operator. The FORTRAN-like syntax of the operator—that is,
101226 ".lt."—was chosen rather than the C-like "<=" meta-characters.

- 101227 The *-a* option allows users to restrict the selected jobs to those that have been submitted (or
101228 altered) to wait until a particular time. The time period is determined by the argument of this

101236 option, which includes both a time and an operator—it is thus possible to select jobs waiting
 101237 until a specific time, jobs waiting until after a certain time, or those waiting for a time before the
 101238 specified time.

101239 The **-A** option allows users to restrict the selected jobs to those that have been submitted (or
 101240 altered) to charge a particular account.

101241 The **-c** option allows users to restrict the selected jobs to those whose checkpointing interval
 101242 falls within the specified range.

101243 The **-l** option allows users to select those jobs whose resource limits fall within the range
 101244 indicated by the value of the option. For example, a user could select those jobs for which the
 101245 CPU time limit is greater than two hours.

101246 The **-N** option allows users to select jobs by job name. For instance, all the parts of a task that
 101247 have been divided in parallel jobs might be given the same name, and thus manipulated as a
 101248 group by means of this option.

101249 The **-q** option allows users to select jobs in a specified queue.

101250 The **-r** option allows users to select only those jobs with a specified rerun criteria. For instance, a
 101251 user might select only those jobs that can be rerun for use with the *qrerun* utility.

101252 The **-s** option allows users to select only those jobs that are in a certain state.

101253 The **-u** option allows users to select jobs that have been submitted to execute under a particular
 101254 account.

101255 The selection criteria provided by the options of the *qselect* utility allow users to select jobs based
 101256 on all the appropriate attributes that can be assigned to jobs by the *qsub* utility.

101257 Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that
 101258 has been introduced in this volume of POSIX.1-200x.

101259 FUTURE DIRECTIONS

101260 The *qselect* utility may be removed in a future version.

101261 SEE ALSO

101262 [Chapter 3](#) (on page 2319), [qdel](#), [qrerun](#), [qrls](#), [qselect](#), [qsub](#), [touch](#)

101263 XBD [Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

101264 CHANGE HISTORY

101265 Derived from IEEE Std 1003.2d-1994.

101266 Issue 7

101267 The *qselect* utility is marked obsolescent.

101268 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101269 **NAME**

101270 qsig — signal batch jobs

101271 **SYNOPSIS**101272 OB BE `qsig [-s signal] job_identifier...`101273 **DESCRIPTION**

101274 To signal a batch job is to send a signal to the session leader of the batch job. A batch job is
 101275 signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a
 101276 user-accessible batch client that requests the signaling of a batch job.

101277 The *qsig* utility shall signal those batch jobs for which a batch *job_identifier* is presented to the
 101278 utility. The *qsig* utility shall not signal any batch jobs whose batch *job_identifiers* are not
 101279 presented to the utility.

101280 The *qsig* utility shall signal batch jobs in the order in which the corresponding batch
 101281 *job_identifiers* are presented to the utility. If the *qsig* utility fails to process a batch *job_identifier*
 101282 successfully, the utility shall proceed to process the remaining batch *job_identifiers*, if any.

101283 The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that
 101284 manages the batch job.

101285 For each successfully processed batch *job_identifier*, the *qsig* utility shall have received a
 101286 completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

101287 **OPTIONS**101288 The *qsig* utility shall conform to XBD [Section 12.2](#) (on page 201).

101289 The following option shall be supported by the implementation:

101290 `-s signal` Define the signal to be sent to the batch job.

101291 The *qsig* utility shall accept a *signal* option-argument that is either a symbolic signal
 101292 name or an unsigned integer signal number (see the POSIX.1-1990 standard,
 101293 Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix
 101294 has been omitted.

101295 If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

101296 If the *signal* option-argument is a number, the *qsig* utility shall send the signal
 101297 value represented by the number.

101298 If the `-s` option is not presented to the *qsig* utility, the utility shall send the signal
 101299 SIGTERM to each signaled batch job.

101300 **OPERANDS**

101301 The *qsig* utility shall accept one or more operands that conform to the syntax for a batch
 101302 *job_identifier* (see [Section 3.3.1](#), on page 2340).

101303 **STDIN**

101304 Not used.

101305 **INPUT FILES**

101306 None.

101307 **ENVIRONMENT VARIABLES**101308 The following environment variables shall affect the execution of *qsig*:101309 *LANG* Provide a default value for the internationalization variables that are unset or null. |
101310 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
101311 used to determine the values of locale categories.)101312 *LC_ALL* If set to a non-empty string value, override the values of all the other
101313 internationalization variables.101314 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
101315 characters (for example, single-byte as opposed to multi-byte characters in
101316 arguments).101317 *LC_MESSAGES*
101318 Determine the locale that should be used to affect the format and contents of
101319 diagnostic messages written to standard error.101320 *LOGNAME* Determine the login name of the user.101321 **ASYNCHRONOUS EVENTS**

101322 Default.

101323 **STDOUT**101324 An implementation of the *qsig* utility may write informative messages to standard output.101325 **STDERR**

101326 The standard error shall be used only for diagnostic messages.

101327 **OUTPUT FILES**

101328 None.

101329 **EXTENDED DESCRIPTION**

101330 None.

101331 **EXIT STATUS**

101332 The following exit values shall be returned:

101333 0 Successful completion.

101334 >0 An error occurred.

101335 **CONSEQUENCES OF ERRORS**101336 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic
101337 message to standard error when the error reply received from a batch server indicates that the
101338 batch *job_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the
101339 diagnostic message while attempting to locate the batch job on other servers is implementation-
101340 defined.101341 **APPLICATION USAGE**

101342 None.

101343 **EXAMPLES**

101344 None.

101345 **RATIONALE**101346 The *qsig* utility allows users to signal batch jobs.101347 A user may be unable to signal a batch job with the *kill* utility of the operating system for a
101348 number of reasons. First, the process ID of the batch job may be unknown to the user. Second,
101349 the processes of the batch job may be on a remote node. However, by virtue of communication
101350 between batch nodes, the *qsig* utility can arrange for the signaling of a process.

101351 Because a batch job that is not running cannot be signaled, and because the signal may not

101352 terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.

101353 The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch
101354 job.

101355 The *-s* option allows users to specify a signal by name or by number, and thus override the
101356 default signal. The POSIX.1-1990 standard defines signals by both name and number.

101357 The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of
101358 POSIX.1-200x in response to user-perceived shortcomings in existing practice.

101359 **FUTURE DIRECTIONS**

101360 The *qsig* utility may be removed in a future version.

101361 **SEE ALSO**

101362 [Chapter 3](#) (on page 2319), *kill*, *qdel*

101363 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

101364 **CHANGE HISTORY**

101365 Derived from IEEE Std 1003.2d-1994.

101366 **Issue 6**

101367 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

101368 **Issue 7**

101369 The *qsig* utility is marked obsolescent.

101370 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

101371 **NAME**
 101372 qstat — show status of batch jobs

101373 **SYNOPSIS**

101374 OB BE qstat [-f] *job_identifier...*
 101375 qstat -Q [-f] *destination...*
 101376 qstat -B [-f] *server_name...*

101377 **DESCRIPTION**

101378 The status of a batch job, batch queue, or batch server is obtained by a request to the server. The
 101379 *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs,
 101380 batch queues, or servers, and writes the status information to standard output.

101381 For each successfully processed batch *job_identifier*, the *qstat* utility shall display information
 101382 about the corresponding batch job.

101383 For each successfully processed destination, the *qstat* utility shall display information about the
 101384 corresponding batch queue.

101385 For each successfully processed server name, the *qstat* utility shall display information about the
 101386 corresponding server.

101387 The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a
 101388 batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue*
 101389 *Status Request* to a batch server. The *qstat* utility shall acquire server status information by
 101390 sending a *Server Status Request* to a batch server.

101391 **OPTIONS**

101392 The *qstat* utility shall conform to XBD [Section 12.2](#) (on page 201).

101393 The following options shall be supported by the implementation:

101394 **-f** Specify that a full display is produced.

101395 The minimum contents of a full display are specified in the `STDOUT` section.

101396 Additional contents and format of a full display are implementation-defined.

101397 **-Q** Specify that the operand is a destination.

101398 The *qstat* utility shall display information about each batch queue at each
 101399 destination identified as an operand.

101400 **-B** Specify that the operand is a server name.

101401 The *qstat* utility shall display information about each server identified as an
 101402 operand.

101403 **OPERANDS**

101404 If the **-Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that
 101405 conform to the syntax for a destination (see [Section 3.3.2](#), on page 2341).

101406 If the **-B** option is presented to the *qstat* utility, the utility shall accept one or more *server_name*
 101407 operands.

101408 If neither the **-B** nor the **-Q** option is presented to the *qstat* utility, the utility shall accept one or
 101409 more operands that conform to the syntax for a batch *job_identifier* (see [Section 3.3.1](#), on page
 101410 2340).

101411 **STDIN**

101412 Not used.

101413 **INPUT FILES**

101414 None.

101415 **ENVIRONMENT VARIABLES**101416 The following environment variables shall affect the execution of *qstat*:101417 *HOME* Determine the pathname of the user's home directory.101418 *LANG* Provide a default value for the internationalization variables that are unset or null. |
101419 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
101420 used to determine the values of locale categories.)101421 *LC_ALL* If set to a non-empty string value, override the values of all the other
101422 internationalization variables.101423 *LC_COLLATE*101424 Determine the locale for the behavior of ranges, equivalence classes, and multi-
101425 character collating elements within regular expressions.101426 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
101427 characters (for example, single-byte as opposed to multi-byte characters in
101428 arguments).101429 *LC_MESSAGES*101430 Determine the locale that should be used to affect the format and contents of
101431 diagnostic messages written to standard error.101432 *LC_NUMERIC*101433 Determine the locale for selecting the radix character used when writing floating-
101434 point formatted output.101435 **ASYNCHRONOUS EVENTS**

101436 Default.

101437 **STDOUT**101438 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is not
101439 specified, the *qstat* utility shall display the following items on a single line, in the stated order,
101440 with white space between each item, for each successfully processed operand:

- 101441
- The batch *job_identifier*
 - 101442 • The batch job name
 - 101443 • The *Job_Owner* attribute
 - 101444 • The CPU time used by the batch job
 - 101445 • The batch job state
 - 101446 • The batch job location

101447 If an operand presented to the *qstat* utility is a batch *job_identifier* and the *-f* option is specified,
101448 the *qstat* utility shall display the following items for each success fully processed operand:

- 101449
- The batch *job_identifier*
 - 101450 • The batch job name
 - 101451 • The *Job_Owner* attribute

- 101452 • The execution user ID
- 101453 • The CPU time used by the batch job
- 101454 • The batch job state
- 101455 • The batch job location
- 101456 • Additional implementation-defined information, if any, about the batch job or batch queue

101457 If an operand presented to the *qstat* utility is a destination, the **-Q** option is specified, and the **-f**
 101458 option is not specified, the *qstat* utility shall display the following items on a single line, in the
 101459 stated order, with white space between each item, for each successfully processed operand:

- 101460 • The batch queue name
- 101461 • The maximum number of batch jobs that shall be run in the batch queue concurrently
- 101462 • The total number of batch jobs in the batch queue
- 101463 • The status of the batch queue
- 101464 • For each state, the number of batch jobs in that state in the batch queue and the name of
 101465 the state
- 101466 • The type of batch queue (execution or routing)

101467 If the operands presented to the *qstat* utility are destinations, the **-Q** option is specified, and the
 101468 **-f** option is specified, the *qstat* utility shall display the following items for each successfully
 101469 processed operand:

- 101470 • The batch queue name
- 101471 • The maximum number of batch jobs that shall be run in the batch queue concurrently
- 101472 • The total number of batch jobs in the batch queue
- 101473 • The status of the batch queue
- 101474 • For each state, the number of batch jobs in that state in the batch queue and the name of
 101475 the state
- 101476 • The type of batch queue (execution or routing)
- 101477 • Additional implementation-defined information, if any, about the batch queue

101478 If the operands presented to the *qstat* utility are batch server names, the **-B** option is specified,
 101479 and the **-f** option is not specified, the *qstat* utility shall display the following items on a single
 101480 line, in the stated order, with white space between each item, for each successfully processed
 101481 operand:

- 101482 • The batch server name
- 101483 • The maximum number of batch jobs that shall be run in the batch queue concurrently
- 101484 • The total number of batch jobs managed by the batch server
- 101485 • The status of the batch server
- 101486 • For each state, the number of batch jobs in that state and the name of the state

101487 If the operands presented to the *qstat* utility are server names, the **-B** option is specified, and the
 101488 **-f** option is specified, the *qstat* utility shall display the following items for each successfully
 101489 processed operand:

- 101490 • The server name

- 101491 • The maximum number of batch jobs that shall be run in the batch queue concurrently
- 101492 • The total number of batch jobs managed by the server
- 101493 • The status of the server
- 101494 • For each state, the number of batch jobs in that state and the name of the state
- 101495 • Additional implementation-defined information, if any, about the server

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output the diagnostic message while attempting to locate the batch job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qstat* utility allows users to display the status of jobs and list the batch jobs in queues.

The operands of the *qstat* utility may be either job identifiers, queues (specified as destination identifiers), or batch server names. The **-Q** and **-B** options, or absence thereof, indicate the nature of the operands.

The other options of the *qstat* utility allow the user to control the amount of information displayed and the format in which it is displayed. Should a user wish to display the status of a set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.

The **-f** option allows users to request a “full” display in an implementation-defined format.

Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice on which it is based.

FUTURE DIRECTIONS

The *qstat* utility may be removed in a future version.

SEE ALSO

[Chapter 3](#) (on page 2319), *qselect*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

101532
101533
101534
101535
101536
101537
101538
101539
101540

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

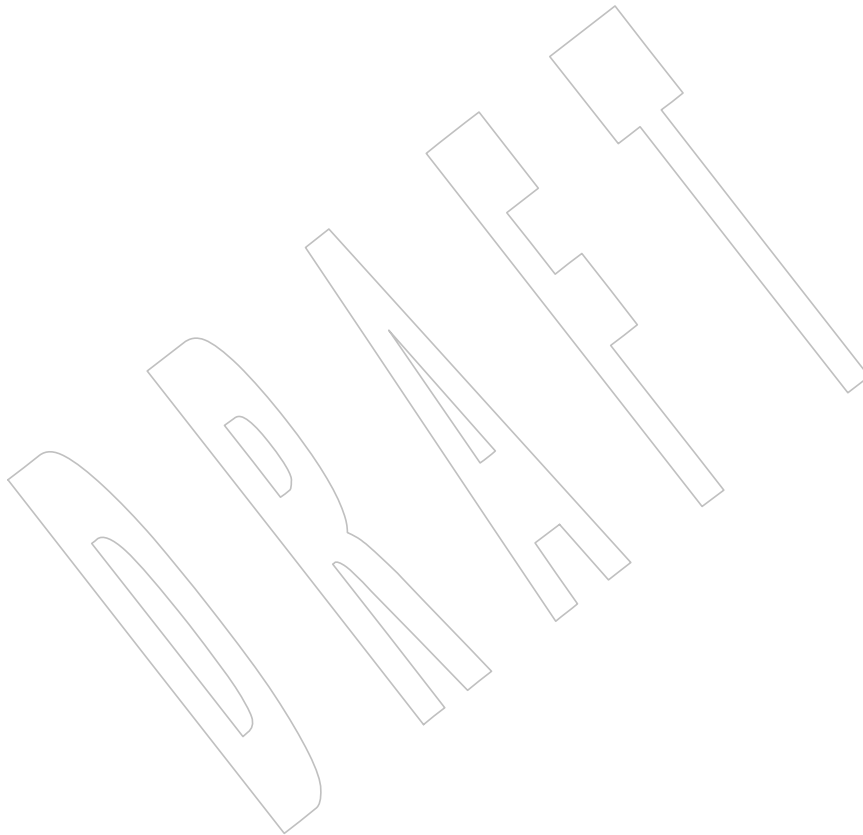
IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.

The *LC_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

Issue 7

The *qstat* utility is marked obsolescent.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



101541 **NAME**

101542 qsub — submit a script

101543 **SYNOPSIS**

```

101544 OB BE qsub [-a date_time] [-A account_string] [-c interval]
101545         [-C directive_prefix] [-e path_name] [-h] [-j join_list]
101546         [-k keep_list] [-m mail_options] [-M mail_list] [-N name]
101547         [-o path_name] [-p priority] [-q destination] [-r y|n]
101548         [-S path_name_list] [-u user_list] [-v variable_list] [-V]
101549         [-z] [script]

```

101550 **DESCRIPTION**

101551 To submit a script is to create a batch job that executes the script. A script is submitted by a
 101552 request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

101553 Upon successful completion, the *qsub* utility shall have created a batch job that will execute the
 101554 submitted script.

101555 The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

101556 The *qsub* utility shall place the value of the following environment variables in the *Variable_List*
 101557 attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name
 101558 of the environment variable shall be the current name prefixed with the string *PBS_O_*.

101559 **Note:** If the current value of the *HOME* variable in the environment space of the *qsub* utility is
 101560 */aa/bb/cc*, then *qsub* shall place *PBS_O_HOME=/aa/bb/cc* in the *Variable_List* attribute of the
 101561 batch job.

101562 In addition to the variables described above, the *qsub* utility shall add the following variables
 101563 with the indicated values to the variable list:

101564 *PBS_O_WORKDIR* The absolute path of the current working directory of the *qsub* utility
 101565 process.

101566 *PBS_O_HOST* The name of the host on which the *qsub* utility is running.

101567 **OPTIONS**

101568 The *qsub* utility shall conform to XBD [Section 12.2](#) (on page 201).

101569 The following options shall be supported by the implementation:

101570 **-a *date_time*** Define the time at which a batch job becomes eligible for execution.

101571 The *qsub* utility shall accept an option-argument that conforms to the syntax of the
 101572 *time* operand of the *touch* utility.

101573

Table 4-18 Environment Variable Values (Utilities)

101574

101575

101576

101577

101578

101579

101580

101581

101582

101583

Variable Name	Value at qsub Time
<i>PBS_O_HOME</i>	<i>HOME</i>
<i>PBS_O_HOST</i>	Client host name
<i>PBS_O_LANG</i>	<i>LANG</i>
<i>PBS_O_LOGNAME</i>	<i>LOGNAME</i>
<i>PBS_O_PATH</i>	<i>PATH</i>
<i>PBS_O_MAIL</i>	<i>MAIL</i>
<i>PBS_O_SHELL</i>	<i>SHELL</i>
<i>PBS_O_TZ</i>	<i>TZ</i>
<i>PBS_O_WORKDIR</i>	Current working directory

101584

101585

Note: The server that initiates execution of the batch job will add other variables to the batch job's environment; see [Section 3.2.2.1](#) (on page 2324).

101586

101587

101588

101589

The *qsub* utility shall set the *Execution_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date_time* option-argument. The Epoch is defined in XBD [Section 3.149](#) (on page 53).

101590

101591

101592

If the *-a* option is not presented to the *qsub* utility, the utility shall set the *Execution_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

101593

101594

101595

-A account_string

Define the account to which the resource consumption of the batch job should be charged.

101596

The syntax of the *account_string* option-argument is unspecified.

101597

101598

The *qsub* utility shall set the *Account_Name* attribute of the batch job to the value of the *account_string* option-argument.

101599

101600

If the *-A* option is not presented to the *qsub* utility, the utility shall omit the *Account_Name* attribute from the attributes of the batch job.

101601

-c interval

Define whether the batch job should be checkpointed, and if so, how often.

101602

101603

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

101604

101605

n No checkpointing shall be performed on the batch job (*NO_CHECKPOINT*).

101606

101607

s Checkpointing shall be performed only when the batch server is shut down (*CHECKPOINT_AT_SHUTDOWN*).

101608

101609

101610

c Automatic periodic checkpointing shall be performed at the *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU minutes (*CHECKPOINT_AT_MIN_CPU_INTERVAL*).

101611

101612

101613

101614

c=minutes Automatic periodic checkpointing shall be performed every *minutes* of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

101615

101616

The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the *interval* option-argument.

101617

If the *-c* option is not presented to the *qsub* utility, the utility shall set the *Checkpoint*

101618 attribute of the batch job to the single character 'u'
 101619 (CHECKPOINT_UNSPECIFIED).

101620 **-C** *directive_prefix*
 101621 Define the prefix that declares a directive to the *qsub* utility within the script.

101622 The *directive_prefix* is not a batch job attribute; it affects the behavior of the *qsub*
 101623 utility.

101624 If the **-C** option is presented to the *qsub* utility, and the value of the *directive_prefix*
 101625 option-argument is the null string, the utility shall not scan the script file for
 101626 directives. If the **-C** option is not presented to the *qsub* utility, then the value of the
 101627 *PBS_DPREFIX* environment variable is used. If the environment variable is not
 101628 defined, then #PBS encoded in the portable character set is the default.

101629 **-e** *path_name*
 101630 Define the path to be used for the standard error stream of the batch job.

101631 The *qsub* utility shall accept a *path_name* option-argument which can be preceded
 101632 by a host name element of the form *hostname*:

101633 If the *path_name* option-argument constitutes an absolute pathname, the *qsub* utility
 101634 shall set the *Error_Path* attribute of the batch job to the value of the *path_name*
 101635 option-argument.

101636 If the *path_name* option-argument constitutes a relative pathname and no host
 101637 name element is specified, the *qsub* utility shall set the *Error_Path* attribute of the
 101638 batch job to the value of the absolute pathname derived by expanding the
 101639 *path_name* option-argument relative to the current directory of the process
 101640 executing *qsub*.

101641 If the *path_name* option-argument constitutes a relative pathname and a host name
 101642 element is specified, the *qsub* utility shall set the *Error_Path* attribute of the batch
 101643 job to the value of the *path_name* option-argument without expansion. The host
 101644 name element shall be included.

101645 If the *path_name* option-argument does not include a host name element, the *qsub*
 101646 utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the
 101647 host upon which the *qsub* utility is being executed.

101648 If the **-e** option is not presented to the *qsub* utility, the utility shall set the
 101649 *Error_Path* attribute of the batch job to the host name and path of the current
 101650 directory of the submitting process and the default filename.

101651 The default filename for standard error has the following format:
 101652 *job_name*.*esequence_number*

101653 **-h** Specify that a USER hold is applied to the batch job.

101654 The *qsub* utility shall set the value of the *Hold_Types* attribute of the batch job to the
 101655 value USER.

101656 If the **-h** option is not presented to the *qsub* utility, the utility shall set the
 101657 *Hold_Types* attribute of the batch job to the value NO_HOLD.

101658 **-j** *join_list* Define which streams of the batch job are to be merged. The *qsub* **-j** option shall
 101659 accept a value for the *join_list* option-argument that is a string of alphanumeric
 101660 characters in the portable character set (see XBD Section 6.1, on page 111).

101661 The *qsub* utility shall accept a *join_list* option-argument that consists of one or more
 101662 of the characters 'e' and 'o', or the single character 'n'.

101663 All of the other batch job output streams specified will be merged into the output
 101664 stream represented by the character listed first in the *join_list* option-argument.

101665 For each unique character in the *join_list* option-argument, the *qsub* utility shall
 101666 add a value to the *Join_Path* attribute of the batch job as follows, each representing
 101667 a different batch job stream to join:

- 101668 e The standard error of the batch job (JOIN_STD_ERROR).
- 101669 o The standard output of the batch job (JOIN_STD_OUTPUT).

101670 An existing *Join_Path* attribute can be cleared by the following join type:

- 101671 n NO_JOIN

101672 If 'n' is specified, then no files are joined. The *qsub* utility shall consider it an error
 101673 if any join type other than 'n' is combined with join type 'n'.

101674 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 101675 'n' within the *join_list* option-argument. The *qsub* utility shall permit the
 101676 repetition of characters, but shall not assign additional meaning to the repeated
 101677 characters.

101678 An implementation may define other join types. The conformance document for an
 101679 implementation shall describe any additional batch job streams, how they are
 101680 specified, their internal behavior, and how they affect the behavior of the utility.

101681 If the *-j* option is not presented to the *qsub* utility, the utility shall set the value of
 101682 the *Join_Path* attribute of the batch job to NO_JOIN.

101683 *-k keep_list* Define which output of the batch job to retain on the execution host.

101684 The *qsub -k* option shall accept a value for the *keep_list* option-argument that is a
 101685 string of alphanumeric characters in the portable character set (see XBD [Section](#)
 101686 [6.1](#), on page 111).

101687 The *qsub* utility shall accept a *keep_list* option-argument that consists of one or
 101688 more of the characters 'e' and 'o', or the single character 'n'.

101689 For each unique character in the *keep_list* option-argument, the *qsub* utility shall
 101690 add a value to the *Keep_Files* attribute of the batch job as follows, each representing
 101691 a different batch job stream to keep:

- 101692 e The standard error of the batch job (KEEP_STD_ERROR).
- 101693 o The standard output of the batch job (KEEP_STD_OUTPUT).

101694 If both 'e' and 'o' are specified, then both files are retained. An existing
 101695 *Keep_Files* attribute can be cleared by the following keep type:

- 101696 n NO_KEEP

101697 If 'n' is specified, then no files are retained. The *qsub* utility shall consider it an
 101698 error if any keep type other than 'n' is combined with keep type 'n'.

101699 Strictly conforming applications shall not repeat any of the characters 'e', 'o', or
 101700 'n' within the *keep_list* option-argument. The *qsub* utility shall permit the
 101701 repetition of characters, but shall not assign additional meaning to the repeated
 101702 characters.

101703 An implementation may define other keep types. The conformance document for
 101704 an implementation shall describe any additional keep types, how they are
 101705 specified, their internal behavior, and how they affect the behavior of the utility. If
 101706 the *-k* option is not presented to the *qsub* utility, the utility shall set the *Keep_Files*

101707 attribute of the batch job to the value NO_KEEP.

101708 **-m** *mail_options*

101709 Define the points in the execution of the batch job at which the batch server that
101710 manages the batch job shall send mail about a change in the state of the batch job.

101711 The *qsub* **-m** option shall accept a value for the *mail_options* option-argument that
101712 is a string of alphanumeric characters in the portable character set (see XBD Section
101713 6.1, on page 111).

101714 The *qsub* utility shall accept a value for the *mail_options* option-argument that is a
101715 string of one or more of the characters 'e', 'b', and 'a', or the single character
101716 'n'.

101717 For each unique character in the *mail_options* option-argument, the *qsub* utility shall
101718 add a value to the *Mail_Users* attribute of the batch job as follows, each
101719 representing a different time during the life of a batch job at which to send mail:

101720 e MAIL_AT_EXIT

101721 b MAIL_AT_BEGINNING

101722 a MAIL_AT_ABORT

101723 If any of these characters are duplicated in the *mail_options* option-argument, the
101724 duplicates shall be ignored.

101725 An existing *Mail_Points* attribute can be cleared by the following mail type:

101726 n NO_MAIL

101727 If 'n' is specified, then mail is not sent. The *qsub* utility shall consider it an error if
101728 any mail type other than 'n' is combined with mail type 'n'.

101729 Strictly conforming applications shall not repeat any of the characters 'e', 'b',
101730 'a', or 'n' within the *mail_options* option-argument.

101731 The *qsub* utility shall permit the repetition of characters, but shall not assign
101732 additional meaning to the repeated characters. An implementation may define
101733 other mail types. The conformance document for an implementation shall describe
101734 any additional mail types, how they are specified, their internal behavior, and how
101735 they affect the behavior of the utility.

101736 If the **-m** option is not presented to the *qsub* utility, the utility shall set the
101737 *Mail_Points* attribute to the value MAIL_AT_ABORT.

101738 **-M** *mail_list* Define the list of users to which a batch server that executes the batch job shall
101739 send mail, if the server sends mail about the batch job.

101740 The syntax of the *mail_list* option-argument is unspecified.

101741 If the implementation of the *qsub* utility uses a name service to locate users, the
101742 utility should accept the syntax used by the name service.

101743 If the implementation of the *qsub* utility does not use a name service to locate users,
101744 the implementation should accept the following syntax for user names:

101745 *mail_address*[, *mail_address* , , . . .]

101746 The interpretation of *mail_address* is implementation-defined.

101747 The *qsub* utility shall set the *Mail_Users* attribute of the batch job to the value of the
101748 *mail_list* option-argument.

101749 If the **-M** option is not presented to the *qsub* utility, the utility shall place only the

101750 user name and host name for the current process in the *Mail_Users* attribute of the
 101751 batch job.

101752 **-N name** Define the name of the batch job.

101753 The *qsub* **-N** option shall accept a value for the *name* option-argument that is a
 101754 string of up to 15 alphanumeric characters in the portable character set (see XBD
 101755 [Section 6.1](#), on page 111) where the first character is alphabetic.

101756 The *qsub* utility shall set the value of the *Job_Name* attribute of the batch job to the
 101757 value of the *name* option-argument.

101758 If the **-N** option is not presented to the *qsub* utility, the utility shall set the *Job_Name*
 101759 attribute of the batch job to the name of the *script* argument from which the
 101760 directory specification if any, has been removed.

101761 If the **-N** option is not presented to the *qsub* utility, and the script is read from
 101762 standard input, the utility shall set the *Job_Name* attribute of the batch job to the
 101763 value STDIN.

101764 **-o path_name** Define the path for the standard output of the batch job.

101765 The *qsub* utility shall accept a *path_name* option-argument that conforms to the
 101766 syntax of the *path_name* element defined in the System Interfaces volume of
 101767 POSIX.1-200x, which can be preceded by a host name element of the form
 101768 *hostname:*.
 101769

101770 If the *path_name* option-argument constitutes an absolute pathname, the *qsub* utility
 101771 shall set the *Output_Path* attribute of the batch job to the value of the *path_name*
 101772 option-argument without expansion.

101773 If the *path_name* option-argument constitutes a relative pathname and no host
 101774 name element is specified, the *qsub* utility shall set the *Output_Path* attribute of the
 101775 batch job to the pathname derived by expanding the value of the *path_name* option-
 101776 argument relative to the current directory of the process executing the *qsub*.

101777 If the *path_name* option-argument constitutes a relative pathname and a host name
 101778 element is specified, the *qsub* utility shall set the *Output_Path* attribute of the batch
 101779 job to the value of the *path_name* option-argument without expansion.

101780 If the *path_name* option-argument does not specify a host name element, the *qsub*
 101781 utility shall prefix the pathname with *hostname:*, where *hostname* is the name of the
 101782 host upon which the *qsub* utility is executing.

101783 If the **-o** option is not presented to the *qsub* utility, the utility shall set the
 101784 *Output_Path* attribute of the batch job to the host name and path of the current
 101785 directory of the submitting process and the default filename.

101786 The default filename for standard output has the following format:
 101787 *job_name.osequence_number*

101788 **-p priority** Define the priority the batch job should have relative to other batch jobs owned by
 101789 the batch server.

101790 The *qsub* utility shall set the *Priority* attribute of the batch job to the value of the
 101791 *priority* option-argument.

101792 If the **-p** option is not presented to the *qsub* utility, the value of the *Priority* attribute
 101793 is implementation-defined.

101794 The *qsub* utility shall accept a value for the *priority* option-argument that conforms

- 101795 to the syntax for signed decimal integers, and which is not less than $-1\,024$ and not
101796 greater than $1\,023$.
- 101797 **-q** *destination*
101798 Define the destination of the batch job.
- 101799 The destination is not a batch job attribute; it determines the batch server, and
101800 possibly the batch queue, to which the *qsub* utility batch queues the batch job.
- 101801 The *qsub* utility shall submit the script to the batch server named by the *destination*
101802 option-argument or the server that owns the batch queue named in the *destination*
101803 option-argument.
- 101804 The *qsub* utility shall accept an option-argument for the **-q** option that conforms to
101805 the syntax for a destination (see [Section 3.3.2](#), on page 2341).
- 101806 If the **-q** option is not presented to the *qsub* utility, the *qsub* utility shall submit the
101807 batch job to the default destination. The mechanism for determining the default
101808 destination is implementation-defined.
- 101809 **-r** *y|n*
101810 Define whether the batch job is rerunnable.
- 101811 If the value of the option-argument is *y*, the *qsub* utility shall set the *Rerunable*
101812 attribute of the batch job to TRUE.
- 101813 If the value of the option-argument is *n*, the *qsub* utility shall set the *Rerunable*
101814 attribute of the batch job to FALSE.
- 101815 If the **-r** option is not presented to the *qsub* utility, the utility shall set the *Rerunable*
101816 attribute of the batch job to TRUE.
- 101817 **-S** *path_name_list*
101818 Define the pathname to the shell under which the batch job is to execute.
- 101819 The *qsub* utility shall accept a *path_name_list* option-argument that conforms to the
101820 following syntax:
pathname[@host][, , *pathname*[@host], , . . .]
- 101821 The *qsub* utility shall allow only one pathname for a given host name. The *qsub*
101822 utility shall allow only one pathname that is missing a corresponding host name.
- 101823 The *qsub* utility shall add a value to the *Shell_Path_List* attribute of the batch job for
101824 each entry in the *path_name_list* option-argument.
- 101825 If the **-S** option is not presented to the *qsub* utility, the utility shall set the
101826 *Shell_Path_List* attribute of the batch job to the null string.
- 101827 The conformance document for an implementation shall describe the mechanism
101828 used to set the default shell and determine the current value of the default shell.
101829 An implementation shall provide a means for the installation to set the default
101830 shell to the login shell of the user under which the batch job is to execute. See
101831 [Section 3.3.3](#) (on page 2341) for a means of removing *keyword=value* (and
101832 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.
- 101833 **-u** *user_list*
101834 Define the user name under which the batch job is to execute.
- 101835 The *qsub* utility shall accept a *user_list* option-argument that conforms to the
101836 following syntax:
username[@host][, , *username*[@host], , . . .]
- 101837 The *qsub* utility shall accept only one user name that is missing a corresponding
101838 host name. The *qsub* utility shall accept only one user name per named host.

101839 The *qsub* utility shall add a value to the *User_List* attribute of the batch job for each
101840 entry in the *user_list* option-argument.

101841 If the *-u* option is not presented to the *qsub* utility, the utility shall set the *User_List*
101842 attribute of the batch job to the user name from which the utility is executing. See
101843 [Section 3.3.3](#) (on page 2341) for a means of removing *keyword=value* (and
101844 *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

101845 *-v variable_list*

101846 Add to the list of variables that are exported to the session leader of the batch job.

101847 A *variable_list* is a set of strings of either the form *<variable>* or *<variable=value>*,
101848 delimited by commas.

101849 If the *-v* option is presented to the *qsub* utility, the utility shall also add, to the
101850 environment *Variable_List* attribute of the batch job, every variable named in the
101851 environment *variable_list* option-argument and, optionally, values of specified
101852 variables.

101853 If a value is not provided on the command line, the *qsub* utility shall set the value
101854 of each variable in the environment *Variable_List* attribute of the batch job to the
101855 value of the corresponding environment variable for the process in which the
101856 utility is executing; see [Table 4-18](#) (on page 3029).

101857 A conforming application shall not repeat a variable in the environment
101858 *variable_list* option-argument.

101859 The *qsub* utility shall not repeat a variable in the environment *Variable_List* attribute
101860 of the batch job. See [Section 3.3.3](#) (on page 2341) for a means of removing
101861 *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented
101862 batch job attributes.

101863 *-V* Specify that all of the environment variables of the process are exported to the
101864 context of the batch job.

101865 The *qsub* utility shall place every environment variable in the process in which the
101866 utility is executing in the list and shall set the value of each variable in the attribute
101867 to the value of that variable in the process.

101868 *-z* Specify that the utility does not write the batch *job_identifier* of the created batch job
101869 to standard output.

101870 If the *-z* option is presented to the *qsub* utility, the utility shall not write the batch
101871 *job_identifier* of the created batch job to standard output.

101872 If the *-z* option is not presented to the *qsub* utility, the utility shall write the
101873 identifier of the created batch job to standard output.

101874 OPERANDS

101875 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.

101876 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character
101877 string *'-'*, the utility shall read the script from standard input.

101878 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current
101879 directory of the process executing the utility.

101880 STDIN

101881 The *qsub* utility reads the script of the batch job from standard input if the script operand is
101882 omitted or is the single character *'-'*.

101883 **INPUT FILES**

101884 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility
 101885 reads the script file and acts on directives in the script.

101886 **ENVIRONMENT VARIABLES**

101887 The following environment variables shall affect the execution of *qsub*:

101888 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 101889 (See XBD Section 8.2 (on page 160) the precedence of internationalization variables |
 101890 used to determine the values of locale categories.)

101891 *LC_ALL* If set to a non-empty string value, override the values of all the other
 101892 internationalization variables.

101893 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 101894 characters (for example, single-byte as opposed to multi-byte characters in
 101895 arguments).

101896 *LC_MESSAGES*

101897 Determine the locale that should be used to affect the format and contents of
 101898 diagnostic messages written to standard error.

101899 *LOGNAME* Determine the login name of the user.

101900 *PBS_DPREFIX*

101901 Determine the default prefix for directives within the script.

101902 *SHELL* Determine the pathname of the preferred command language interpreter of the
 101903 user.

101904 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
 101905 unset or null, an unspecified default timezone shall be used.

101906 **ASYNCHRONOUS EVENTS**

101907 Once created, a batch job exists until it exits, aborts, or is deleted.

101908 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or
 101909 delete the batch job.

101910 **STDOUT**

101911 The *qsub* utility writes the batch *job_identifier* assigned to the batch job to standard output, unless
 101912 the *-z* option is specified.

101913 **STDERR**

101914 The standard error shall be used only for diagnostic messages.

101915 **OUTPUT FILES**

101916 None.

101917 **EXTENDED DESCRIPTION**101918 **Script Preservation**

101919 The *qsub* utility shall make the script available to the server executing the batch job in such a
 101920 way that the server executes the script as it exists at the time of submission.

101921 The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a
 101922 temporary copy of the script in a location specified to the server.

101923 **Option Specification**101924 A script can contain directives to the *qsub* utility.101925 The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first
101926 line that begins with a string other than the directive string; if directives occur on subsequent
101927 lines, the utility shall ignore those directives.101928 Lines are separated by a <newline>. If the first line of the script begins with "#!" or a colon
101929 (' : '), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and
101930 only if the string of characters from the first non-white-space character on the line until the first
101931 <space> or <tab> on the line match the directive prefix. If a line in the script contains a directive
101932 and the final characters of the line are backslash ('\ ') and <newline>, then the next line shall be
101933 interpreted as a continuation of that directive.101934 The *qsub* utility shall process the options and option-arguments contained on the directive prefix
101935 line using the same syntax as if the options were input on the *qsub* utility.101936 The *qsub* utility shall continue to process a directive prefix line until after a <newline> is
101937 encountered. An implementation may ignore lines which, according to the syntax of the shell
101938 that will interpret the script, are comments. An implementation shall describe in the
101939 conformance document the format of any shell comments that it will recognize.101940 If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall
101941 ignore the option and the corresponding option-argument, if any, in the directive.101942 If an option that is present in the directive is not present in the arguments to the *qsub* utility, the
101943 utility shall process the option and the option-argument, if any.101944 In order of preference, the *qsub* utility shall select the directive prefix from one of the following
101945 sources:

- 101946
- If the `-C` option is presented to the utility, the value of the *directive_prefix* option-argument
 - If the environment variable *PBS_DPREFIX* is defined, the value of that variable
 - The four-character string "#PBS" encoded in the portable character set

101949 If the `-C` option is present in the script file it shall be ignored.101950 **EXIT STATUS**

101951 The following exit values shall be returned:

101952 0 Successful completion.

101953 >0 An error occurred.

101954 **CONSEQUENCES OF ERRORS**

101955 Default.

101956 **APPLICATION USAGE**

101957 None.

101958 **EXAMPLES**

101959 None.

101960 **RATIONALE**101961 The *qsub* utility allows users to create a batch job that will process the script specified as the
101962 operand of the utility.101963 The options of the *qsub* utility allow users to control many aspects of the queuing and execution
101964 of a batch job.101965 The `-a` option allows users to designate the time after which the batch job will become eligible to
101966 run. By specifying an execution time, users can take advantage of resources at off-peak hours,

101967 synchronize jobs with chronologically predictable events, and perhaps take advantage of off-
 101968 peak pricing of computing time. For these reasons and others, a timing option is existing
 101969 practice on the part of almost every batch system, including NQS.

101970 The **-A** option allows users to specify the account that will be charged for the batch job. Support
 101971 for account is not mandatory for conforming batch servers.

101972 The **-C** option allows users to prescribe the prefix for directives within the script file. The default
 101973 prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as
 101974 specified by the **-S** option.

101975 The **-c** option allows users to establish the checkpointing interval for their jobs. A checkpointing
 101976 system, which is not defined by this volume of POSIX.1-200x, allows recovery of a batch job at
 101977 the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that
 101978 consume expensive computing time or must meet a critical schedule. Users should be allowed to
 101979 make the tradeoff between the overhead of checkpointing and the risk to the timely completion
 101980 of the batch job; therefore, this volume of POSIX.1-200x provides the checkpointing interval
 101981 option. Support for checkpointing is optional for batch servers.

101982 The **-e** option allows users to redirect the standard error streams of their jobs to a non-default
 101983 path. For example, if the submitted script generally produces a great deal of useless error
 101984 output, a user might redirect the standard error output to the null device. Or, if the file system
 101985 holding the default location (the home directory of the user) has too little free space, the user
 101986 might redirect the standard error stream to a file in another file system.

101987 The **-h** option allows users to create a batch job that is held until explicitly released. The ability
 101988 to create a held job is useful when some external event must complete before the batch job can
 101989 execute. For example, the user might submit a held job and release it when the system load has
 101990 dropped.

101991 The **-j** option allows users to merge the standard error of a batch job into its standard output
 101992 stream, which has the advantage of showing the sequential relationship between output and
 101993 error messages.

101994 The **-m** option allows users to designate those points in the execution of a batch job at which
 101995 mail will be sent to the submitting user, or to the account(s) indicated by the **-M** option. By
 101996 requesting mail notification at points of interest in the life of a job, the submitting user, or other
 101997 designated users, can track the progress of a batch job.

101998 The **-N** option allows users to associate a name with the batch job. The job name in no way
 101999 affects the processing of the batch job, but rather serves as a mnemonic handle for users. For
 102000 example, the batch job name can help the user distinguish between multiple jobs listed by the
 102001 *qstat* utility.

102002 The **-o** option allows users to redirect the standard output stream. A user might, for example,
 102003 wish to redirect to the null device the standard output stream of a job that produces copious yet
 102004 superfluous output.

102005 The **-P** option allows users to designate the relative priority of a batch job for selection from a
 102006 queue.

102007 The **-q** option allows users to specify an initial queue for the batch job. If the user specifies a
 102008 routing queue, the batch server routes the batch job to another queue for execution or further
 102009 routing. If the user specifies a non-routing queue, the batch server of the queue eventually
 102010 executes the batch job.

102011 The **-r** option allows users to control whether the submitted job will be rerun if the controlling
 102012 batch node fails during execution of the batch job. The **-r** option likewise allows users to
 102013 indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot
 102014 be correctly rerun because of changes they make in the state of databases or other aspects of

102015 their environment. This volume of POSIX.1-200x specifies that the default, if the `-r` option is not
 102016 presented to the utility, will be that the batch job cannot be rerun, since the result of rerunning a
 102017 non-rerunnable job might be catastrophic.

102018 The `-S` option allows users to specify the program (usually a shell) that will be invoked to
 102019 process the script of the batch job. This option has been modified to allow a list of shell names
 102020 and locations associated with different hosts.

102021 The `-u` option is useful when the submitting user is authorized to use more than one account on
 102022 a given host, in which case the `-u` option allows the user to select from among those accounts.
 102023 The option-argument is a list of user-host pairs, so that the submitting user can provide different
 102024 user identifiers for different nodes in the event the batch job is routed. The `-u` option provides a
 102025 lot of flexibility to accommodate sites with complex account structures. Users that have the same
 102026 user identifier on all the hosts they are authorized to use will not need to use the `-u` option.

102027 The `-V` option allows users to export all their current environment variables, as of the time the
 102028 batch job is submitted, to the context of the processes of the batch job.

102029 The `-v` option allows users to export specific environment variables from their current process to
 102030 the processes of the batch job.

102031 The `-z` option allows users to suppress the writing of the batch job identifier to standard output.
 102032 The `-z` option is an existing NQS practice that has been standardized.

102033 Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the
 102034 existing practice on which it is based. Some changes and additions have been made to the *qsub*
 102035 utility in this volume of POSIX.1-200x, *vis-a-vis* NQS, as a result of the growing pool of
 102036 experience with distributed batch systems.

102037 The set of features of the *qsub* utility as defined in this volume of POSIX.1-200x appears to
 102038 incorporate all the common existing practice on potentially conforming platforms.

102039 FUTURE DIRECTIONS

102040 The *qsub* utility may be removed in a future version.

102041 SEE ALSO

102042 [Chapter 3](#) (on page 2319), [*qrerun*](#), [*qstat*](#), [*touch*](#)

102043 XBD [Section 3.149](#) (on page 53), [Section 6.1](#) (on page 111), [Chapter 8](#) (on page 159), [Section 12.2](#) +
 102044 (on page 201)

102045 CHANGE HISTORY

102046 Derived from IEEE Std 1003.2d-1994.

102047 Issue 6

102048 The `-I` option has been removed as there is no portable description of the resources that are
 102049 allowed or required by the batch job.

102050 Issue 7

102051 The *qsub* utility is marked obsolescent.

102052 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

102053 **NAME**
 102054 read — read a line from standard input

102055 **SYNOPSIS**
 102056 read [-r] var...

102057 **DESCRIPTION**
 102058 The *read* utility shall read a single line from standard input.

102059 By default, unless the *-r* option is specified, backslash ('**') shall act as an escape character, as
 102060 described in [Section 2.2.1](#) (on page 2246). If standard input is a terminal device and the invoking
 102061 shell is interactive, *read* shall prompt for a continuation line when:

- 102062 • The shell reads an input line ending with a backslash <newline>, unless the *-r* option is
 102063 specified.
- 102064 • A here-document is not terminated after a <newline> is entered.

102065 The terminating <newline> (if any) shall be removed from the input and the results shall be split
 102066 into fields as in the shell for the results of parameter expansion (see [Section 2.6.5](#), on page 2258);
 102067 the first field shall be assigned to the first variable *var*, the second field to the second variable
 102068 *var*, and so on. If there are fewer fields than there are *var* operands, the remaining *vars* shall be
 102069 set to empty strings. If there are fewer *vars* than fields, the last *var* shall be set to a value
 102070 comprising the following elements:

- 102071 • The field that corresponds to the last *var* in the normal assignment sequence described
 102072 above
- 102073 • The delimiter(s) that follow the field corresponding to the last *var*
- 102074 • The remaining fields and their delimiters, with trailing *IFS* white space ignored

102075 The setting of variables specified by the *var* operands shall affect the current shell execution
 102076 environment; see [Section 2.12](#) (on page 2277). If it is called in a subshell or separate utility
 102077 execution environment, such as one of the following:

```
102078 (read foo)
102079 nohup read ...
102080 find . -exec read ... \;
```

102081 it shall not affect the shell variables in the caller's environment.

102082 **OPTIONS**
 102083 The *read* utility shall conform to XBD [Section 12.2](#) (on page 201).

102084 The following option is supported:

- 102085 *-r* Do not treat a backslash character in any special way. Consider each backslash to
 102086 be part of the input line.

102087 **OPERANDS**
 102088 The following operand shall be supported:
 102089 *var* The name of an existing or nonexisting shell variable.

102090 **STDIN**
 102091 The standard input shall be a text file.

102092 **INPUT FILES**

102093 None.

102094 **ENVIRONMENT VARIABLES**102095 The following environment variables shall affect the execution of *read*:102096 *IFS* Determine the internal field separators used to delimit fields; see [Section 2.5.3](#) (on
102097 page 2250).102098 *LANG* Provide a default value for the internationalization variables that are unset or null. |
102099 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
102100 variables used to determine the values of locale categories.)102101 *LC_ALL* If set to a non-empty string value, override the values of all the other
102102 internationalization variables.102103 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
102104 characters (for example, single-byte as opposed to multi-byte characters in
102105 arguments).102106 *LC_MESSAGES*102107 Determine the locale that should be used to affect the format and contents of
102108 diagnostic messages written to standard error.102109 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.102110 *PS2* Provide the prompt string that an interactive shell shall write to standard error
102111 when a line ending with a backslash <newline> is read and the *-r* option was not
102112 specified, or if a here-document is not terminated after a <newline> is entered.102113 **ASYNCHRONOUS EVENTS**

102114 Default.

102115 **STDOUT**

102116 Not used.

102117 **STDERR**

102118 The standard error shall be used for diagnostic messages and prompts for continued input.

102119 **OUTPUT FILES**

102120 None.

102121 **EXTENDED DESCRIPTION**

102122 None.

102123 **EXIT STATUS**

102124 The following exit values shall be returned:

102125 0 Successful completion.

102126 >0 End-of-file was detected or an error occurred.

102127 **CONSEQUENCES OF ERRORS**

102128 Default.

APPLICATION USAGE

The `-r` option is included to enable `read` to subsume the purpose of the `line` utility, which is not included in POSIX.1-200x.

EXAMPLES

The following command:

```
while read -r xx yy
do
    printf "%s %s\n" "$yy" "$xx"
done < input_file
```

prints a file with the first field of each line moved to the end of the line.

RATIONALE

The `read` utility historically has been a shell built-in. It was separated off into its own utility to take advantage of the richer description of functionality introduced by this volume of POSIX.1-200x.

Since `read` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(read foo)
nohup read ...
find . -exec read ... \;
```

it does not affect the shell variables in the environment of the caller.

Although the standard input is required to be a text file, and therefore will always end with a `<newline>` (unless it is an empty file), the processing of continuation lines when the `-r` option is not used can result in the input not ending with a `<newline>`. This occurs if the last line of the input file ends with backslash `<newline>`. It is for this reason that “if any” is used in “The terminating `<newline>` (if any) shall be removed from the input” in the description. It is not a relaxation of the requirement for standard input to be a text file.

FUTURE DIRECTIONS

None.

SEE ALSO

[Chapter 2](#) (on page 2245)

[XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

CHANGE HISTORY

First released in Issue 2.

Issue 7

SD5-XCU-ERN-126 is applied, clarifying that input lines end with a `<newline>`.

102165 **NAME**
 102166 renice — set nice values of running processes

102167 **SYNOPSIS**
 102168 renice [-g|-p|-u] -n *increment* *ID*...

102169 **DESCRIPTION**
 102170 The *renice* utility shall request that the nice values (see XBD Section 3.238, on page 66) of one or
 102171 more running processes be changed. By default, the applicable processes are specified by their
 102172 process IDs. When a process group is specified (see *-g*), the request shall apply to all processes
 102173 in the process group.

102174 The nice value shall be bounded in an implementation-defined manner. If the requested
 102175 *increment* would raise or lower the nice value of the executed utility beyond implementation-
 102176 defined limits, then the limit whose value was exceeded shall be used.

102177 When a user is *reniced*, the request applies to all processes whose saved set-user-ID matches the
 102178 user ID corresponding to the user.

102179 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values
 102180 of any process unless the user requesting such a change has appropriate privileges to do so for
 102181 the specified process. If the user lacks appropriate privileges to perform the requested action, the
 102182 utility shall return an error status.

102183 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when
 102184 *renice* attempts to determine the user ID of the process in order to determine whether the user
 102185 has appropriate privileges.

102186 **OPTIONS**
 102187 The *renice* utility shall conform to XBD Section 12.2 (on page 201), except for Guideline 9.

102188 The following options shall be supported:

- 102189 **-g** Interpret the following operands as unsigned decimal integer process group IDs.
- 102190 **-n *increment*** Specify how the nice value of the specified process or processes is to be adjusted.
 102191 The *increment* option-argument is a positive or negative decimal integer that shall
 102192 be used to modify the nice value of the specified process or processes.
- 102193 Positive *increment* values shall cause a lower nice value. Negative *increment* values
 102194 may require appropriate privileges and shall cause a higher nice value.
- 102195 **-p** Interpret the following operands as unsigned decimal integer process IDs. The **-p**
 102196 option is the default if no options are specified.
- 102197 **-u** Interpret the following operands as users. If a user exists with a user name equal to
 102198 the operand, then the user ID of that user is used in further processing. Otherwise,
 102199 if the operand represents an unsigned decimal integer, it shall be used as the
 102200 numeric user ID of the user.

102201 **OPERANDS**
 102202 The following operands shall be supported:

102203 ***ID*** A process ID, process group ID, or user name/user ID, depending on the option
 102204 selected.

renice

Utilities

102205 **STDIN**

102206 Not used.

102207 **INPUT FILES**

102208 None.

102209 **ENVIRONMENT VARIABLES**102210 The following environment variables shall affect the execution of *renice*:

102211 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 102212 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 102213 variables used to determine the values of locale categories.)

102214 *LC_ALL* If set to a non-empty string value, override the values of all the other
 102215 internationalization variables.

102216 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 102217 characters (for example, single-byte as opposed to multi-byte characters in
 102218 arguments).

102219 *LC_MESSAGES*

102220 Determine the locale that should be used to affect the format and contents of
 102221 diagnostic messages written to standard error.

102222 *NSI* *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

102223 **ASYNCHRONOUS EVENTS**

102224 Default.

102225 **STDOUT**

102226 Not used.

102227 **STDERR**

102228 The standard error shall be used only for diagnostic messages.

102229 **OUTPUT FILES**

102230 None.

102231 **EXTENDED DESCRIPTION**

102232 None.

102233 **EXIT STATUS**

102234 The following exit values shall be returned:

102235 0 Successful completion.

102236 >0 An error occurred.

102237 **CONSEQUENCES OF ERRORS**

102238 Default.

102239 **APPLICATION USAGE**

102240 None.

102241 **EXAMPLES**

102242 1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

102243 `renice -n 5 -p 987 32`

102244 2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the
 102245 user has the appropriate privileges to do so:

102246 `renice -n -4 -g 324 76`

3. Adjust the nice value so that numeric user ID 8 and user `sas` would have a lower nice value:

```
renice -n 4 -u 8 sas
```

Useful nice value increments on historical systems include 19 or 20 (the affected processes run only when nothing else in the system attempts to run) and any negative number (to make processes run faster).

RATIONALE

The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-argument. However, for clarity, they have been included in the OPTIONS section, rather than the OPERANDS section.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of POSIX.1-200x make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of POSIX.1-200x, and others may have no effect at all. Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of implementation strategies are possible.

Originally, this utility was written in the historical manner, using the term “nice value”. This was always a point of concern with users because it was never intuitively obvious what this meant. With a newer version of *renice*, which used the term “system scheduling priority”, it was hoped that novice users could better understand what this utility was meant to do. Also, it would be easier to document what the utility was meant to do. Unfortunately, the addition of the POSIX realtime scheduling capabilities introduced the concepts of process and thread scheduling priorities that were totally unaffected by the *nice/renice* utilities or the *nice()/setpriority()* functions. Continuing to use the term “system scheduling priority” would have incorrectly suggested that these utilities and functions were indeed affecting these realtime priorities. It was decided to revert to the historical term “nice value” to reference this unrelated process attribute.

Although this utility has use by system administrators (and in fact appears in the system administration portion of the BSD documentation), the standard developers considered that it was very useful for individual end users to control their own processes.

Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
renice nice_value[-p] pid...[-g gid...][-p pid...][-u user...]
renice nice_value -g gid...[-g gid...]-p pid...[-u user...]
renice nice_value -u user...[-g gid...]-p pid...[-u user...]
```

These forms are no longer specified by POSIX.1-200x but may be present in some implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

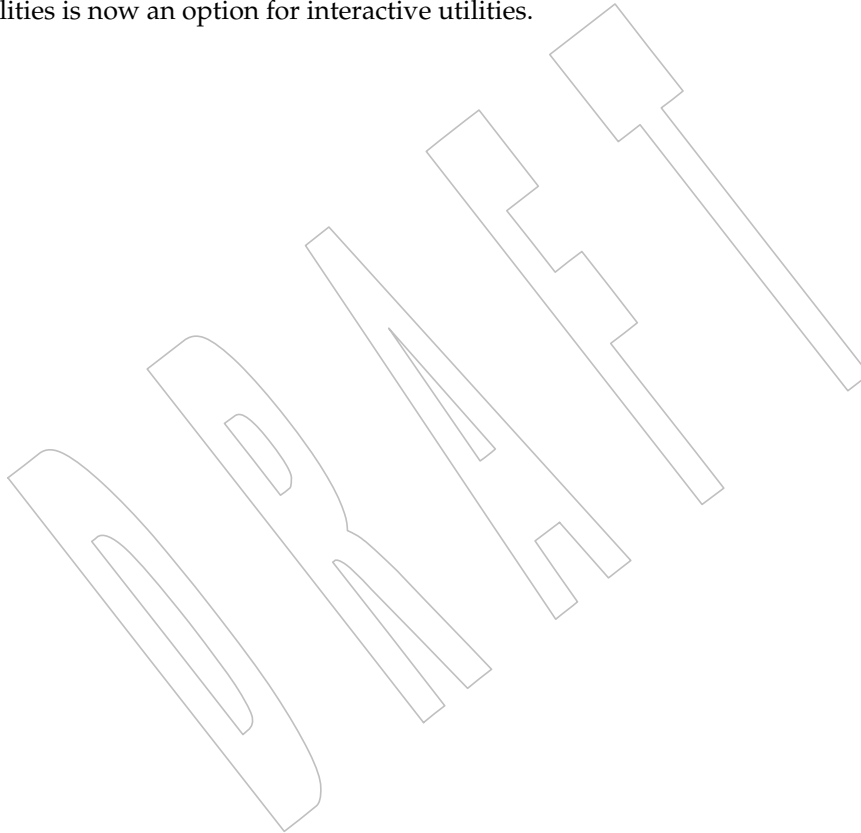
[nice](#)

XBD [Section 3.238](#) (on page 66), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

CHANGE HISTORY

First released in Issue 4.

102292	Issue 5	
102293		In the SYNOPSIS, an ellipsis is added to the -u option in all three obsolescent forms.
102294	Issue 6	
102295		This utility is marked as part of the User Portability Utilities option.
102296		The APPLICATION USAGE section is added.
102297		The obsolescent forms of the SYNOPSIS are removed.
102298		Text previously conditional on POSIX_SAVED_IDS is mandatory in this version. This is a FIPS
102299		requirement.
102300	Issue 7	
102301		Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility
102302		Syntax Guidelines does not apply.
102303		SD5-XCU-ERN-97 is applied, updating the SYNOPSIS. +
102304		The <i>renice</i> utility is moved from the User Portability Utilities option to the Base. User Portability
102305		Utilities is now an option for interactive utilities. -



102306 **NAME**
 102307 `rm` — remove directory entries

102308 **SYNOPSIS**
 102309 `rm [-fiRr] file...`

102310 **DESCRIPTION**

102311 The *rm* utility shall remove the directory entry specified by each *file* argument.

102312 If either of the files `dot` or `dot-dot` are specified as the basename portion of an operand (that is,
 102313 the final pathname component) or if an operand resolves to the root directory, *rm* shall write a
 102314 diagnostic message to standard error and do nothing more with such operands.

102315 For each *file* the following steps shall be taken:

- 102316 1. If the *file* does not exist:
- 102317 a. If the `-f` option is not specified, *rm* shall write a diagnostic message to standard
 102318 error.
 - 102319 b. Go on to any remaining *files*.
- 102320 2. If *file* is of type directory, the following steps shall be taken:
- 102321 a. If neither the `-R` option nor the `-r` option is specified, *rm* shall write a diagnostic
 102322 message to standard error, do nothing more with *file*, and go on to any remaining
 102323 files.
 - 102324 b. If the `-f` option is not specified, and either the permissions of *file* do not permit
 102325 writing and the standard input is a terminal or the `-i` option is specified, *rm* shall
 102326 write a prompt to standard error and read a line from the standard input. If the
 102327 response is not affirmative, *rm* shall do nothing more with the current file and go
 102328 on to any remaining files.
 - 102329 c. For each entry contained in *file*, other than `dot` or `dot-dot`, the four steps listed here
 102330 (1 to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall
 102331 not traverse directories by following symbolic links into other parts of the
 102332 hierarchy, but shall remove the links themselves.
 - 102333 d. If the `-i` option is specified, *rm* shall write a prompt to standard error and read a
 102334 line from the standard input. If the response is not affirmative, *rm* shall do nothing
 102335 more with the current file, and go on to any remaining files.
- 102336 3. If *file* is not of type directory, the `-f` option is not specified, and either the permissions of
 102337 *file* do not permit writing and the standard input is a terminal or the `-i` option is specified,
 102338 *rm* shall write a prompt to the standard error and read a line from the standard input. If
 102339 the response is not affirmative, *rm* shall do nothing more with the current file and go on
 102340 to any remaining files.
- 102341 4. If the current file is a directory, *rm* shall perform actions equivalent to the `rmdir()` function
 102342 defined in the System Interfaces volume of POSIX.1-200x called with a pathname of the
 102343 current file used as the *path* argument. If the current file is not a directory, *rm* shall
 102344 perform actions equivalent to the `unlink()` function defined in the System Interfaces
 102345 volume of POSIX.1-200x called with a pathname of the current file used as the *path*
 102346 argument.

102347 If this fails for any reason, *rm* shall write a diagnostic message to standard error, do
 102348 nothing more with the current file, and go on to any remaining files.

102349 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail

102350 due to path length limitations (unless an operand specified by the user exceeds system
102351 limitations).

OPTIONS

102352 The *rm* utility shall conform to XBD [Section 12.2](#) (on page 201). |

102353 The following options shall be supported:

- 102354
- 102355 **-f** Do not prompt for confirmation. Do not write diagnostic messages or modify the
102356 exit status in the case of nonexistent operands. Any previous occurrences of the **-i**
102357 option shall be ignored.
- 102358 **-i** Prompt for confirmation as described previously. Any previous occurrences of the
102359 **-f** option shall be ignored.
- 102360 **-R** Remove file hierarchies. See the DESCRIPTION.
- 102361 **-r** Equivalent to **-R**.

OPERANDS

102362 The following operand shall be supported:

102363 *file* A pathname of a directory entry to be removed.

STDIN

102365 The standard input shall be used to read an input line in response to each prompt specified in
102366 the STDOUT section. Otherwise, the standard input shall not be used.
102367

INPUT FILES

102368 None.
102369

ENVIRONMENT VARIABLES

102370 The following environment variables shall affect the execution of *rm*:

102371

102372 **LANG** Provide a default value for the internationalization variables that are unset or null. |
102373 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
102374 variables used to determine the values of locale categories.)

102375 **LC_ALL** If set to a non-empty string value, override the values of all the other
102376 internationalization variables.

102377 **LC_COLLATE**
102378 Determine the locale for the behavior of ranges, equivalence classes, and multi-
102379 character collating elements used in the extended regular expression defined for
102380 the **yesexpr** locale keyword in the **LC_MESSAGES** category.

102381 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
102382 characters (for example, single-byte as opposed to multi-byte characters in
102383 arguments) and the behavior of character classes within regular expressions used
102384 in the extended regular expression defined for the **yesexpr** locale keyword in the
102385 **LC_MESSAGES** category.

102386 **LC_MESSAGES**
102387 Determine the locale for the processing of affirmative responses that should be
102388 used to affect the format and contents of diagnostic messages written to standard
102389 error.

102390 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

102391 **ASYNCHRONOUS EVENTS**

102392 Default.

102393 **STDOUT**

102394 Not used.

102395 **STDERR**

102396 Prompts shall be written to standard error under the conditions specified in the DESCRIPTION
 102397 and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is
 102398 otherwise unspecified. The standard error also shall be used for diagnostic messages.

102399 **OUTPUT FILES**

102400 None.

102401 **EXTENDED DESCRIPTION**

102402 None.

102403 **EXIT STATUS**

102404 The following exit values shall be returned:

102405 0 Each directory entry was successfully removed, unless its removal was canceled by a non-
 102406 affirmative response to a prompt for confirmation.

102407 >0 An error occurred.

102408 **CONSEQUENCES OF ERRORS**

102409 Default.

102410 **APPLICATION USAGE**

102411 The *rm* utility is forbidden to remove the names *dot* and *dot-dot* in order to avoid the
 102412 consequences of inadvertently doing something like:

102413 `rm -r .*`

102414 Some implementations do not permit the removal of the last link to an executable binary file that
 102415 is being executed; see the [EBUSY] error in the *unlink()* function defined in the System Interfaces
 102416 volume of POSIX.1-200x. Thus, the *rm* utility can fail to remove such files.

102417 The `-i` option causes *rm* to prompt and read the standard input even if the standard input is not
 102418 a terminal, but in the absence of `-i` the mode prompting is not done when the standard input is
 102419 not a terminal.

102420 **EXAMPLES**

102421 1. The following command:

102422 `rm a.out core`102423 removes the directory entries: **a.out** and **core**.

102424 2. The following command:

102425 `rm -Rf junk`102426 removes the directory **junk** and all its contents, without prompting.102427 **RATIONALE**

102428 For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior
 102429 when prompting for confirmation, should be interpreted in the following manner:

```
102430 if ((NOT f_option) AND
102431     ((not_writable AND input_is_terminal) OR i_option))
```

102432 The exact format of the interactive prompts is unspecified. Only the general nature of the
 102433 contents of prompts are specified because implementations may desire more descriptive

102434 prompts than those used on historical implementations. Therefore, an application not using the
 102435 `-f` option, or using the `-i` option, relies on the system to provide the most suitable dialog directly
 102436 with the user, based on the behavior specified.

102437 The `-r` option is historical practice on all known systems. The synonym `-R` option is provided
 102438 for consistency with the other utilities in this volume of POSIX.1-200x that provide options
 102439 requesting recursive descent through the file hierarchy.

102440 The behavior of the `-f` option in historical versions of *rm* is inconsistent. In general, along with
 102441 “forcing” the unlink without prompting for permission, it always causes diagnostic messages to
 102442 be suppressed and the exit status to be unmodified for nonexistent operands and files that
 102443 cannot be unlinked. In some versions, however, the `-f` option suppresses usage messages and
 102444 system errors as well. Suppressing such messages is not a service to either shell scripts or users.

102445 It is less clear that error messages regarding files that cannot be unlinked (removed) should be
 102446 suppressed. Although this is historical practice, this volume of POSIX.1-200x does not permit the
 102447 `-f` option to suppress such messages.

102448 When given the `-r` and `-i` options, historical versions of *rm* prompt the user twice for each
 102449 directory, once before removing its contents and once before actually attempting to delete the
 102450 directory entry that names it. This allows the user to “prune” the file hierarchy walk. Historical
 102451 versions of *rm* were inconsistent in that some did not do the former prompt for directories
 102452 named on the command line and others had obscure prompting behavior when the `-i` option
 102453 was specified and the permissions of the file did not permit writing. The POSIX Shell and
 102454 Utilities *rm* differs little from historic practice, but does require that prompts be consistent.
 102455 Historical versions of *rm* were also inconsistent in that prompts were done to both standard
 102456 output and standard error. This volume of POSIX.1-200x requires that prompts be done to
 102457 standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that
 102458 provide an option to list deleted files on standard output.

102459 The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be
 102460 deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its
 102461 descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the
 102462 historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted
 102463 to fail because of path length restrictions, unless an operand specified by the user is longer than
 102464 `{PATH_MAX}`.

102465 The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of
 102466 the dependence on the *unlink()* functionality, per the DESCRIPTION. When removing
 102467 hierarchies with `-r` or `-R`, the prohibition on following symbolic links has to be made explicit.

102468 **FUTURE DIRECTIONS**

102469 None.

102470 **SEE ALSO**

102471 *rmdir*

102472 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

102473 XSH *remove()*, *rmdir*, *unlink*

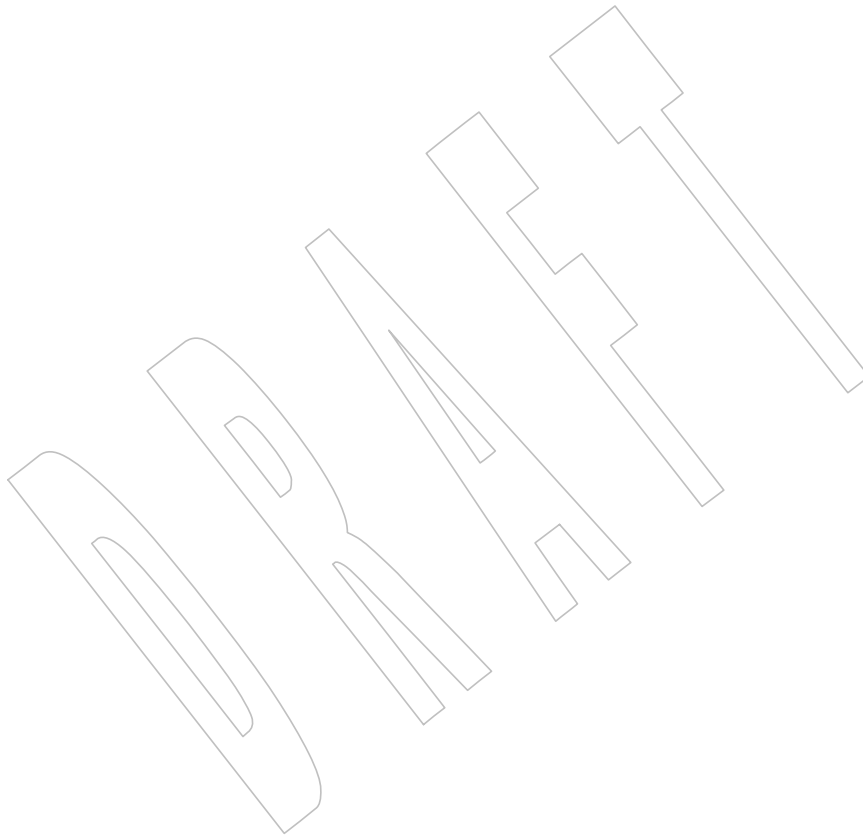
102474 **CHANGE HISTORY**

102475 First released in Issue 2.

102476 **Issue 5**

102477 The FUTURE DIRECTIONS section is added.

- 102478 **Issue 6**
- 102479 Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft
- 102480 standard.
- 102481 **Issue 7**
- 102482 Austin Group Interpretations 1003.1-2001 #019 and #091 are applied.



102483 **NAME**102484 rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)102485 **SYNOPSIS**102486 XSI `rmdel -r SID file...`102487 **DESCRIPTION**

102488 The *rmdel* utility shall remove the delta specified by the SID from each named SCCS file. The
 102489 delta to be removed shall be the most recent delta in its branch in the delta chain of each named
 102490 SCCS file. In addition, the application shall ensure that the SID specified is not that of a version
 102491 being edited for the purpose of making a delta; that is, if a *p-file* (see *get*) exists for the named
 102492 SCCS file, the SID specified shall not appear in any entry of the *p-file*.

102493 Removal of a delta shall be restricted to:

- 102494 1. The user who made the delta
- 102495 2. The owner of the SCCS file
- 102496 3. The owner of the directory containing the SCCS file

102497 **OPTIONS**102498 The *rmdel* utility shall conform to XBD [Section 12.2](#) (on page 201).

102499 The following option shall be supported:

102500 **-r** *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.102501 **OPERANDS**

102502 The following operand shall be supported:

102503 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel*
 102504 utility shall behave as though each file in the directory were specified as a named
 102505 file, except that non-SCCS files (last component of the pathname does not begin
 102506 with **s**.) and unreadable files shall be silently ignored.

102507 If exactly one *file* operand appears, and it is `'-'`, the standard input shall be read;
 102508 each line of the standard input is taken to be the name of an SCCS file to be
 102509 processed. Non-SCCS files and unreadable files shall be silently ignored.

102510 **STDIN**

102511 The standard input shall be a text file used only when the *file* operand is specified as `'-'`. Each
 102512 line of the text file shall be interpreted as an SCCS pathname.

102513 **INPUT FILES**

102514 The SCCS files shall be files of unspecified format.

102515 **ENVIRONMENT VARIABLES**102516 The following environment variables shall affect the execution of *rmdel*:

102517 **LANG** Provide a default value for the internationalization variables that are unset or null.
 102518 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 102519 variables used to determine the values of locale categories.)

102520 **LC_ALL** If set to a non-empty string value, override the values of all the other
 102521 internationalization variables.

102522 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 102523 characters (for example, single-byte as opposed to multi-byte characters in
 102524 arguments and input files).

102525	LC_MESSAGES	
102526		Determine the locale that should be used to affect the format and contents of
102527		diagnostic messages written to standard error.
102528	NLSPATH	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
102529	ASYNCHRONOUS EVENTS	
102530		Default.
102531	STDOUT	
102532		Not used.
102533	STDERR	
102534		The standard error shall be used only for diagnostic messages.
102535	OUTPUT FILES	
102536		The SCCS files shall be files of unspecified format. During processing of a <i>file</i> , a temporary <i>x-file</i> ,
102537		as described in <i>admin</i> , may be created and deleted; a locking <i>z-file</i> , as described in <i>get</i> , may be
102538		created and deleted.
102539	EXTENDED DESCRIPTION	
102540		None.
102541	EXIT STATUS	
102542		The following exit values shall be returned:
102543	0	Successful completion.
102544	>0	An error occurred.
102545	CONSEQUENCES OF ERRORS	
102546		Default.
102547	APPLICATION USAGE	
102548		None.
102549	EXAMPLES	
102550		None.
102551	RATIONALE	
102552		None.
102553	FUTURE DIRECTIONS	
102554		None.
102555	SEE ALSO	
102556		<i>admin</i> , <i>delta</i> , <i>get</i> , <i>prs</i>
102557		XBD Chapter 8 (on page 159), Section 12.2 (on page 201)
102558	CHANGE HISTORY	
102559		First released in Issue 2.
102560	Issue 6	
102561		The normative text is reworded to avoid use of the term “must” for application requirements.

102562 **NAME**102563 `rmdir` — remove directories102564 **SYNOPSIS**102565 `rmdir [-p] dir...`102566 **DESCRIPTION**102567 The *rmdir* utility shall remove the directory entry specified by each *dir* operand.102568 For each *dir* operand, the *rmdir* utility shall perform actions equivalent to the *rmdir()* function
102569 called with the *dir* operand as its only argument.102570 Directories shall be processed in the order specified. If a directory and a subdirectory of that
102571 directory are specified in a single invocation of the *rmdir* utility, the application shall specify the
102572 subdirectory before the parent directory so that the parent directory will be empty when the
102573 *rmdir* utility tries to remove it.102574 **OPTIONS**102575 The *rmdir* utility shall conform to XBD [Section 12.2](#) (on page 201).

102576 The following option shall be supported:

102577 **-p** Remove all directories in a pathname. For each *dir* operand:

- 102578
1. The directory entry it names shall be removed.
 2. If the *dir* operand includes more than one pathname component, effects
102579 equivalent to the following command shall occur:
102580

102581 `rmdir -p $(dirname dir)`102582 **OPERANDS**

102583 The following operand shall be supported:

102584 *dir* A pathname of an empty directory to be removed.102585 **STDIN**

102586 Not used.

102587 **INPUT FILES**

102588 None.

102589 **ENVIRONMENT VARIABLES**102590 The following environment variables shall affect the execution of *rmdir*:102591 **LANG** Provide a default value for the internationalization variables that are unset or null.
102592 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
102593 variables used to determine the values of locale categories.)102594 **LC_ALL** If set to a non-empty string value, override the values of all the other
102595 internationalization variables.102596 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
102597 characters (for example, single-byte as opposed to multi-byte characters in
102598 arguments).102599 **LC_MESSAGES**102600 Determine the locale that should be used to affect the format and contents of
102601 diagnostic messages written to standard error.

102602 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

102603 ASYNCHRONOUS EVENTS

102604 Default.

102605 STDOUT

102606 Not used.

102607 STDERR

102608 The standard error shall be used only for diagnostic messages.

102609 OUTPUT FILES

102610 None.

102611 EXTENDED DESCRIPTION

102612 None.

102613 EXIT STATUS

102614 The following exit values shall be returned:

102615 0 Each directory entry specified by a *dir* operand was removed successfully.

102616 >0 An error occurred.

102617 CONSEQUENCES OF ERRORS

102618 Default.

102619 APPLICATION USAGE

102620 The definition of an empty directory is one that contains, at most, directory entries for dot and
102621 dot-dot.

102622 EXAMPLES

102623 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
102624 except it contains a directory **c**:

102625 `rmdir -p a/b/c`

102626 removes all three directories.

102627 RATIONALE

102628 On historical System V systems, the `-p` option also caused a message to be written to the
102629 standard output. The message indicated whether the whole path was removed or whether part
102630 of the path remained for some reason. The STDERR section requires this diagnostic when the
102631 entire path specified by a *dir* operand is not removed, but does not allow the status message
102632 reporting success to be written as a diagnostic.

102633 The *rmdir* utility on System V also included a `-s` option that suppressed the informational
102634 message output by the `-p` option. This option has been omitted because the informational
102635 message is not specified by this volume of POSIX.1-200x.

102636 FUTURE DIRECTIONS

102637 None.

102638 SEE ALSO

102639 *rm*

102640 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

102641 XSH *remove()*, *rmdir*, *unlink*

102642

CHANGE HISTORY

102643

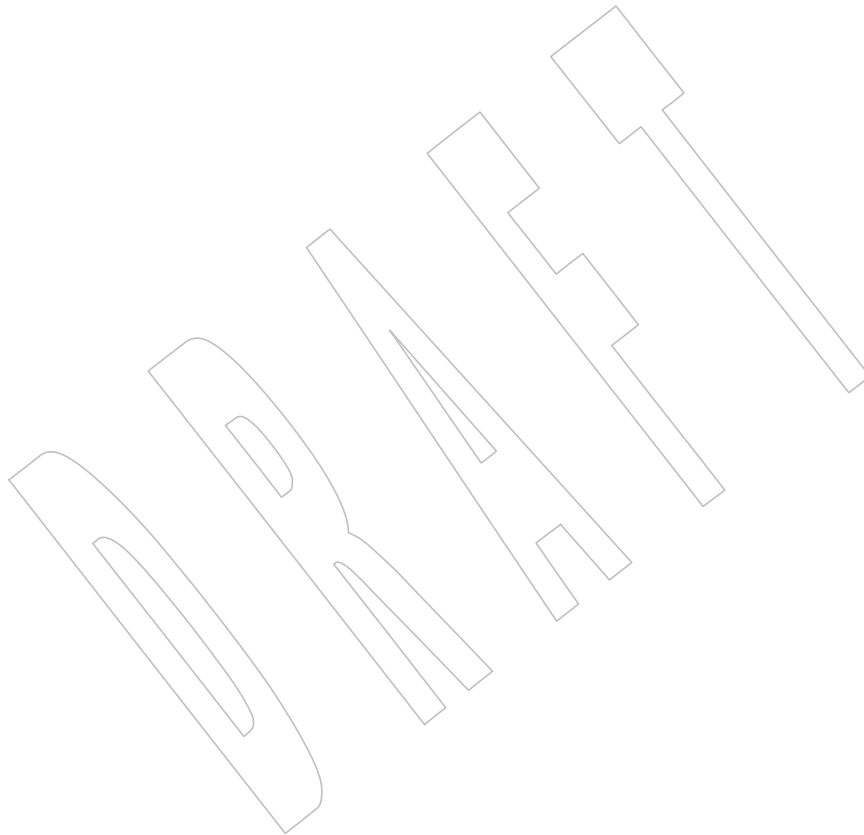
First released in Issue 2.

102644

Issue 6

102645

The normative text is reworded to avoid use of the term “must” for application requirements.



102646 **NAME**
 102647 sact — print current SCCS file-editing activity (**DEVELOPMENT**)

102648 **SYNOPSIS**
 102649 XSI `sact file...`

102650 **DESCRIPTION**
 102651 The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a
 102652 list to standard output. This situation occurs when *get -e* has been executed previously without
 102653 a subsequent execution of *delta*, *unget*, or *sccs unedit*.

102654 **OPTIONS**
 102655 None.

102656 **OPERANDS**
 102657 The following operand shall be supported:

102658 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact*
 102659 utility shall behave as though each file in the directory were specified as a named
 102660 file, except that non-SCCS files (last component of the pathname does not begin
 102661 with **s**.) and unreadable files shall be silently ignored.

102662 If exactly one *file* operand appears, and it is *'-'*, the standard input shall be read;
 102663 each line of the standard input shall be taken to be the name of an SCCS file to be
 102664 processed. Non-SCCS files and unreadable files shall be silently ignored.

102665 **STDIN**
 102666 The standard input shall be a text file used only when the *file* operand is specified as *'-'*. Each
 102667 line of the text file shall be interpreted as an SCCS pathname.

102668 **INPUT FILES**
 102669 Any SCCS files interrogated are files of an unspecified format.

102670 **ENVIRONMENT VARIABLES**
 102671 The following environment variables shall affect the execution of *sact*:

102672 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 102673 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 102674 variables used to determine the values of locale categories.)

102675 *LC_ALL* If set to a non-empty string value, override the values of all the other
 102676 internationalization variables.

102677 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 102678 characters (for example, single-byte as opposed to multi-byte characters in
 102679 arguments and input files).

102680 *LC_MESSAGES*
 102681 Determine the locale that should be used to affect the format and contents of
 102682 diagnostic messages written to standard error.

102683 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

102684 **ASYNCHRONOUS EVENTS**
 102685 Default.

STDOUT

The output for each named file shall consist of a line in the following format:

```
"%sΔ%sΔ%sΔ%sΔ%s\n", <SID>, <new SID>, <login>, <date>, <time>
```

<SID> Specifies the SID of a delta that currently exists in the SCCS file to which changes are made to make the new delta.

<new SID> Specifies the SID for the new delta to be created.

<login> Contains the login name of the user who makes the delta (that is, who executed a *get* for editing).

<date> Contains the date that *get -e* was executed, in the format used by the *prs :D:* data keyword.

<time> Contains the time that *get -e* was executed, in the format used by the *prs :T:* data keyword.

If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the preceding lines:

```
"\n%s:\n", <pathname>
```

STDERR

The standard error shall be used only for optional informative messages concerning SCCS files with no impending deltas, and for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

delta, get, sccs, unget

XBD Chapter 8 (on page 159)

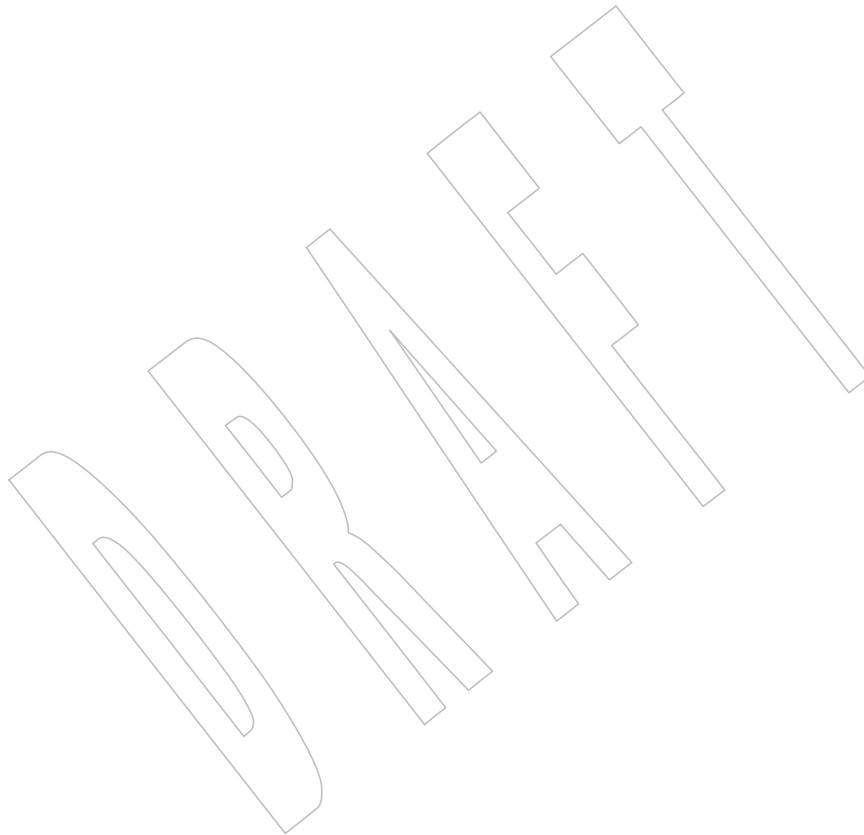
+

102725

102726

CHANGE HISTORY

First released in Issue 2.



102727 **NAME**
 102728 `sccs` — front end for the SCCS subsystem (**DEVELOPMENT**)

102729 **SYNOPSIS**
 102730 XSI `sccs [-r] [-d path] [-p path] command [options...] [operands...]`

102731 **DESCRIPTION**
 102732 The `sccs` utility is a front end to the SCCS programs. It also includes the capability to run set-
 102733 user-id to another user to provide additional protection.

102734 The `sccs` utility shall invoke the specified *command* with the specified *options* and *operands*. By
 102735 default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s. ".

102736 The *command* can be the name of one of the SCCS utilities in this volume of POSIX.1-200x (*admin*,
 102737 *delta*, *get*, *prs*, *rmDEL*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the
 102738 EXTENDED DESCRIPTION section.

102739 **OPTIONS**
 102740 The `sccs` utility shall conform to XBD [Section 12.2](#) (on page 201), except that *options* operands are
 102741 actually options to be passed to the utility named by *command*. When the portion of the
 102742 command:

102743 `command [options ...] [operands ...]`

102744 is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax
 102745 Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the
 102746 Guidelines to the extent indicated by their individual OPTIONS sections.

102747 The following options shall be supported preceding the *command* operand:

102748 **-d path** A pathname of a directory to be used as a root directory for the SCCS files. The
 102749 default shall be the current directory. The **-d** option shall take precedence over the
 102750 *PROJECTDIR* variable. See **-p**.

102751 **-p path** A pathname of a directory in which the SCCS files are located. The default shall be
 102752 the **SCCS** directory.

102753 The **-p** option differs from the **-d** option in that the **-d** option-argument shall be
 102754 prefixed to the entire pathname and the **-p** option-argument shall be inserted
 102755 before the final component of the pathname. For example:

102756 `sccs -d /x -p y get a/b`

102757 converts to:

102758 `get /x/a/y/s.b`

102759 This allows the creation of aliases such as:

102760 `alias syssccs="sccs -d /usr/src"`

102761 which is used as:

102762 `syssccs get cmd/who.c`

102763 **-r** Invoke *command* with the real user ID of the process, not any effective user ID that
 102764 the `sccs` utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmDEL*,
 102765 and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to
 102766 change the authorizations. These commands are always run as the real user.

102767 **OPERANDS**

102768 The following operands shall be supported:

102769 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the
102770 EXTENDED DESCRIPTION section.102771 *options* An option or option-argument to be passed to *command*.102772 *operands* An operand to be passed to *command*.102773 **STDIN**102774 See the utility description for the specified *command*.102775 **INPUT FILES**102776 See the utility description for the specified *command*.102777 **ENVIRONMENT VARIABLES**102778 The following environment variables shall affect the execution of *sccs*:102779 *LANG* Provide a default value for the internationalization variables that are unset or null. |
102780 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
102781 variables used to determine the values of locale categories.)102782 *LC_ALL* If set to a non-empty string value, override the values of all the other
102783 internationalization variables.102784 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
102785 characters (for example, single-byte as opposed to multi-byte characters in
102786 arguments and input files).102787 *LC_MESSAGES*102788 Determine the locale that should be used to affect the format and contents of
102789 diagnostic messages written to standard error.102790 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.102791 *PROJECTDIR*102792 Provide a default value for the *-d path* option. If the value of *PROJECTDIR* begins
102793 with a slash, it shall be considered an absolute pathname; otherwise, the value of
102794 *PROJECTDIR* is treated as a user name and that user's initial working directory
102795 shall be examined for a subdirectory *src* or *source*. If such a directory is found, it
102796 shall be used. Otherwise, the value shall be used as a relative pathname.102797 Additional environment variable effects may be found in the utility description for the specified
102798 *command*.102799 **ASYNCHRONOUS EVENTS**

102800 Default.

102801 **STDOUT**102802 See the utility description for the specified *command*.102803 **STDERR**102804 See the utility description for the specified *command*.102805 **OUTPUT FILES**102806 See the utility description for the specified *command*.102807 **EXTENDED DESCRIPTION**102808 The following pseudo-utilities shall be supported as *command* operands. All options referred to
102809 in the following list are values given in the *options* operands following *command*.

- 102810 **check** Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a
 102811 non-zero exit status shall be returned if anything is being edited. The intent is to have
 102812 this included in an “install” entry in a makefile to ensure that everything is included
 102813 into the SCCS file before a version is installed.
- 102814 **clean** Remove everything from the current directory that can be recreated from SCCS files,
 102815 but do not remove any files being edited. If the **-b** option is given, branches shall be
 102816 ignored in the determination of whether they are being edited; this is dangerous if
 102817 branches are kept in the same directory.
- 102818 **create** Create an SCCS file, taking the initial contents from the file of the same name. Any
 102819 options to *admin* are accepted. If the creation is successful, the original files shall be
 102820 renamed by prefixing the basenames with a comma. These renamed files should be
 102821 removed after it has been verified that the SCCS files have been created successfully.
- 102822 **delget** Perform a *delta* on the named files and then *get* new versions. The new versions shall
 102823 have ID keywords expanded and shall not be editable. Any **-m**, **-p**, **-r**, **-s**, and **-y**
 102824 options shall be passed to *delta*, and any **-b**, **-c**, **-e**, **-i**, **-k**, **-l**, **-s**, and **-x** options shall be
 102825 passed to *get*.
- 102826 **deledit** Equivalent to **delget**, except that the *get* phase shall include the **-e** option. This option is
 102827 useful for making a checkpoint of the current editing phase. The same options shall be
 102828 passed to *delta* as described above, and all the options listed for *get* above except **-e**
 102829 shall be passed to **edit**.
- 102830 **diffs** Write a difference listing between the current version of the files checked out for editing
 102831 and the versions in SCCS format. Any **-r**, **-c**, **-i**, **-x**, and **-t** options shall be passed to
 102832 *get*; any **-l**, **-s**, **-e**, **-f**, **-h**, and **-b** options shall be passed to *diff*. A **-C** option shall be
 102833 passed to *diff* as **-c**.
- 102834 **edit** Equivalent to *get -e*.
- 102835 **fix** Remove the named delta, but leave a copy of the delta with the changes that were in it.
 102836 It is useful for fixing small compiler bugs, and so on. The application shall ensure that it
 102837 is followed by a **-r SID** option. Since **fix** does not leave audit trails, it should be used
 102838 carefully.
- 102839 **info** Write a listing of all files being edited. If the **-b** option is given, branches (that is, SIDs
 102840 with two or fewer components) shall be ignored. If a **-u user** option is given, then only
 102841 files being edited by the named user shall be listed. A **-U** option shall be equivalent to
 102842 **-u<current user>**.
- 102843 **print** Write out verbose information about the named files, equivalent to *sccs prs*.
- 102844 **tell** Write a <newline>-separated list of the files being edited to standard output. Takes the
 102845 **-b**, **-u**, and **-U** options like **info** and **check**.
- 102846 **unedit** This is the opposite of an **edit** or a *get -e*. It should be used with caution, since any
 102847 changes made since the *get* are lost.

EXIT STATUS

102848 The following exit values shall be returned:

- 102849 0 Successful completion.
- 102850 >0 An error occurred.

CONSEQUENCES OF ERRORS

102852 Default.

102853

APPLICATION USAGE

Many of the SCCS utilities take directory names as operands as well as specific filenames. The pseudo-utilities supported by *sccs* are not described as having this capability, but are not prohibited from doing so.

EXAMPLES

1. To get a file for editing, edit it and produce a new delta:

```
sccs get -e file.c
ex file.c
sccs delta file.c
```

2. To get a file from another directory:

```
sccs -p /usr/src/sccs/s. get cc.c
or:
```

```
sccs get /usr/src/sccs/s.cc.c
```

3. To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

4. To get a list of files being edited that are not on branches:

```
sccs info -b
```

5. To delta everything being edited by the current user:

```
sccs delta $(sccs tell -U)
```

6. In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
    sccs get $(REL) $@
```

RATIONALE

sccs and its associated utilities are part of the XSI Development Utilities option within the XSI option.

SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement tracking tool. When a file is put under SCCS, the source code control system maintains the file and, when changes are made, identifies and stores them in the file with the original source code and/or documentation. As other changes are made, they too are identified and retained in the file.

Retrieval of the original and any set of changes is possible. Any version of the file as it develops can be reconstructed for inspection or additional modification. History data can be stored with each version, documenting why the changes were made, who made them, and when they were made.

FUTURE DIRECTIONS

None.

SEE ALSO

admin, delta, get, make, prs, rmdel, sact, unget, val, what

XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

+

102894

CHANGE HISTORY

102895

First released in Issue 4.

102896

Issue 6

102897

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from “otherwise, the home directory of a user of that name is examined” to “otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined”.

102898

102899

The normative text is reworded to avoid use of the term “must” for application requirements.

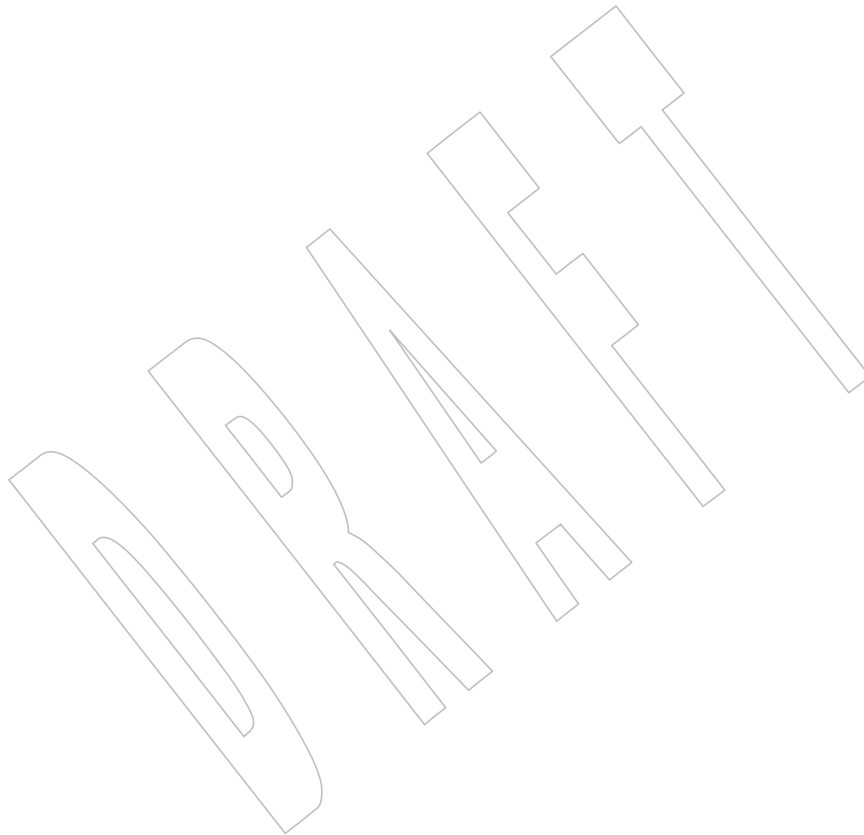
102900

102901

Issue 7

102902

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



102903 **NAME**102904 `sed` — stream editor102905 **SYNOPSIS**102906 `sed [-n] script [file...]`102907 `sed [-n] -e script [-e script]... [-f script_file]... [file...]` |102908 `sed [-n] [-e script]... -f script_file [-f script_file]... [file...]` |102909 **DESCRIPTION**

102910 The `sed` utility is a stream editor that shall read one or more text files, make editing changes
 102911 according to a script of editing commands, and write the results to standard output. The script
 102912 shall be obtained from either the `script` operand string or a combination of the option-arguments
 102913 from the `-e script` and `-f script_file` options.

102914 **OPTIONS**

102915 The `sed` utility shall conform to XBD [Section 12.2](#) (on page 201), except that the order of
 102916 presentation of the `-e` and `-f` options is significant. |

102917 The following options shall be supported:

102918 `-e script` Add the editing commands specified by the `script` option-argument to the end of
 102919 the script of editing commands. The `script` option-argument shall have the same
 102920 properties as the `script` operand, described in the OPERANDS section.

102921 `-f script_file` Add the editing commands in the file `script_file` to the end of the script.

102922 `-n` Suppress the default output (in which each line, after it is examined for editing, is
 102923 written to standard output). Only lines explicitly selected for output are written.

102924 Multiple `-e` and `-f` options may be specified. All commands shall be added to the script in the
 102925 order specified, regardless of their origin.

102926 **OPERANDS**

102927 The following operands shall be supported:

102928 `file` A pathname of a file whose contents are read and edited. If multiple `file` operands
 102929 are specified, the named files shall be read in the order specified and the
 102930 concatenation shall be edited. If no `file` operands are specified, the standard input
 102931 shall be used.

102932 `script` A string to be used as the script of editing commands. The application shall not
 102933 present a `script` that violates the restrictions of a text file except that the final
 102934 character need not be a <newline>.

102935 **STDIN**

102936 The standard input shall be used only if no `file` operands are specified. See the INPUT FILES
 102937 section.

102938 **INPUT FILES**

102939 The input files shall be text files. The `script_files` named by the `-f` option shall consist of editing
 102940 commands.

102941 **ENVIRONMENT VARIABLES**102942 The following environment variables shall affect the execution of `sed`:

102943 `LANG` Provide a default value for the internationalization variables that are unset or null. |
 102944 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
 102945 variables used to determine the values of locale categories.)

- 102946 *LC_ALL* If set to a non-empty string value, override the values of all the other
102947 internationalization variables.
- 102948 *LC_COLLATE*
102949 Determine the locale for the behavior of ranges, equivalence classes, and multi-
102950 character collating elements within regular expressions.
- 102951 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
102952 characters (for example, single-byte as opposed to multi-byte characters in
102953 arguments and input files), and the behavior of character classes within regular
102954 expressions.
- 102955 *LC_MESSAGES*
102956 Determine the locale that should be used to affect the format and contents of
102957 diagnostic messages written to standard error.
- 102958 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

102959 Default.
102960

STDOUT

102961 The input files shall be written to standard output, with the editing commands specified in the
102962 script applied. If the *-n* option is specified, only those input lines selected by the script shall be
102963 written to standard output.
102964

STDERR

102965 The standard error shall be used only for diagnostic messages.
102966

OUTPUT FILES

102967 The output files shall be text files whose formats are dependent on the editing commands given.
102968

EXTENDED DESCRIPTION

102969 The *script* shall consist of editing commands of the following form:
102970

102971 *[address[, address]]function*

102972 where *function* represents a single-character command verb from the list in [Editing Commands](#)
102973 [in sed](#) (on page 3067), followed by any applicable arguments.

102974 The command can be preceded by <blank>s and/or semicolons. The function can be preceded
102975 by <blank>s. These optional characters shall have no effect.

102976 In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>,
102977 into the pattern space. Normally the pattern space will be empty, unless a **D** command
102978 terminated the last cycle. The *sed* utility shall then apply in sequence all commands whose
102979 addresses select that pattern space, and at the end of the script copy the pattern space to
102980 standard output (except when *-n* is specified) and delete the pattern space. Whenever the
102981 pattern space is written to standard output or a named file, *sed* shall immediately follow it with a
102982 <newline>.

102983 Some of the editing commands use a hold space to save all or part of the pattern space for
102984 subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

102985 **Addresses in sed**

102986 An address is either a decimal number that counts input lines cumulatively across files, a '\$' character that addresses the last line of input, or a context address (which consists of a BRE, as described in [Regular Expressions in sed](#), preceded and followed by a delimiter, usually a slash).

102989 An editing command with no addresses shall select every pattern space.

102990 An editing command with one address shall select each pattern space that matches the address.

102991 An editing command with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter, the process shall be repeated. Omitting either or both of the address components in the following form produces undefined results:

102997 `[address[, address]]`

102998 **Regular Expressions in sed**

102999 The *sed* utility shall support the BREs described in XBD [Section 9.3](#) (on page 169), with the following additions:

- 103001 • In a context address, the construction "`\cBREc`", where *c* is any character other than
103002 backslash or <newline>, shall be identical to "`/BRE/`". If the character designated by *c*
103003 appears following a backslash, then it shall be considered to be that literal character, which
103004 shall not terminate the BRE. For example, in the context address "`\xabc\xdefx`", the
103005 second *x* stands for itself, so that the BRE is "`abcxdef`".
- 103006 • The escape sequence '`\n`' shall match a <newline> embedded in the pattern space. A
103007 literal <newline> shall not be used in the BRE of a context address or in the substitute
103008 function.
- 103009 • If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in
103010 the last command applied (either as an address or as part of a substitute command) was
103011 specified.

103012 **Editing Commands in sed**

103013 In the following list of editing commands, the maximum number of permissible addresses for
103014 each function is indicated by `[0addr]`, `[1addr]`, or `[2addr]`, representing zero, one, or two
103015 addresses.

103016 The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall
103017 be preceded by a backslash. Other backslashes in text shall be removed, and the following
103018 character shall be treated literally.

103019 The **r** and **w** command verbs, and the *w* flag to the **s** command, take an *rfile* (or *wfile*) parameter,
103020 separated from the command verb letter or flag by one or more <blank>s; implementations may
103021 allow zero separation as an extension.

103022 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be
103023 created before processing begins. Implementations shall support at least ten *wfile* arguments in
103024 the script; the actual number (greater than or equal to 10) that is supported by the
103025 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially
103026 created, if it does not exist, or shall replace the contents of an existing file.

103027 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following
103028 synopses indicate which arguments shall be separated from the command verbs by a single
103029 <space>.

103030 The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and
 103031 the contents of the file specified for the **r** command, shall be written to standard output just
 103032 before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when
 103033 reaching the end of the script. If written when reaching the end of the script, and the **-n** option
 103034 was not specified, the text shall be written after copying the pattern space to standard output.
 103035 The contents of the file specified for the **r** command shall be as of the time the output is written,
 103036 not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r**
 103037 commands were applied to the input.

103038 Command verbs other than **{**, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a semicolon, optional
 103039 **<blank>s**, and another command verb. However, when the **s** command verb is used with the **w**
 103040 flag, following it with another command in this manner produces undefined results.

103041 A function can be preceded by one or more **' ! '** characters, in which case the function shall be
 103042 applied if the addresses do not select the pattern space. Zero or more **<blank>s** shall be accepted
 103043 before the first **' ! '** character. It is unspecified whether **<blank>s** can follow a **' ! '** character, and
 103044 conforming applications shall not follow a **' ! '** character with **<blank>s**.

103045 **[2addr]{function**
 103046 **function**
 103047 **...**
 103048 **}**

Execute a list of *sed* functions only when the pattern space is selected. The list of *sed* functions shall be surrounded by braces and separated by **<newline>s**, and conform to the following rules. The braces can be preceded or followed by **<blank>s**. The functions can be preceded by **<blank>s**, but shall not be followed by **<blank>s**. The **<right-brace>** shall be preceded by a **<newline>** and can be preceded or followed by **<blank>s**.

103054 **[1addr]a**
 103055 **text**

Write text to standard output as described previously.

103056 **[2addr]b [label]**

Branch to the **:** function bearing the *label*. If *label* is not specified, branch to the end of the script. The implementation shall support *labels* recognized as unique up to at least 8 characters; the actual length (greater than or equal to 8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

103062 **[2addr]c**
 103063 **text**

Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range, place *text* on the output and start the next cycle.

103065 **[2addr]d** Delete the pattern space and start the next cycle.

103066 **[2addr]D** Delete the initial segment of the pattern space through the first **<newline>** and
 103067 start the next cycle.

103068 **[2addr]g** Replace the contents of the pattern space by the contents of the hold space.

103069 **[2addr]G** Append to the pattern space a **<newline>** followed by the contents of the hold
 103070 space.

103071 **[2addr]h** Replace the contents of the hold space with the contents of the pattern space.

103072 **[2addr]H** Append to the hold space a **<newline>** followed by the contents of the pattern
 103073 space.

103074 **[1addr]i**
 103075 **text**

Write *text* to standard output.

103076 103077 103078 103079 103080 103081	[2addr]l	(The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in XBD Table 5-1 (on page 108) ('\\', '\a', '\b', '\f', '\r', '\t', '\v') shall be written as the corresponding escape sequence; the '\n' in that table is not applicable. Non-printable characters not in that table shall be written as one three-digit octal number (with a preceding backslash) for each byte in the character (most significant byte first).
103082 103083 103084 103085		Long lines shall be folded, with the point of folding indicated by writing a backslash followed by a <newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a '\$'.
103086 103087 103088	[2addr]n	Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input, less its terminating <newline>.
103089 103090		If no next line of input is available, the n command verb shall branch to the end of the script and quit without starting a new cycle.
103091 103092 103093	[2addr]N	Append the next line of input, less its terminating <newline>, to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.
103094 103095 103096		If no next line of input is available, the N command verb shall branch to the end of the script and quit without starting a new cycle or copying the pattern space to standard output.
103097	[2addr]p	Write the pattern space to standard output.
103098	[2addr]P	Write the pattern space, up to the first <newline>, to standard output.
103099	[1addr]q	Branch to the end of the script and quit without starting a new cycle.
103100 103101 103102	[1addr]r rfile	Copy the contents of <i>rfile</i> to standard output as described previously. If <i>rfile</i> does not exist or cannot be read, it shall be treated as if it were an empty file, causing no error condition.
103103 103104 103105 103106 103107 103108	[2addr]s/BRE/replacement/flags	Substitute the replacement string for instances of the BRE in the pattern space. Any character other than backslash or <newline> can be used instead of a slash to delimit the BRE and the replacement. Within the BRE and the replacement, the BRE delimiter itself can be used as a literal character if it is preceded by a backslash.
103109 103110 103111 103112 103113 103114 103115 103116 103117 103118 103119		The replacement string shall be scanned from beginning to end. An ampersand ('&') appearing in the replacement shall be replaced by the string matching the BRE. The special meaning of '&' in this context can be suppressed by preceding it by a backslash. The characters "\n", where <i>n</i> is a digit, shall be replaced by the text matched by the corresponding back-reference expression. If the corresponding back-reference expression does not match, then the characters "\n" shall be replaced by the empty string. The special meaning of "\n" where <i>n</i> is a digit in this context, can be suppressed by preceding it by a backslash. For each other backslash ('\') encountered, the following character shall lose its special meaning (if any). The meaning of a '\ ' immediately followed by any character other than '&', '\', a digit, or the delimiter character used for this command, is unspecified.
103120 103121 103122 103123		A line can be split by substituting a <newline> into it. The application shall escape the <newline> in the replacement by preceding it by a backslash. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces. Any backslash used to alter the default

103124		meaning of a subsequent character shall be discarded from the BRE or the
103125		replacement before evaluating the BRE or using the replacement.
103126		The value of <i>flags</i> shall be zero or more of:
103127	<i>n</i>	Substitute for the <i>n</i> th occurrence only of the BRE found within the
103128		pattern space.
103129	g	Globally substitute for all non-overlapping instances of the BRE
103130		rather than just the first one. If both g and <i>n</i> are specified, the results
103131		are unspecified.
103132	p	Write the pattern space to standard output if a replacement was
103133		made.
103134	w <i>wfile</i>	Write. Append the pattern space to <i>wfile</i> if a replacement was made.
103135		A conforming application shall precede the <i>wfile</i> argument with one
103136		or more <blank>s. If the w flag is not the last flag value given in a
103137		concatenation of multiple flag values, the results are undefined.
103138	[2addr]t [<i>label</i>]	
103139		Test. Branch to the : command verb bearing the <i>label</i> if any substitutions have been
103140		made since the most recent reading of an input line or execution of a t . If <i>label</i> is
103141		not specified, branch to the end of the script.
103142	[2addr]w <i>wfile</i>	
103143		Append (write) the pattern space to <i>wfile</i> .
103144	[2addr]x	Exchange the contents of the pattern and hold spaces.
103145	[2addr]y / <i>string1</i> / <i>string2</i> /	
103146		Replace all occurrences of characters in <i>string1</i> with the corresponding characters
103147		in <i>string2</i> . If a backslash followed by an 'n' appear in <i>string1</i> or <i>string2</i> , the two
103148		characters shall be handled as a single <newline>. If the number of characters in
103149		<i>string1</i> and <i>string2</i> are not equal, or if any of the characters in <i>string1</i> appear more
103150		than once, the results are undefined. Any character other than backslash or
103151		<newline> can be used instead of slash to delimit the strings. If the delimiter is not
103152		'n', within <i>string1</i> and <i>string2</i> , the delimiter itself can be used as a literal character
103153		if it is preceded by a backslash. If a backslash character is immediately followed by
103154		a backslash character in <i>string1</i> or <i>string2</i> , the two backslash characters shall be
103155		counted as a single literal backslash character. The meaning of a backslash
103156		followed by any character that is not 'n', a backslash, or the delimiter character is
103157		undefined.
103158	[0addr]:label	Do nothing. This command bears a <i>label</i> to which the b and t commands branch.
103159	[1addr]=	Write the following to standard output:
103160		"%d\n", <current line number>
103161	[0addr]	Ignore this empty command.
103162	[0addr]#	Ignore the '#' and the remainder of the line (treat them as a comment), with the
103163		single exception that if the first two characters in the script are "#n", the default
103164		output shall be suppressed; this shall be the equivalent of specifying -n on the
103165		command line.

EXIT STATUS

The following exit values shall be returned:

103168 0 Successful completion.

103169 >0 An error occurred.

103170 CONSEQUENCES OF ERRORS

103171 Default.

103172 APPLICATION USAGE

103173 Regular expressions match entire strings, not just individual lines, but a <newline> is matched
103174 by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression
103175 in POSIX.1-200x. Also note that '\n' cannot be used to match a <newline> at the end of an
103176 arbitrary input line; <newline>s appear in the pattern space as a result of the N editing
103177 command.

103178 EXAMPLES

103179 This *sed* script simulates the BSD *cat -s* command, squeezing excess empty lines from standard |
103180 input.

```
103181 sed -n '  
103182 # Write non-empty lines.  
103183 ./ {  
103184     p  
103185     d  
103186 }  
103187 # Write a single empty line, then look for more empty lines.  
103188 /^$/ p  
103189 # Get next line, discard the held <newline> (empty line),  
103190 # and look for more empty lines.  
103191 :Empty  
103192 /^$/ {  
103193     N  
103194     s/./.  
103195     b Empty  
103196 }  
103197 # Write the non-empty line before going back to search  
103198 # for the first in a set of empty lines.  
103199     p  
103200 '
```

103201 The following *sed* command is a much simpler method of squeezing empty lines, although it is +
103202 not quite the same as *cat -s* since it removes any initial empty lines: +

```
103203 sed -n './,/^$/p'
```

103204 RATIONALE

103205 This volume of POSIX.1-200x requires implementations to support at least ten distinct *wfiles*,
103206 matching historical practice on many implementations. Implementations are encouraged to
103207 support more, but conforming applications should not exceed this limit.

103208 The exit status codes specified here are different from those in System V. System V returns 2 for
103209 garbled *sed* commands, but returns zero with its usage message or if the input file could not be
103210 opened. The standard developers considered this to be a bug.

103211 The manner in which the I command writes non-printable characters was changed to avoid the
103212 historical backspace-overstrike method, and other requirements to achieve unambiguous output
103213 were added. See the RATIONALE for *ed* for details of the format chosen, which is the same as
103214 that chosen for *sed*.

103215 This volume of POSIX.1-200x requires implementations to provide pattern and hold spaces of at
103216 least 8 192 bytes, larger than the 4 000 bytes spaces used by some historical implementations, but

103217 less than the 20 480 bytes limit used in an early proposal. Implementations are encouraged to
103218 allocate dynamically larger pattern and hold spaces as needed.

103219 The requirements for acceptance of <blank>s and <space>s in command lines has been made
103220 more explicit than in early proposals to describe clearly the historical practice and to remove
103221 confusion about the phrase “protect initial blanks [*sic*] and tabs from the stripping that is done
103222 on every script line” that appears in much of the historical documentation of the *sed* utility
103223 description of text. (Not all implementations are known to have stripped <blank>s from text
103224 lines, although they all have allowed leading <blank>s preceding the address on a command
103225 line.)

103226 The treatment of ‘#’ comments differs from the SVID which only allows a comment as the first
103227 line of the script, but matches BSD-derived implementations. The comment character is treated
103228 as a command, and it has the same properties in terms of being accepted with leading <blank>s;
103229 the BSD implementation has historically supported this.

103230 Early proposals required that a *script_file* have at least one non-comment line. Some historical
103231 implementations have behaved in unexpected ways if this were not the case. The standard
103232 developers considered that this was incorrect behavior and that application developers should
103233 not have to avoid this feature. A correct implementation of this volume of POSIX.1-200x shall
103234 permit *script_files* that consist only of comment lines.

103235 Early proposals indicated that if **-e** and **-f** options were intermixed, all **-e** options were
103236 processed before any **-f** options. This has been changed to process them in the order presented
103237 because it matches historical practice and is more intuitive.

103238 The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems
103239 when the default output is suppressed. In the two examples:

```
103240 echo a | sed 's/a/A/p'
103241 echo a | sed -n 's/a/A/p'
```

103242 this volume of POSIX.1-200x, BSD, System V documentation, and the SVID indicate that the first
103243 example should write two lines with **A**, whereas the second should write one. Some System V
103244 systems write the **A** only once in both examples because the **p** flag is ignored if the **-n** option is
103245 not specified.

103246 This is a case of a diametrical difference between systems that could not be reconciled through
103247 the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V
103248 documentation behavior was adopted for this volume of POSIX.1-200x because:

- 103249 • No known documentation for any historic system describes the interaction between the **p**
103250 flag and the **-n** option.
- 103251 • The selected behavior is more correct as there is no technical justification for any
103252 interaction between the **p** flag and the **-n** option. A relationship between **-n** and the **p** flag
103253 might imply that they are only used together, but this ignores valid scripts that interrupt
103254 the cyclical nature of the processing through the use of the **D**, **d**, **q**, or branching
103255 commands. Such scripts rely on the **p** suffix to write the pattern space because they do not
103256 make use of the default output at the “bottom” of the script.
- 103257 • Because the **-n** option makes the **p** flag unnecessary, any interaction would only be useful
103258 if *sed* scripts were written to run both with and without the **-n** option. This is believed to
103259 be unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to
103260 be unnecessary. Because the interaction was not documented, the likelihood of a
103261 programmer discovering the interaction and depending on it is further decreased.
- 103262 • Finally, scripts that break under the specified behavior produce too much output instead of
103263 too little, which is easier to diagnose and correct.

103264 The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in

103265 an early proposal. This limit has been removed because there is no reason an editor processing
 103266 lines of {LINE_MAX} length should have this restriction. The command *s/a/A/2047* should be
 103267 able to substitute the 2047th occurrence of *a* on a line.

103268 The *b*, *t*, and *:* commands are documented to ignore leading white space, but no mention is
 103269 made of trailing white space. Historical implementations of *sed* assigned different locations to
 103270 the labels '*x*' and "*x*". This is not useful, and leads to subtle programming errors, but it is
 103271 historical practice, and changing it could theoretically break working scripts. Implementors are
 103272 encouraged to provide warning messages about labels that are never used or jumps to labels
 103273 that do not exist.

103274 Historically, the *sed !* and *}* editing commands did not permit multiple commands on a single
 103275 line using a semicolon as a command delimiter. Implementations are permitted, but not
 103276 required, to support this extension.

103277 Earlier versions of this standard allowed for implementations with bytes other than eight bits,
 103278 but this has been modified in this version.

103279 FUTURE DIRECTIONS

103280 None.

103281 SEE ALSO

103282 *awk*, *ed*, *grep*

103283 XBD Table 5-1 (on page 108), Chapter 8 (on page 159), Section 9.3 (on page 169), Section 12.2 (on
 103284 page 201)

103285 CHANGE HISTORY

103286 First released in Issue 2.

103287 Issue 5

103288 The FUTURE DIRECTIONS section is added.

103289 Issue 6

103290 The following new requirements on POSIX implementations derive from alignment with the
 103291 Single UNIX Specification:

- 103292 • Implementations are required to support at least ten *wfile* arguments in an editing
 103293 command.

103294 The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

103295 IEEE PASC Interpretation 1003.2 #190 is applied.

103296 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the backslash escape
 103297 sequences in a replacement string for a BRE.

103298 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/28 is applied, removing text describing
 103299 behavior on systems with bytes consisting of more than eight bits.

103300 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/29 is applied, making an editorial
 103301 correction within the Editing Commands in *sed* section.

103302 Issue 7

103303 Austin Group Interpretations 1003.1-2001 #006 and #036 are applied.

103304 SD5-XCU-ERN-97 and SD5-XCU-ERN-123 are applied, updating the SYNOPSIS.

103305 A second example is added.

103306 **NAME**
 103307 sh — shell, the standard command language interpreter

103308 **SYNOPSIS**

103309 sh [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 103310 [*command_file* [*argument*...]]
 103311 sh -c [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 103312 *command_string* [*command_name* [*argument*...]]
 103313 sh -s [-abCefhimnuvx] [-o *option*]... [+abCefhimnuvx] [+o *option*]...
 103314 [*argument*...]

103315 **DESCRIPTION**

103316 The *sh* utility is a command language interpreter that shall execute commands read from a
 103317 command line string, the standard input, or a specified file. The application shall ensure that the
 103318 commands to be executed are expressed in the language described in [Chapter 2](#) (on page 2245).

103319 Pathname expansion shall not fail due to the size of a file.

103320 Shell input and output redirections have an implementation-defined offset maximum that is
 103321 established in the open file description.

103322 **OPTIONS**

103323 The *sh* utility shall conform to XBD [Section 12.2](#) (on page 201), with an extension for support of a
 103324 leading plus sign ('+') as noted below.

103325 The **-a**, **-b**, **-C**, **-e**, **-f**, **-m**, **-n**, **-o option**, **-u**, **-v**, and **-x** options are described as part of the *set*
 103326 utility in [Section 2.14](#) (on page 2280). The option letters derived from the *set* special built-in shall
 103327 also be accepted with a leading plus sign ('+') instead of a leading hyphen (meaning the
 103328 reverse case of the option as described in this volume of POSIX.1-200x).

103329 The following additional options shall be supported:

103330 **-c** Read commands from the *command_string* operand. Set the value of special
 103331 parameter 0 (see [Section 2.5.2](#), on page 2250) from the value of the *command_name*
 103332 operand and the positional parameters (\$1, \$2, and so on) in sequence from the
 103333 remaining *argument* operands. No commands shall be read from the standard
 103334 input.

103335 **-i** Specify that the shell is *interactive*; see below. An implementation may treat
 103336 specifying the **-i** option as an error if the real user ID of the calling process does
 103337 not equal the effective user ID or if the real group ID does not equal the effective
 103338 group ID.

103339 **-s** Read commands from the standard input.

103340 If there are no operands and the **-c** option is not specified, the **-s** option shall be assumed.

103341 If the **-i** option is present, or if there are no operands and the shell's standard input and
 103342 standard error are attached to a terminal, the shell is considered to be *interactive*.

103343 **OPERANDS**

103344 The following operands shall be supported:

103345 **-** A single hyphen shall be treated as the first operand and then ignored. If both **'-'**
 103346 and **"--"** are given as arguments, or if other operands precede the single hyphen,
 103347 the results are undefined.

- 103348 *argument* The positional parameters (\$1, \$2, and so on) shall be set to *arguments*, if any.
- 103349 *command_file* The pathname of a file containing commands. If the pathname contains one or
103350 more slash characters, the implementation attempts to read that file; the file need
103351 not be executable. If the pathname does not contain a slash character:
- The implementation shall attempt to read that file from the current working
103352 directory; the file need not be executable.
 - If the file is not in the current working directory, the implementation may
103353 perform a search for an executable file using the value of *PATH*, as described
103354 in [Section 2.9.1.1](#) (on page 2264).

103357 Special parameter 0 (see [Section 2.5.2](#), on page 2250) shall be set to the value of
103358 *command_file*. If *sh* is called using a synopsis form that omits *command_file*, special
103359 parameter 0 shall be set to the value of the first argument passed to *sh* from its
103360 parent (for example, *argv*[0] for a C program), which is normally a pathname used
103361 to execute the *sh* utility.

103362 *command_name*
103363 A string assigned to special parameter 0 when executing the commands in
103364 *command_string*. If *command_name* is not specified, special parameter 0 shall be set
103365 to the value of the first argument passed to *sh* from its parent (for example, *argv*[0]
103366 for a C program), which is normally a pathname used to execute the *sh* utility.

103367 *command_string*
103368 A string that shall be interpreted by the shell as one or more commands, as if the
103369 string were the argument to the *system*(*)* function defined in the System Interfaces
103370 volume of POSIX.1-200x. If the *command_string* operand is an empty string, *sh* shall
103371 exit with a zero exit status.

STDIN

- 103372 The standard input shall be used only if one of the following is true:
103373
- The *-s* option is specified.
 - The *-c* option is not specified and no operands are specified.
 - The script executes one or more commands that require input from standard input (such as
103376 a *read* command that does not redirect its input).

103377 See the INPUT FILES section.

103378
103379 When the shell is using standard input and it invokes a command that also uses standard input,
103380 the shell shall ensure that the standard input file pointer points directly after the command it has
103381 read when the command begins execution. It shall not read ahead in such a manner that any
103382 characters intended to be read by the invoked command are consumed by the shell (whether
103383 interpreted by the shell or not) or that characters that are not read by the invoked command are
103384 not seen by the shell. When the command expecting to read standard input is started
103385 asynchronously by an interactive shell, it is unspecified whether characters are read by the
103386 command or interpreted by the shell.

103387 If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh*
103388 shall enable blocking reads on standard input. This shall remain in effect when the command
103389 completes.

INPUT FILES

103390 The input file shall be a text file, except that line lengths shall be unlimited. If the input file is
103391 empty or consists solely of blank lines or comments, or both, *sh* shall exit with a zero exit status.
103392

ENVIRONMENT VARIABLES

103393			
103394			The following environment variables shall affect the execution of <i>sh</i> :
103395		<i>ENV</i>	This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2 , on page 2254) by the shell, and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of <i>ENV</i> is not an absolute pathname, the results are unspecified. <i>ENV</i> shall be ignored if the real and effective user IDs or real and effective group IDs of the process are different.
103402	UP	<i>FCEDIT</i>	This variable, when expanded by the shell, shall determine the default value for the <i>-e editor</i> option's <i>editor</i> option-argument. If <i>FCEDIT</i> is null or unset, <i>ed</i> shall be used as the editor.
103403			
103404			
103405	UP	<i>HISTFILE</i>	Determine a pathname naming a command history file. If the <i>HISTFILE</i> variable is not set, the shell may attempt to access or create a file <i>.sh_history</i> in the directory referred to by the <i>HOME</i> environment variable. If the shell cannot obtain both read and write access to, or create, the history file, it shall use an unspecified mechanism that allows the history to operate properly. (References to history "file" in this section shall be understood to mean this unspecified mechanism in such cases.) An implementation may choose to access this variable only when initializing the history file; this initialization shall occur when <i>fc</i> or <i>sh</i> first attempt to retrieve entries from, or add entries to, the file, as the result of commands issued by the user, the file named by the <i>ENV</i> variable, or implementation-defined system start-up files. Implementations may choose to disable the history list mechanism for users with appropriate privileges who do not set <i>HISTFILE</i> ; the specific circumstances under which this occurs are implementation-defined. If more than one instance of the shell is using the same history file, it is unspecified how updates to the history file from those shells interact. As entries are deleted from the history file, they shall be deleted oldest first. It is unspecified when history file entries are physically removed from the history file.
103406			
103407			
103408			
103409			
103410			
103411			
103412			
103413			
103414			
103415			
103416			
103417			
103418			
103419			
103420			
103421			
103422		<i>HISTSIZE</i>	Determine a decimal number representing the limit to the number of previous commands that are accessible. If this variable is unset, an unspecified default greater than or equal to 128 shall be used. The maximum number of commands in the history list is unspecified, but shall be at least 128. An implementation may choose to access this variable only when initializing the history file, as described under <i>HISTFILE</i> . Therefore, it is unspecified whether changes made to <i>HISTSIZE</i> after the history file has been initialized are effective.
103423			
103424			
103425			
103426			
103427			
103428			
103429	UP	<i>HOME</i>	Determine the pathname of the user's home directory. The contents of <i>HOME</i> are used in tilde expansion as described in Section 2.6.1 (on page 2253).
103430			
103431		<i>IFS</i>	A string treated as a list of characters that is used for field splitting and to split lines into fields with the <i>read</i> command.
103432			
103433			If <i>IFS</i> is not set, it shall behave as normal for an unset variable, except that field splitting by the shell and line splitting by the <i>read</i> command shall be performed as if the value of <i>IFS</i> is <space><tab><newline>; see Section 2.6.5 (on page 2258).
103434			
103435			
103436			Implementations may ignore the value of <i>IFS</i> in the environment, or the absence of <i>IFS</i> from the environment, at the time the shell is invoked, in which case the shell shall set <i>IFS</i> to <space><tab><newline> when it is invoked.
103437			
103438			
103439		<i>LANG</i>	Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)
103440			
103441			

103442		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
103443			
103444		<i>LC_COLLATE</i>	Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.
103445			
103446			
103447		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class alpha), and the behavior of character classes within pattern matching.
103448			
103449			
103450			
103451		<i>LC_MESSAGES</i>	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
103452			
103453			
103454	UP	<i>MAIL</i>	Determine a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the <i>MAILCHECK</i> variable after the last such check. The user shall be informed only if <i>MAIL</i> is set and <i>MAILPATH</i> is not set.
103455			
103456			
103457			
103458			
103459			
103460			
103461			
103462	UP	<i>MAILCHECK</i>	Establish a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the <i>MAILPATH</i> or <i>MAIL</i> variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt.
103463			
103464			
103465			
103466			
103467	UP	<i>MAILPATH</i>	Provide a list of pathnames and optional messages separated by colons. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for <i>MAIL</i> for descriptions of mail arrival and user informing.) Each pathname can be followed by ' <i>%</i> ' and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a ' <i>%</i> ' character in the pathname is preceded by a backslash, it shall be treated as a literal ' <i>%</i> ' in the pathname. The default message is unspecified.
103468			
103469			
103470			
103471			
103472			
103473			
103474			
103475			The <i>MAILPATH</i> environment variable takes precedence over the <i>MAIL</i> variable.
103476	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
103477		<i>PATH</i>	Establish a string formatted as described in XBD Chapter 8 (on page 159), used to effect command interpretation; see Section 2.9.1.1 (on page 2264).
103478			
103479		<i>PWD</i>	This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored unless the value is an absolute pathname of the current working directory and there are no filename components of dot or dot-dot.
103480			
103481			
103482			
103483		ASYNCHRONOUS EVENTS	
103484		Default.	
103485		STDOUT	
103486		See the <i>STDERR</i> section.	

103487 **STDERR**

103488 Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode),
 103489 standard error shall be used only for diagnostic messages.

103490 **OUTPUT FILES**

103491 None.

103492 **EXTENDED DESCRIPTION**

103493 UP See [Chapter 2](#). The functionality described in the rest of the EXTENDED DESCRIPTION section
 103494 shall be provided on implementations that support the User Portability Utilities option (and the
 103495 rest of this section is not further shaded for this option).

103496 **Command History List**

103497 When the *sh* utility is being used interactively, it shall maintain a list of commands previously
 103498 entered from the terminal in the file named by the *HISTFILE* environment variable. The type,
 103499 size, and internal format of this file are unspecified. Multiple *sh* processes can share access to the
 103500 file for a user, if file access permissions allow this; see the description of the *HISTFILE*
 103501 environment variable.

103502 **Command Line Editing**

103503 When *sh* is being used interactively from a terminal, the current command and the command
 103504 history (see *fc*) can be edited using *vi*-mode command line editing. This mode uses commands,
 103505 described below, similar to a subset of those described in the *vi* utility. Implementations may
 103506 offer other command line editing modes corresponding to other editing utilities.

103507 The command *set -o vi* shall enable *vi*-mode editing and place *sh* into *vi* insert mode (see
 103508 [Command Line Editing \(vi-mode\)](#)). This command also shall disable any other editing mode
 103509 that the implementation may provide. The command *set +o vi* disables *vi*-mode editing.

103510 Certain block-mode terminals may be unable to support shell command line editing. If a
 103511 terminal is unable to provide either edit mode, it need not be possible to *set -o vi* when using the
 103512 shell on this terminal.

103513 In the following sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the
 103514 *stty* utility.

103515 **Command Line Editing (vi-mode)**

103516 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations
 103517 which modify a line affect the edit line. The edit line is always the newest line in the command
 103518 history buffer.

103519 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

103520 When in insert mode, an entered character shall be inserted into the command line, except as
 103521 noted in [vi Line Editing Insert Mode](#) (on page 3079). Upon entering *sh* and after termination of
 103522 the previous command, *sh* shall be in insert mode.

103523 Typing an escape character shall switch *sh* into command mode (see [vi Line Editing Command](#)
 103524 [Mode](#), on page 3079). In command mode, an entered character shall either invoke a defined
 103525 operation, be used as part of a multi-character operation, or be treated as an error. A character
 103526 that is not recognized as part of an editing command shall terminate any specific editing
 103527 command and shall alert the terminal. Typing the *interrupt* character in command mode shall
 103528 cause *sh* to terminate command line editing on the current command line, reissue the prompt on
 103529 the next line of the terminal, and reset the command history (see *fc*) so that the most recently
 103530 executed command is the previous command (that is, the command that was being edited when
 103531 it was interrupted is not reentered into the history).

103532 In the following sections, the phrase “move the cursor to the beginning of the word” shall mean
 103533 “move the cursor to the first character of the current word” and the phrase “move the cursor to
 103534 the end of the word” shall mean “move the cursor to the last character of the current word”. The
 103535 phrase “beginning of the command line” indicates the point between the end of the prompt
 103536 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and
 103537 the first character of the command text.

103538 **vi Line Editing Insert Mode**

103539 While in insert mode, any character typed shall be inserted in the current command line, unless
 103540 it is from the following set.

103541 <newline> Execute the current command line. If the current command line is not empty, this
 103542 line shall be entered into the command history (see *fc*).

103543 *erase* Delete the character previous to the current cursor position and move the current
 103544 cursor position back one character. In insert mode, characters shall be erased from
 103545 both the screen and the buffer when backspacing.

103546 *interrupt* Terminate command line editing with the same effects as described for
 103547 interrupting command mode; see [Command Line Editing \(vi-mode\)](#) (on page
 103548 3078).

103549 *kill* Clear all the characters from the input line.

103550 <control>-V Insert the next character input, even if the character is otherwise a special insert
 103551 mode character.

103552 <control>-W Delete the characters from the one preceding the cursor to the preceding word
 103553 boundary. The word boundary in this case is the closer to the cursor of either the
 103554 beginning of the line or a character that is in neither the **blank** nor **punct** character
 103555 classification of the current locale.

103556 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the
 103557 beginning of an input line. If *end-of-file* is entered other than at the beginning of the
 103558 line, the results are unspecified.

103559 <ESC> Place *sh* into command mode.

103560 **vi Line Editing Command Mode**

103561 In command mode for the command line editing feature, decimal digits not beginning with 0
 103562 that precede a command letter shall be remembered. Some commands use these decimal digits
 103563 as a count number that affects the operation.

103564 The term *motion command* represents one of the commands:

103565 <space> 0 b F l W ^ \$; E f T w | , B e h t

103566 If the current line is not the edit line, any command that modifies the current line shall cause the
 103567 content of the current line to replace the content of the edit line, and the current line shall
 103568 become the edit line. This replacement cannot be undone (see the **u** and **U** commands below).
 103569 The modification requested shall then be performed to the edit line. When the current line is the
 103570 edit line, the modification shall be done directly to the edit line.

103571 Any command that is preceded by *count* shall take a count (the numeric value of any preceding
 103572 decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat
 103573 by the number of times specified by the count. Also unless otherwise noted, a *count* that is out
 103574 of range is considered an error condition and shall alert the terminal, but neither the cursor
 103575 position, nor the command line, shall change.

103576 The terms *word* and *bigword* are used as defined in the *vi* description. The term *save buffer*

103577		corresponds to the term <i>unnamed buffer</i> in <i>vi</i> .
103578		The following commands shall be recognized in command mode:
103579	<newline>	Execute the current command line. If the current command line is not empty, this
103580		line shall be entered into the command history (see <i>fc</i>).
103581	<control>-L	Redraw the current command line. Position the cursor at the same location on the
103582		redrawn line.
103583	#	Insert the character '#' at the beginning of the current command line and treat the
103584		resulting edit line as a comment. This line shall be entered into the command
103585		history; see <i>fc</i> .
103586	=	Display the possible shell word expansions (see Section 2.6 , on page 2253) of the
103587		bigword at the current command line position.
103588	Note:	This does not modify the content of the current line, and therefore does not cause
103589		the current line to become the edit line.
103590		These expansions shall be displayed on subsequent terminal lines. If the bigword
103591		contains none of the characters '?', '*', or '[', an asterisk ('*') shall be
103592		implicitly assumed at the end. If any directories are matched, these expansions
103593		shall have a '/' character appended. After the expansion, the line shall be
103594		redrawn, the cursor repositioned at the current cursor position, and <i>sh</i> shall be
103595		placed in command mode.
103596	\	Perform pathname expansion (see Section 2.6.6 , on page 2259) on the current
103597		bigword, up to the largest set of characters that can be matched uniquely. If the
103598		bigword contains none of the characters '?', '*', or '[', an asterisk ('*') shall
103599		be implicitly assumed at the end. This maximal expansion then shall replace the
103600		original bigword in the command line, and the cursor shall be placed after this
103601		expansion. If the resulting bigword completely and uniquely matches a directory, a
103602		'/' character shall be inserted directly after the bigword. If some other file is
103603		completely matched, a single <space> shall be inserted after the bigword. After
103604		this operation, <i>sh</i> shall be placed in insert mode.
103605	*	Perform pathname expansion on the current bigword and insert all expansions
103606		into the command to replace the current bigword, with each expansion separated
103607		by a single <space>. If at the end of the line, the current cursor position shall be
103608		moved to the first column position following the expansions and <i>sh</i> shall be placed
103609		in insert mode. Otherwise, the current cursor position shall be the last column
103610		position of the first character after the expansions and <i>sh</i> shall be placed in insert
103611		mode. If the current bigword contains none of the characters '?', '*', or '[',
103612		before the operation, an asterisk shall be implicitly assumed at the end.
103613	@ <i>letter</i>	Insert the value of the alias named <i>_letter</i> . The symbol <i>letter</i> represents a single
103614		alphabetic character from the portable character set; implementations may support
103615		additional characters as an extension. If the alias <i>_letter</i> contains other editing
103616		commands, these commands shall be performed as part of the insertion. If no alias
103617		<i>_letter</i> is enabled, this command shall have no effect.
103618	[<i>count</i>] [~]	Convert, if the current character is a lowercase letter, to the equivalent uppercase
103619		letter and <i>vice versa</i> , as prescribed by the current locale. The current cursor position
103620		then shall be advanced by one character. If the cursor was positioned on the last
103621		character of the line, the case conversion shall occur, but the cursor shall not
103622		advance. If the '~' command is preceded by a <i>count</i> , that number of characters
103623		shall be converted, and the cursor shall be advanced to the character position after
103624		the last character converted. If the <i>count</i> is larger than the number of characters
103625		after the cursor, this shall not be considered an error; the cursor shall advance to

103626		the last character on the line.
103627	[<i>count</i>].	Repeat the most recent non-motion command, even if it was executed on an earlier command line. If the previous command was preceded by a <i>count</i> , and no count is given on the '.' command, the count from the previous command shall be included as part of the repeated command. If the '.' command is preceded by a <i>count</i> , this shall override any <i>count</i> argument to the previous command. The <i>count</i> specified in the '.' command shall become the count for subsequent '.' commands issued without a count.
103628		
103629		
103630		
103631		
103632		
103633		
103634	[<i>number</i>]v	Invoke the <i>vi</i> editor to edit the current command line in a temporary file. When the editor exits, the commands in the temporary file shall be executed and placed in the command history. If a <i>number</i> is included, it specifies the command number in the command history to be edited, rather than the current command line.
103635		
103636		
103637		
103638	[<i>count</i>]l (ell)	
103639	[<i>count</i>]<space>	
103640		Move the current cursor position to the next character position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.
103641		
103642		
103643		
103644		
103645	[<i>count</i>]h	Move the current cursor position to the <i>count</i> th (default 1) previous character position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the count is larger than the number of characters before the cursor, this shall not be considered an error; the cursor shall move to the first character on the line.
103646		
103647		
103648		
103649		
103650	[<i>count</i>]w	Move to the start of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.
103651		
103652		
103653		
103654		
103655	[<i>count</i>]W	Move to the start of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.
103656		
103657		
103658		
103659		
103660	[<i>count</i>]e	Move to the end of the current word. If at the end of a word, move to the end of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.
103661		
103662		
103663		
103664		
103665	[<i>count</i>]E	Move to the end of the current bigword. If at the end of a bigword, move to the end of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the <i>count</i> is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.
103666		
103667		
103668		
103669		
103670	[<i>count</i>]b	Move to the beginning of the current word. If at the beginning of a word, move to the beginning of the previous word. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the <i>count</i> is larger than the number of words preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.
103671		
103672		
103673		
103674		
103675		

103676	[count]B	Move to the beginning of the current bigword. If at the beginning of a bigword, move to the beginning of the previous bigword. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the <i>count</i> is larger than the number of bigwords preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.
103677		
103678		
103679		
103680		
103681		
103682	^	Move the current cursor position to the first character on the input line that is not a <blank>.
103683		
103684	\$	Move to the last character position on the current command line.
103685	0	(Zero.) Move to the first character position on the current command line.
103686	[count]l	Move to the <i>count</i> th character position on the current command line. If no number is specified, move to the first position. The first character position shall be numbered 1. If the count is larger than the number of characters on the line, this shall not be considered an error; the cursor shall be placed on the last character on the line.
103687		
103688		
103689		
103690		
103691	[count]fc	Move to the first occurrence of the character 'c' that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character 'c' does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
103692		
103693		
103694		
103695		
103696	[count]Fc	Move to the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
103697		
103698		
103699		
103700		
103701	[count]tc	Move to the character before the first occurrence of the character 'c' that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character 'c' does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
103702		
103703		
103704		
103705		
103706	[count]Tc	Move to the character after the first occurrence of the character 'c' that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character 'c' does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.
103707		
103708		
103709		
103710		
103711	[count];	Repeat the most recent f , F , t , or T command. Any number argument on that previous command shall be ignored. Errors are those described for the repeated command.
103712		
103713		
103714	[count],	Repeat the most recent f , F , t , or T command. Any number argument on that previous command shall be ignored. However, reverse the direction of that command.
103715		
103716		
103717	a	Enter insert mode after the current cursor position. Characters that are entered shall be inserted before the next character.
103718		
103719	A	Enter insert mode after the end of the current command line.
103720	i	Enter insert mode at the current cursor position. Characters that are entered shall be inserted before the current character.
103721		

103722	I	Enter insert mode at the beginning of the current command line.
103723	R	Enter insert mode, replacing characters from the command line beginning at the
103724		current cursor position.
103725	[count]c <i>motion</i>	
103726		Delete the characters between the current cursor position and the cursor position
103727		that would result from the specified motion command. Then enter insert mode
103728		before the first character following any deleted characters. If <i>count</i> is specified, it
103729		shall be applied to the motion command. A <i>count</i> shall be ignored for the following
103730		motion commands:
103731		0 ^ \$ c
103732		If the motion command is the character 'c', the current command line shall be
103733		cleared and insert mode shall be entered. If the motion command would move the
103734		current cursor position toward the beginning of the command line, the character
103735		under the current cursor position shall not be deleted. If the motion command
103736		would move the current cursor position toward the end of the command line, the
103737		character under the current cursor position shall be deleted. If the <i>count</i> is larger
103738		than the number of characters between the current cursor position and the end of
103739		the command line toward which the motion command would move the cursor, this
103740		shall not be considered an error; all of the remaining characters in the
103741		mentioned range shall be deleted and insert mode shall be entered. If the
103742		motion command is invalid, the terminal shall be alerted, the cursor shall not be
103743		moved, and no text shall be deleted.
103744	C	Delete from the current character to the end of the line and enter insert mode at the
103745		new end-of-line.
103746	S	Clear the entire edit line and enter insert mode.
103747	[count]rc	Replace the current character with the character 'c'. With a number <i>count</i> ,
103748		replace the current and the following <i>count</i> -1 characters. After this command, the
103749		current cursor position shall be on the last character that was changed. If the <i>count</i>
103750		is larger than the number of characters after the cursor, this shall not be considered
103751		an error; all of the remaining characters shall be changed.
103752	[count]_	Append a <space> after the current character position and then append the last
103753		bigword in the previous input line after the <space>. Then enter insert mode after
103754		the last character just appended. With a number <i>count</i> , append the <i>count</i> th bigword
103755		in the previous line.
103756	[count]x	Delete the character at the current cursor position and place the deleted characters
103757		in the save buffer. If the cursor was positioned on the last character of the line, the
103758		character shall be deleted and the cursor position shall be moved to the previous
103759		character (the new last character). If the <i>count</i> is larger than the number of
103760		characters after the cursor, this shall not be considered an error; all the characters
103761		from the cursor to the end of the line shall be deleted.
103762	[count]X	Delete the character before the current cursor position and place the deleted
103763		characters in the save buffer. The character under the current cursor position shall
103764		not change. If the cursor was positioned on the first character of the line, the
103765		terminal shall be alerted, and the X command shall have no effect. If the line
103766		contained a single character, the X command shall have no effect. If the line
103767		contained no characters, the terminal shall be alerted and the cursor shall not be
103768		moved. If the <i>count</i> is larger than the number of characters before the cursor, this
103769		shall not be considered an error; all the characters from before the cursor to the
103770		beginning of the line shall be deleted.

103771	[count]d <i>motion</i>	
103772		Delete the characters between the current cursor position and the character
103773		position that would result from the motion command. A number <i>count</i> repeats the
103774		motion command <i>count</i> times. If the motion command would move toward the
103775		beginning of the command line, the character under the current cursor position
103776		shall not be deleted. If the motion command is d , the entire current command line
103777		shall be cleared. If the <i>count</i> is larger than the number of characters between the
103778		current cursor position and the end of the command line toward which the motion
103779		command would move the cursor, this shall not be considered an error; all of the
103780		remaining characters in the aforementioned range shall be deleted. The deleted
103781		characters shall be placed in the save buffer.
103782	D	Delete all characters from the current cursor position to the end of the line. The
103783		deleted characters shall be placed in the save buffer.
103784	[count]y <i>motion</i>	
103785		Yank (that is, copy) the characters from the current cursor position to the position
103786		resulting from the motion command into the save buffer. A number <i>count</i> shall be
103787		applied to the motion command. If the motion command would move toward the
103788		beginning of the command line, the character under the current cursor position
103789		shall not be included in the set of yanked characters. If the motion command is y ,
103790		the entire current command line shall be yanked into the save buffer. The current
103791		cursor position shall be unchanged. If the <i>count</i> is larger than the number of
103792		characters between the current cursor position and the end of the command line
103793		toward which the motion command would move the cursor, this shall not be
103794		considered an error; all of the remaining characters in the aforementioned range
103795		shall be yanked.
103796	Y	Yank the characters from the current cursor position to the end of the line into the
103797		save buffer. The current character position shall be unchanged.
103798	[count]p	Put a copy of the current contents of the save buffer after the current cursor
103799		position. The current cursor position shall be advanced to the last character put
103800		from the save buffer. A <i>count</i> shall indicate how many copies of the save buffer
103801		shall be put.
103802	[count]P	Put a copy of the current contents of the save buffer before the current cursor
103803		position. The current cursor position shall be moved to the last character put from
103804		the save buffer. A <i>count</i> shall indicate how many copies of the save buffer shall be
103805		put.
103806	u	Undo the last command that changed the edit line. This operation shall not undo
103807		the copy of any command line to the edit line.
103808	U	Undo all changes made to the edit line. This operation shall not undo the copy of
103809		any command line to the edit line.
103810	[count]k	
103811	[count]-	Set the current command line to be the <i>count</i> th previous command line in the shell
103812		command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be
103813		positioned on the first character of the new command. If a k or - command would
103814		retreat past the maximum number of commands in effect for this shell (affected by
103815		the <i>HISTSIZE</i> environment variable), the terminal shall be alerted, and the
103816		command shall have no effect.
103817	[count]j	
103818	[count]+	Set the current command line to be the <i>count</i> th next command line in the shell
103819		command history. If <i>count</i> is not specified, it shall default to 1. The cursor shall be
103820		positioned on the first character of the new command. If a j or + command

103821 advances past the edit line, the current command line shall be restored to the edit
103822 line and the terminal shall be alerted.

103823 **[number]G** Set the current command line to be the oldest command line stored in the shell
103824 command history. With a number *number*, set the current command line to be the
103825 command line *number* in the history. If command line *number* does not exist, the
103826 terminal shall be alerted and the command line shall not be changed.

103827 **/pattern<newline>**

103828 Move backwards through the command history, searching for the specified
103829 pattern, beginning with the previous command line. Patterns use the pattern
103830 matching notation described in Section 2.13 (on page 2278), except that the '^'
103831 character shall have special meaning when it appears as the first character of
103832 *pattern*. In this case, the '^' is discarded and the characters after the '^' shall be
103833 matched only at the beginning of a line. Commands in the command history shall
103834 be treated as strings, not as filenames. If the pattern is not found, the current
103835 command line shall be unchanged and the terminal is alerted. If it is found in a
103836 previous line, the current command line shall be set to that line and the cursor
103837 shall be set to the first character of the new command line.

103838 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If
103839 there is no previous non-empty pattern, the terminal shall be alerted and the
103840 current command line shall remain unchanged.

103841 **?pattern<newline>**

103842 Move forwards through the command history, searching for the specified pattern,
103843 beginning with the next command line. Patterns use the pattern matching notation
103844 described in Section 2.13 (on page 2278), except that the '^' character shall have
103845 special meaning when it appears as the first character of *pattern*. In this case, the
103846 '^' is discarded and the characters after the '^' shall be matched only at the
103847 beginning of a line. Commands in the command history shall be treated as strings,
103848 not as filenames. If the pattern is not found, the current command line shall be
103849 unchanged and the terminal alerted. If it is found in a following line, the current
103850 command line shall be set to that line and the cursor shall be set to the first
103851 character of the new command line.

103852 If *pattern* is empty, the last non-empty pattern provided to / or ? shall be used. If
103853 there is no previous non-empty pattern, the terminal shall be alerted and the
103854 current command line shall remain unchanged.

103855 **n** Repeat the most recent / or ? command. If there is no previous / or ?, the terminal
103856 shall be alerted and the current command line shall remain unchanged.

103857 **N** Repeat the most recent / or ? command, reversing the direction of the search. If
103858 there is no previous / or ?, the terminal shall be alerted and the current command
103859 line shall remain unchanged.

103860 EXIT STATUS

103861 The following exit values shall be returned:

103862 0 The script to be executed consisted solely of zero or more blank lines or comments, or
103863 both.

103864 1-125 A non-interactive shell detected a syntax, redirection, or variable assignment error.

103865 127 A specified *command_file* could not be found by a non-interactive shell.

103866 Otherwise, the shell shall return the exit status of the last command it invoked or attempted to
103867 invoke (see also the *exit* utility in Section 2.14, on page 2280).

CONSEQUENCES OF ERRORS

See [Section 2.8.1](#) (on page 2262).

APPLICATION USAGE

Standard input and standard error are the files that determine whether a shell is interactive when `-i` is not specified. For example:

```
sh > file
```

and:

```
sh 2> file
```

create interactive and non-interactive shells, respectively. Although both accept terminal input, the results of error conditions are different, as described in [Section 2.8.1](#) (on page 2262); in the second example a redirection error encountered by a special built-in utility aborts the shell.

A conforming application must protect its first operand, if it starts with a plus sign, by preceding it with the `--` argument that denotes the end of the options.

Applications should note that the standard *PATH* to the shell cannot be assumed to be either `/bin/sh` or `/usr/bin/sh`, and should be determined by interrogation of the *PATH* returned by *getconf PATH*, ensuring that the returned pathname is an absolute pathname and not a shell built-in.

For example, to determine the location of the standard *sh* utility:

```
command -v sh
```

On some implementations this might return:

```
/usr/xpg4/bin/sh
```

Furthermore, on systems that support executable scripts (the `#!` construct), it is recommended that applications using executable scripts install them using *getconf PATH* to determine the shell pathname and update the `#!` script appropriately as it is being installed (for example, with *sed*). For example:

```
#
# Installation time script to install correct POSIX shell pathname
#
# Get list of paths to check
#
Sifs=$IFS
Sifs_set=${IFS+y}
IFS=:
set -- $(getconf PATH)
if [ "$Sifs_set" = y ]
then
    IFS=$Sifs
else
    unset IFS
fi
#
# Check each path for 'sh'
#
for i
do
    if [ -x "${i}"/sh ]
    then
        Pshell=${i}/sh
```

```

103916         fi
103917     done
103918     #
103919     # This is the list of scripts to update. They should be of the
103920     # form '${name}.source' and will be transformed to '${name}'.
103921     # Each script should begin:
103922     #
103923     # #!INSTALLSHELLPATH
103924     #
103925     scripts="a b c"
103926     #
103927     # Transform each script
103928     #
103929     for i in ${scripts}
103930     do
103931         sed -e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
103932     done

```

EXAMPLES

1. Execute a shell command from a string:

```
sh -c "cat myfile"
```

2. Execute a shell script from a file in the current directory:

```
sh my_shell_cmds
```

RATIONALE

The *sh* utility and the *set* special built-in utility share a common set of options.

The name *IFS* was originally an abbreviation of “Input Field Separators”; however, this name is misleading as the *IFS* characters are actually used as field terminators. The KornShell ignores the contents of *IFS* upon entry to the script. A conforming application cannot rely on importing *IFS*. One justification for this, beyond security considerations, is to assist possible future shell compilers. Allowing *IFS* to be imported from the environment prevents many optimizations that might otherwise be performed via dataflow analysis of the script itself.

The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been invoked, probably by a C-language program, with standard input that has been opened using the `O_NONBLOCK` flag; see `open()` in the System Interfaces volume of POSIX.1-200x. If the shell did not reset this flag, it would immediately terminate because no input data would be available yet and that would be considered the same as end-of-file.

The options associated with a *restricted shell* (command name *rsh* and the `-r` option) were excluded because the standard developers considered that the implied level of security could not be achieved and they did not want to raise false expectations.

On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the name `-i`. When it is called by a sequence such as:

```
sh -
```

or by:

```
#! usr/bin/sh -
```

the historical systems have assumed that no option letters follow. Thus, this volume of POSIX.1-200x allows the single hyphen to mark the end of the options, in addition to the use of the regular `--` argument, because it was considered that the older practice was so pervasive. An alternative approach is taken by the KornShell, where real and effective user/group IDs

103963 must match for an interactive shell; this behavior is specifically allowed by this volume of
103964 POSIX.1-200x.

103965 **Note:** There are other problems with set-user-ID scripts that the two approaches described here do not
103966 resolve.

103967 The initialization process for the history file can be dependent on the system start-up files, in
103968 that they may contain commands that effectively preempt the user's settings of *HISTFILE* and
103969 *HISTSIZ*E. For example, function definition commands are recorded in the history file, unless
103970 the *set -o nolog* option is set. If the system administrator includes function definitions in some
103971 system start-up file called before the *ENV* file, the history file is initialized before the user gets a
103972 chance to influence its characteristics. In some historical shells, the history file is initialized just
103973 after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes
103974 made to *HISTFILE* after the history file has been initialized are effective.

103975 The default messages for the various *MAIL*-related messages are unspecified because they vary
103976 across implementations. Typical messages are:

103977 "you have mail\n"

103978 or:

103979 "you have new mail\n"

103980 It is important that the descriptions of command line editing refer to the same shell as that in
103981 POSIX.1-200x so that interactive users can also be application programmers without having to
103982 deal with programmatic differences in their two environments. It is also essential that the utility
103983 name *sh* be specified because this explicit utility name is too firmly rooted in historical practice
103984 of application programs for it to change.

103985 Consideration was given to mandating a diagnostic message when attempting to set *vi*-mode on
103986 terminals that do not support command line editing. However, it is not historical practice for the
103987 shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in
103988 all cases. Implementations are encouraged to supply diagnostics in this case whenever possible,
103989 rather than leaving the user in a state where editing commands work incorrectly.

103990 In early proposals, the KornShell-derived *emacs* mode of command line editing was included,
103991 even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant
103992 that the full *emacs* editor not be standardized because they were concerned that an attempt to
103993 standardize this very powerful environment would encourage vendors to ship strictly
103994 conforming versions lacking the extensibility required by the community. The author of the
103995 original *emacs* program also expressed his desire to omit the program. Furthermore, there were a
103996 number of historical systems that did not include *emacs*, or included it without supporting it, but
103997 there were very few that did not include and support *vi*. The shell *emacs* command line editing
103998 mode was finally omitted because it became apparent that the KornShell version and the editor
103999 being distributed with the GNU system had diverged in some respects. The author of *emacs*
104000 requested that the POSIX *emacs* mode either be deleted or have a significant number of
104001 unspecified conditions. Although the KornShell author agreed to consider changes to bring the
104002 shell into alignment, the standard developers decided to defer specification at that time. At the
104003 time, it was assumed that convergence on an acceptable definition would occur for a subsequent
104004 draft, but that has not happened, and there appears to be no impetus to do so. In any case,
104005 implementations are free to offer additional command line editing modes based on the exact
104006 models of editors their users are most comfortable with.

104007 Early proposals had the following list entry in *vi Line Editing Insert Mode* (on page 3079):

104008 \ If followed by the *erase* or *kill* character, that character shall be inserted into the input line.
104009 Otherwise, the backslash itself shall be inserted into the input line.

104010 However, this is not actually a feature of *sh* command line editing insert mode, but one of some
104011 historical terminal line drivers. Some conforming implementations continue to do this when the

104012 `stty iexten` flag is set.

104013 FUTURE DIRECTIONS

104014 None.

104015 SEE ALSO

104016 [Chapter 2](#) (on page 2245), *cd*, *echo*, *exit*, *fc*, *pwd*, *read*, *set*, *stty*, *test*, *umask*, *vi*

104017 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) |

104018 XSH *dup()*, *exec*, *exit()*, *fork()*, *open()*, *pipe()*, *signal()*, *system()*, *ulimit*, *umask*, *wait*

104019 CHANGE HISTORY

104020 First released in Issue 2.

104021 Issue 5

104022 The FUTURE DIRECTIONS section is added.

104023 Text is added to the DESCRIPTION for the Large File Summit proposal.

104024 Issue 6

104025 The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

104026 The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

104027 The following new requirements on POSIX implementations derive from alignment with the
104028 Single UNIX Specification:

- 104029 • The option letters derived from the *set* special built-in are also accepted with a leading plus
104030 sign ('+').
- 104031 • Large file extensions are added:
 - 104032 — Pathname expansion does not fail due to the size of a file.
 - 104033 — Shell input and output redirections have an implementation-defined offset maximum
104034 that is established in the open file description.

104035 In the ENVIRONMENT VARIABLES section, the text “user’s home directory” is updated to
104036 “directory referred to by the *HOME* environment variable”.

104037 Descriptions for the *ENV* and *PWD* environment variables are included to align with the
104038 IEEE P1003.2b draft standard.

104039 The normative text is reworded to avoid use of the term “must” for application requirements.

104040 Issue 7

104041 Austin Group Interpretation 1003.1-2001 #098 is applied, changing the definition of *IFS*.

104042 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS. +

104043 Minor editorial changes are made to the User Portability Utilities option shading. No normative
104044 changes are implied.

104045 Minor changes are made to the install script example in the APPLICATION USAGE section. |

104046 **NAME**

104047 sleep — suspend execution for an interval

104048 **SYNOPSIS**104049 sleep *time*104050 **DESCRIPTION**104051 The *sleep* utility shall suspend execution for at least the integral number of seconds specified by
104052 the *time* operand.104053 **OPTIONS**

104054 None.

104055 **OPERANDS**

104056 The following operand shall be supported:

104057 *time* A non-negative decimal integer specifying the number of seconds for which to
104058 suspend execution.104059 **STDIN**

104060 Not used.

104061 **INPUT FILES**

104062 None.

104063 **ENVIRONMENT VARIABLES**104064 The following environment variables shall affect the execution of *sleep*:104065 *LANG* Provide a default value for the internationalization variables that are unset or null. |
104066 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
104067 variables used to determine the values of locale categories.)104068 *LC_ALL* If set to a non-empty string value, override the values of all the other
104069 internationalization variables.104070 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
104071 characters (for example, single-byte as opposed to multi-byte characters in
104072 arguments).104073 *LC_MESSAGES*
104074 Determine the locale that should be used to affect the format and contents of
104075 diagnostic messages written to standard error.104076 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.104077 **ASYNCHRONOUS EVENTS**104078 If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

- 104079 1. Terminate normally with a zero exit status.
-
- 104080 2. Effectively ignore the signal.
-
- 104081 3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS
-
- 104082 section of Section 1.4 (on page 2235). This could include terminating with a non-zero exit
-
- 104083 status.

104084 The *sleep* utility shall take the standard action for all other signals.

104085 **STDOUT**

104086 Not used.

104087 **STDERR**

104088 The standard error shall be used only for diagnostic messages.

104089 **OUTPUT FILES**

104090 None.

104091 **EXTENDED DESCRIPTION**

104092 None.

104093 **EXIT STATUS**

104094 The following exit values shall be returned:

104095 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal
104096 was received. See the ASYNCHRONOUS EVENTS section.

104097 >0 An error occurred.

104098 **CONSEQUENCES OF ERRORS**

104099 Default.

104100 **APPLICATION USAGE**

104101 None.

104102 **EXAMPLES**104103 The *sleep* utility can be used to execute a command after a certain amount of time, as in:104104 `(sleep 105; command) &`

104105 or to execute a command every so often, as in:

104106 `while true`
104107 `do`
104108 `command`
104109 `sleep 37`
104110 `done`104111 **RATIONALE**104112 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because
104113 most implementations of this utility rely on the arrival of that signal to notify them that the
104114 requested finishing time has been successfully attained. Such implementations thus do not
104115 distinguish this situation from the successful completion case. Other implementations are
104116 allowed to catch the signal and go back to sleep until the requested time expires or to provide
104117 the normal signal termination procedures.104118 As with all other utilities that take integral operands and do not specify subranges of allowed
104119 values, *sleep* is required by this volume of POSIX.1-200x to deal with *time* requests of up to
104120 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls to
104121 the delay mechanism of the underlying operating system if its argument range is less than this.104122 **FUTURE DIRECTIONS**

104123 None.

104124 **SEE ALSO**104125 *wait*

104126 XBD Chapter 8 (on page 159)

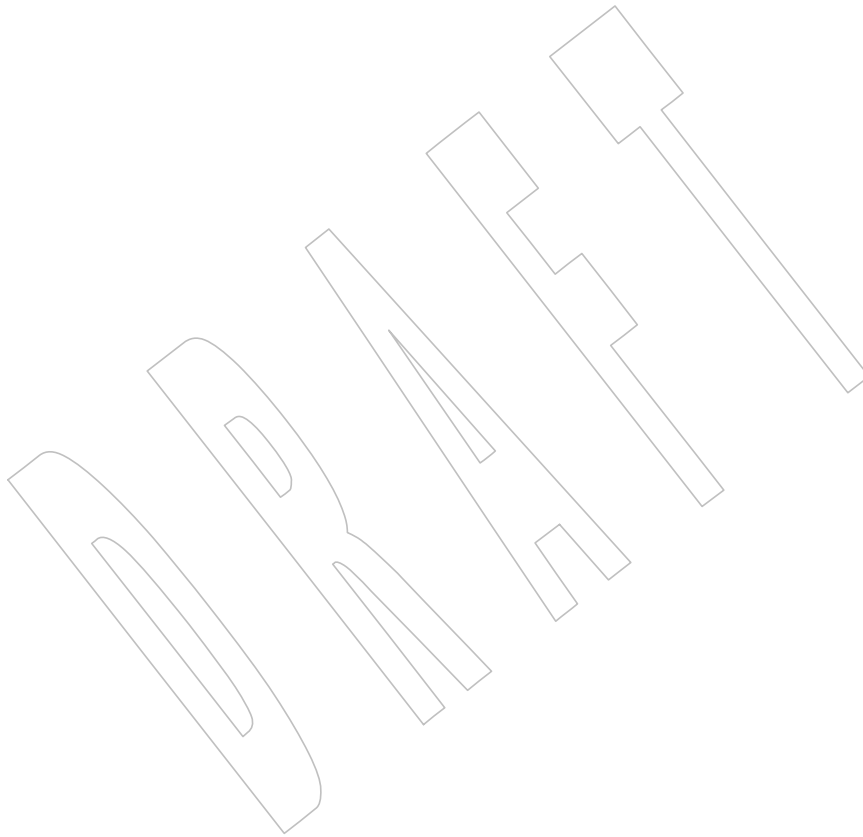
104127 XSH *alarm()*, *sleep*

104128

104129

CHANGE HISTORY

First released in Issue 2.



104130 **NAME**104131 `sort` — sort, merge, or sequence check text files104132 **SYNOPSIS**104133 `sort [-m] [-o output] [-bdfinru] [-t char] [-k keydef]... [file...]`104134 `sort [-c|-C] [-bdfinru] [-t char] [-k keydef] [file]`104135 **DESCRIPTION**104136 The *sort* utility shall perform one of the following functions:

- 104137 1. Sort lines of all the named files together and write the result to the specified output.
- 104138 2. Merge lines of all the named (presorted) files together and write the result to the specified
104139 output.
- 104140 3. Check that a single input file is correctly presorted.

104141 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no
104142 sort keys are specified, the entire line up to, but not including, the terminating <newline>), and
104143 shall be performed using the collating sequence of the current locale.

104144 **OPTIONS**

104145 The *sort* utility shall conform to XBD Section 12.2 (on page 201), except for Guideline 9, and the
104146 `-k keydef` option should follow the `-b`, `-d`, `-f`, `-i`, `-n`, and `-r` options. In addition, '+' may be
104147 recognized as an option delimiter as well as '-'. |

104148 The following options shall be supported:

104149 `-c` Check that the single input file is ordered as specified by the arguments and the
104150 collating sequence of the current locale. Output shall not be sent to standard
104151 output. The exit code shall indicate whether or not disorder was detected or an
104152 error occurred. If disorder (or, with `-u`, a duplicate key) is detected, a warning
104153 message shall be sent to standard error indicating where the disorder or duplicate
104154 key was found. |

104155 `-C` Same as `-c`, except that a warning message shall not be sent to standard error if
104156 disorder or, with `-u`, a duplicate key is detected. |

104157 `-m` Merge only; the input file shall be assumed to be already sorted.

104158 `-o output` Specify the name of an output file to be used instead of the standard output. This
104159 file can be the same as one of the input *files*.

104160 `-u` Unique: suppress all but one in each set of lines having equal keys. If used with
104161 the `-c` option, check that there are no lines with duplicate keys, in addition to
104162 checking that the input file is sorted.

104163 The following options shall override the default ordering rules. When ordering options appear
104164 independent of any key field specifications, the requested field ordering rules shall be applied
104165 globally to all sort keys. When attached to a specific key (see `-k`), the specified ordering options
104166 shall override all global ordering options for that key.

104167 `-d` Specify that only <blank>s and alphanumeric characters, according to the current
104168 setting of *LC_CTYPE*, shall be significant in comparisons. The behavior is
104169 undefined for a sort key to which `-i` or `-n` also applies.

104170 `-f` Consider all lowercase characters that have uppercase equivalents, according to
104171 the current setting of *LC_CTYPE*, to be the uppercase equivalent for the purposes
104172 of comparison.

- 104173 **-i** Ignore all characters that are non-printable, according to the current setting of
104174 *LC_CTYPE*. The behavior is undefined for a sort key for which **-n** also applies.
- 104175 **-n** Restrict the sort key to an initial numeric string, consisting of optional <blank>s,
104176 optional minus sign, and zero or more digits with an optional radix character and
104177 thousands separators (as defined in the current locale), which shall be sorted by
104178 arithmetic value. An empty digit string shall be treated as zero. Leading zeros and
104179 signs on zeros shall not affect ordering.
- 104180 **-r** Reverse the sense of comparisons.

104181 The treatment of field separators can be altered using the options:

- 104182 **-b** Ignore leading <blank>s when determining the starting and ending positions of a
104183 restricted sort key. If the **-b** option is specified before the first **-k** option, it shall be
104184 applied to all **-k** options. Otherwise, the **-b** option can be attached independently
104185 to each **-k** *field_start* or *field_end* option-argument (see below).
- 104186 **-t char** Use *char* as the field separator character; *char* shall not be considered to be part of a
104187 field (although it can be included in a sort key). Each occurrence of *char* shall be
104188 significant (for example, <*char*><*char*> delimits an empty field). If **-t** is not
104189 specified, <blank>s shall be used as default field separators; each maximal non-
104190 empty sequence of <blank>s that follows a non-<blank> shall be a field separator.

104191 Sort keys can be specified using the options:

- 104192 **-k keydef** The *keydef* argument is a restricted sort key field definition. The format of this
104193 definition is:

104194 *field_start*[*type*][, *field_end*[*type*]]

104195 where *field_start* and *field_end* define a key field restricted to a portion of the line
104196 (see the EXTENDED DESCRIPTION section), and *type* is a modifier from the list of
104197 characters 'b', 'd', 'f', 'i', 'n', 'r'. The 'b' modifier shall behave like the
104198 **-b** option, but shall apply only to the *field_start* or *field_end* to which it is attached.
104199 The other modifiers shall behave like the corresponding options, but shall apply
104200 only to the key field to which they are attached; they shall have this effect if
104201 specified with *field_start*, *field_end*, or both. If any modifier is attached to a
104202 *field_start* or to a *field_end*, no option shall apply to either. Implementations shall
104203 support at least nine occurrences of the **-k** option, which shall be significant in
104204 command line order. If no **-k** option is specified, a default sort key of the entire
104205 line shall be used.

104206 When there are multiple key fields, later keys shall be compared only after all
104207 earlier keys compare equal. Except when the **-u** option is specified, lines that
104208 otherwise compare equal shall be ordered as if none of the options **-d**, **-f**, **-i**, **-n**, or
104209 **-k** were present (but with **-r** still in effect, if it was specified) and with all bytes in
104210 the lines significant to the comparison. The order in which lines that still compare
104211 equal are written is unspecified.

104212 OPERANDS

104213 The following operand shall be supported:

- 104214 *file* A pathname of a file to be sorted, merged, or checked. If no *file* operands are
104215 specified, or if a *file* operand is '-', the standard input shall be used.

104216 STDIN

104217 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '-'.
104218 See the INPUT FILES section.

104219 **INPUT FILES**

104220 The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a
 104221 file ending with an incomplete last line.

104222 **ENVIRONMENT VARIABLES**

104223 The following environment variables shall affect the execution of *sort*:

104224 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 104225 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 104226 variables used to determine the values of locale categories.)

104227 **LC_ALL** If set to a non-empty string value, override the values of all the other
 104228 internationalization variables.

104229 **LC_COLLATE**
 104230 Determine the locale for ordering rules.

104231 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 104232 characters (for example, single-byte as opposed to multi-byte characters in
 104233 arguments and input files) and the behavior of character classification for the **-b**,
 104234 **-d**, **-f**, **-i**, and **-n** options.

104235 **LC_MESSAGES**
 104236 Determine the locale that should be used to affect the format and contents of
 104237 diagnostic messages written to standard error.

104238 **LC_NUMERIC**
 104239 Determine the locale for the definition of the radix character and thousands
 104240 separator for the **-n** option.

104241 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

104242 **ASYNCHRONOUS EVENTS**

104243 Default.

104244 **STDOUT**

104245 Unless the **-o** or **-c** options are in effect, the standard output shall contain the sorted input.

104246 **STDERR**

104247 The standard error shall be used for diagnostic messages. When **-c** is specified, if disorder is |
 104248 detected (or if **-u** is also specified and a duplicate key is detected), a message shall be written to |
 104249 the standard error which identifies the input line at which disorder (or a duplicate key) was |
 104250 detected. A warning message about correcting an incomplete last line of an input file may be
 104251 generated, but need not affect the final exit status.

104252 **OUTPUT FILES**

104253 If the **-o** option is in effect, the sorted input shall be written to the file *output*.

104254 **EXTENDED DESCRIPTION**

104255 The notation:

104256 **-k** *field_start*[*type*][,*field_end*[*type*]]

104257 shall define a key field that begins at *field_start* and ends at *field_end* inclusive, unless *field_start*
 104258 falls beyond the end of the line or after *field_end*, in which case the key field is empty. A missing
 104259 *field_end* shall mean the last character of the line.

104260 A field comprises a maximal sequence of non-separating characters and, in the absence of option
 104261 **-t**, any preceding field separator.

104262 The *field_start* portion of the *keydef* option-argument shall have the form:

104263 *field_number*[.*first_character*]

104264 Fields and characters within fields shall be numbered starting with 1. The *field_number* and
 104265 *first_character* pieces, interpreted as positive decimal integers, shall specify the first character to
 104266 be used as part of a sort key. If *first_character* is omitted, it shall refer to the first character of the
 104267 field.

104268 The *field_end* portion of the *keydef* option-argument shall have the form:

104269 *field_number*[. *last_character*]

104270 The *field_number* shall be as described above for *field_start*. The *last_character* piece, interpreted
 104271 as a non-negative decimal integer, shall specify the last character to be used as part of the sort
 104272 key. If *last_character* evaluates to zero or *last_character* is omitted, it shall refer to the last
 104273 character of the field specified by *field_number*.

104274 If the **-b** option or **b** type modifier is in effect, characters within a field shall be counted from the
 104275 first non-<blank> in the field. (This shall apply separately to *first_character* and *last_character*.)

104276 EXIT STATUS

104277 The following exit values shall be returned:

- 104278 0 All input files were output successfully, or **-c** was specified and the input file was correctly
 104279 sorted.
- 104280 1 Under the **-c** option, the file was not ordered as specified, or if the **-c** and **-u** options were
 104281 both specified, two input lines were found with equal keys.
- 104282 >1 An error occurred.

104283 CONSEQUENCES OF ERRORS

104284 Default.

104285 APPLICATION USAGE

104286 The default value for **-t**, <blank>, has different properties from, for example, **-t"<space>"**. If a
 104287 line contains:

104288 <space><space>foo

104289 the following treatment would occur with default separation as opposed to specifically selecting
 104290 a <space>:

Field	Default	-t "<space>"
1	<space><space>foo	empty
2	empty	empty
3	empty	foo

104295 The leading field separator itself is included in a field when **-t** is not used. For example, this
 104296 command returns an exit status of zero, meaning the input was already sorted:

```
104297 sort -c -k 2 <<eof
104298 y<tab>b
104299 x<space>a
104300 eof
```

104301 (assuming that a <tab> precedes the <space> in the current collating sequence). The field
 104302 separator is not included in a field when it is explicitly set via **-t**. This is historical practice and
 104303 allows usage such as:

```
104304 sort -t "|" -k 2n <<eof
104305 Atlanta|425022|Georgia
104306 Birmingham|284413|Alabama
104307 Columbia|100385|South Carolina
104308 eof
```


104309 where the second field can be correctly sorted numerically without regard to the non-numeric
104310 field separator.

104311 The wording in the OPTIONS section clarifies that the **-b**, **-d**, **-f**, **-i**, **-n**, and **-r** options have to
104312 come before the first sort key specified if they are intended to apply to all specified keys. The
104313 way it is described in this volume of POSIX.1-200x matches historical practice, not historical
104314 documentation. The results are unspecified if these options are specified after a **-k** option.

104315 The **-f** option might not work as expected in locales where there is not a one-to-one mapping
104316 between an uppercase and a lowercase letter.

104317 EXAMPLES

- 104318 1. The following command sorts the contents of **infile** with the second field as the sort key:

```
104319 sort -k 2,2 infile
```

- 104320 2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**,
104321 placing the output in **outfile** and using the second character of the second field as the sort
104322 key (assuming that the first character of the second field is the field separator):

```
104323 sort -r -o outfile -k 2.2,2.2 infile1 infile2
```

- 104324 3. The following command sorts the contents of **infile1** and **infile2** using the second
104325 non-**<blank>** of the second field as the sort key:

```
104326 sort -k 2.2b,2.2b infile1 infile2
```

- 104327 4. The following command prints the System V password file (user database) sorted by the
104328 numeric user ID (the third colon-separated field):

```
104329 sort -t : -k 3,3n /etc/passwd
```

- 104330 5. The following command prints the lines of the already sorted file **infile**, suppressing all
104331 but one occurrence of lines having the same third field:

```
104332 sort -um -k 3.1,3.0 infile
```

104333 RATIONALE

104334 Examples in some historical documentation state that options **-um** with one input file keep the
104335 first in each set of lines with equal keys. This behavior was deemed to be an implementation
104336 artifact and was not standardized.

104337 The **-z** option was omitted; it is not standard practice on most systems and is inconsistent with
104338 using *sort* to sort several files individually and then merge them together. The text concerning **-z**
104339 in historical documentation appeared to require implementations to determine the proper buffer
104340 length during the sort phase of operation, but not during the merge.

104341 The **-y** option was omitted because of non-portability. The **-M** option, present in System V, was
104342 omitted because of non-portability in international usage.

104343 An undocumented **-T** option exists in some implementations. It is used to specify a directory for
104344 intermediate files. Implementations are encouraged to support the use of the *TMPDIR*
104345 environment variable instead of adding an option to support this functionality.

104346 The **-k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is
104347 not consistent with other utility conventions. Second, it did not meet syntax guideline
104348 requirements.

104349 Historical documentation indicates that “setting **-n** implies **-b**”. The description of **-n** already
104350 states that optional leading **<blank>**s are tolerated in doing the comparison. If **-b** is enabled,
104351 rather than implied, by **-n**, this has unusual side effects. When a character offset is used in a
104352 column of numbers (for example, to sort modulo 100), that offset is measured relative to the
104353 most significant digit, not to the column. Based upon a recommendation from the author of the

104354 original *sort* utility, the `-b` implication has been omitted from this volume of POSIX.1-200x, and
 104355 an application wishing to achieve the previously mentioned side effects has to code the `-b` flag
 104356 explicitly.

104357 Earlier versions of this standard allowed the `-o` option to appear after operands. Historical
 104358 practice allowed all options to be interspersed with operands. This version of the standard
 104359 allows implementations to accept options after operands but conforming applications should
 104360 not use this form.

104361 Earlier versions of this standard also allowed the `-number` and `+number` options. These options |
 104362 are no longer specified by POSIX.1-200x but may be present in some implementations.

104363 Historical implementations produced a message on standard error when `-c` was specified and +
 104364 disorder was detected, and when `-c` and `-u` were specified and a duplicate key was detected. An +
 104365 earlier version of this standard contained wording that did not make it clear that this message +
 104366 was allowed and some implementations removed this message to be sure that they conformed +
 104367 to the standard's requirements. Confronted with this difference in behavior, interactive users +
 104368 that wanted to be sure that they got visual feedback instead of just exit code 1 could have used a +
 104369 command like:

```
104370 sort -c file || echo disorder +
```

104371 whether or not the *sort* utility provided a message in this case. But, it was not easy for a user to +
 104372 find where the disorder or duplicate key occurred on implementations that do not produce a +
 104373 message, especially when some parts of the input line were not part of the key and when one or +
 104374 more of the `-b`, `-d`, `-f`, `-i`, `-n`, or `-r` options or *keydef* type modifiers were in use. POSIX.1-200x +
 104375 requires a message to be produced in this case. POSIX.1-200x also contains the `-C` option giving +
 104376 users the ability to choose either behavior. +

104377 When a disorder or duplicate is found when the `-c` option is specified, some implementations +
 104378 print a message containing the first line that is out of order or contains a duplicate key; others +
 104379 print a message specifying the line number of the offending line. This standard allows either +
 104380 type of message.

104381 FUTURE DIRECTIONS

104382 None.

104383 SEE ALSO

104384 [comm](#), [join](#), [uniq](#)

104385 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) |

104386 XSH [toupเปอร์\(\)](#)

104387 CHANGE HISTORY

104388 First released in Issue 2.

104389 Issue 6

104390 IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

104391 IEEE PASC Interpretation 1003.2 #168 is applied.

104392 Issue 7

104393 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that Guideline 9 of the Utility -
 104394 Syntax Guidelines does not apply and noting that '+' may be recognized as an option delimiter.

104395 Austin Group Interpretation 1003.1-2001 #120 is applied, clarifying the use of the `-c` option and +
 104396 introducing the `-C` option. +

104397 XCU-ERN-81 is applied, modifying the description of the `-i` option. +

104398 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104399 **NAME**
 104400 `split` — split files into pieces

104401 **SYNOPSIS**
 104402 `split [-l line_count] [-a suffix_length] [file[name]]`
 104403 `split -b n[k|m] [-a suffix_length] [file[name]]`

104404 **DESCRIPTION**
 104405 The *split* utility shall read an input file and write one or more output files. The default size of
 104406 each output file shall be 1 000 lines. The size of the output files can be modified by specification
 104407 of the `-b` or `-l` options. Each output file shall be created with a unique suffix. The suffix shall
 104408 consist of exactly *suffix_length* lowercase letters from the POSIX locale. The letters of the suffix
 104409 shall be used as if they were a base-26 digit system, with the first suffix to be created consisting
 104410 of all 'a' characters, the second with a 'b' replacing the last 'a', and so on, until a name of all
 104411 'z' characters is created. By default, the names of the output files shall be 'x', followed by a
 104412 two-character suffix from the character set as described above, starting with "aa", "ab", "ac",
 104413 and so on, and continuing until the suffix "zz", for a maximum of 676 files.

104414 If the number of files required exceeds the maximum allowed by the suffix length provided,
 104415 such that the last allowable file would be larger than the requested size, the *split* utility shall fail
 104416 after creating the last file with a valid suffix; *split* shall not delete the files it created with valid
 104417 suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the
 104418 input file, and may be smaller than the requested size.

104419 **OPTIONS**
 104420 The *split* utility shall conform to XBD [Section 12.2](#) (on page 201).

104421 The following options shall be supported:

104422 `-a suffix_length`
 104423 Use *suffix_length* letters to form the suffix portion of the filenames of the split file. If
 104424 `-a` is not specified, the default suffix length shall be two. If the sum of the *name*
 104425 operand and the *suffix_length* option-argument would create a filename exceeding
 104426 {NAME_MAX} bytes, an error shall result; *split* shall exit with a diagnostic message
 104427 and no files shall be created.

104428 `-b n` Split a file into pieces *n* bytes in size.

104429 `-b nk` Split a file into pieces *n**1 024 bytes in size.

104430 `-b nm` Split a file into pieces *n**1 048 576 bytes in size.

104431 `-l line_count` Specify the number of lines in each resulting file piece. The *line_count* argument is
 104432 an unsigned decimal integer. The default is 1 000. If the input does not end with a
 104433 <newline>, the partial line shall be included in the last output file.

104434 **OPERANDS**
 104435 The following operands shall be supported:

104436 *file* The pathname of the ordinary file to be split. If no input file is given or *file* is '-',
 104437 the standard input shall be used.

104438 *name* The prefix to be used for each of the files resulting from the split operation. If no
 104439 *name* argument is given, 'x' shall be used as the prefix of the output files. The
 104440 combined length of the basename of *prefix* and *suffix_length* cannot exceed
 104441 {NAME_MAX} bytes. See the OPTIONS section.

104442 **STDIN**

104443 See the INPUT FILES section.

104444 **INPUT FILES**

104445 Any file can be used as input.

104446 **ENVIRONMENT VARIABLES**104447 The following environment variables shall affect the execution of *split*:

104448 **LANG** Provide a default value for the internationalization variables that are unset or null. |
 104449 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 104450 variables used to determine the values of locale categories.)

104451 **LC_ALL** If set to a non-empty string value, override the values of all the other
 104452 internationalization variables.

104453 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 104454 characters (for example, single-byte as opposed to multi-byte characters in
 104455 arguments and input files).

104456 **LC_MESSAGES**
 104457 Determine the locale that should be used to affect the format and contents of
 104458 diagnostic messages written to standard error.

104459 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

104460 **ASYNCHRONOUS EVENTS**

104461 Default.

104462 **STDOUT**

104463 Not used.

104464 **STDERR**

104465 The standard error shall be used only for diagnostic messages.

104466 **OUTPUT FILES**

104467 The output files contain portions of the original input file; otherwise, unchanged.

104468 **EXTENDED DESCRIPTION**

104469 None.

104470 **EXIT STATUS**

104471 The following exit values shall be returned:

104472 0 Successful completion.

104473 >0 An error occurred.

104474 **CONSEQUENCES OF ERRORS**

104475 Default.

104476 **APPLICATION USAGE**

104477 None.

104478 **EXAMPLES**104479 In the following examples **foo** is a text file that contains 5 000 lines.104480 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:104481 `split foo`

104482 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,
 104483 **xaab**, **xaac**, **xaad**, and **xaae**:

104484 `split -a 3 foo`

- 104485 3. Create three files with four-letter suffixes and a supplied prefix, **bar_aaa**, **bar_aaab**, and
 104486 **bar_aaac**:
- ```
104487 split -a 4 -l 2000 foo bar_
```
- 104488 4. Create as many files as are necessary to contain at most 20\*1024 bytes, each with the  
 104489 default prefix of **x** and a five-letter suffix:
- ```
104490 split -a 5 -b 20k foo
```

RATIONALE

104491 The **-b** option was added to provide a mechanism for splitting files other than by lines. While
 104492 most uses of the **-b** option are for transmitting files over networks, some believed it would have
 104493 additional uses.
 104494

104495 The **-a** option was added to overcome the limitation of being able to create only 676 files.

104496 Consideration was given to deleting this utility, using the rationale that the functionality
 104497 provided by this utility is available via the *csplit* utility (see *csplit*). Upon reconsideration of the
 104498 purpose of the User Portability Utilities option, it was decided to retain both this utility and the
 104499 *csplit* utility because users use both utilities and have historical expectations of their behavior.
 104500 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical
 104501 *csplit*.

104502 The text "*split* shall not delete the files it created with valid suffixes" would normally be
 104503 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the
 104504 historical behavior of *split* is made explicit to avoid misinterpretation.

104505 Earlier versions of this standard allowed a *-line_count* option. This form is no longer specified by
 104506 POSIX.1-200x but may be present in some implementations.

FUTURE DIRECTIONS

104507 None.
 104508

SEE ALSO

104509 *csplit*
 104510

104511 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

CHANGE HISTORY

104512 First released in Issue 2.
 104513

Issue 6

104514 This utility is marked as part of the User Portability Utilities option.

104515 The APPLICATION USAGE section is added.

104516 The obsolescent SYNOPSIS is removed.
 104517

Issue 7

104518 Austin Group Interpretation 1003.1-2001 #027 is applied.

104519 The *split* utility is moved from the User Portability Utilities option to the Base. User Portability
 104520 Utilities is now an option for interactive utilities.
 104521

104522 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104523 **NAME**

104524 strings — find printable strings in files

104525 **SYNOPSIS**104526 strings [-a] [-t *format*] [-n *number*] [*file...*]104527 **DESCRIPTION**

104528 The *strings* utility shall look for printable strings in regular files and shall write those strings to
 104529 standard output. A printable string is any sequence of four (by default) or more printable
 104530 characters terminated by a <newline> or NUL character. Additional implementation-defined
 104531 strings may be written; see *localedef*.

104532 If the first argument is '- ', the results are unspecified.

104533 **OPTIONS**104534 The *strings* utility shall conform to XBD Section 12.2 (on page 201), except for the unspecified
104535 usage of '- '.

104536 The following options shall be supported:

104537 **-a** Scan files in their entirety. If **-a** is not specified, it is implementation-defined what
104538 portion of each file is scanned for strings.104539 **-n *number*** Specify the minimum string length, where the *number* argument is a positive
104540 decimal integer. The default shall be 4.104541 **-t *format*** Write each string preceded by its byte offset from the start of the file. The format
104542 shall be dependent on the single character used as the *format* option-argument:

104543 d The offset shall be written in decimal.

104544 o The offset shall be written in octal.

104545 x The offset shall be written in hexadecimal.

104546 **OPERANDS**

104547 The following operand shall be supported:

104548 *file* A pathname of a regular file to be used as input. If no *file* operand is specified, the
104549 *strings* utility shall read from the standard input.104550 **STDIN**

104551 See the INPUT FILES section.

104552 **INPUT FILES**104553 The input files named by the utility arguments or the standard input shall be regular files of any
104554 format.104555 **ENVIRONMENT VARIABLES**104556 The following environment variables shall affect the execution of *strings*:104557 **LANG** Provide a default value for the internationalization variables that are unset or null.
104558 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
104559 variables used to determine the values of locale categories.)104560 **LC_ALL** If set to a non-empty string value, override the values of all the other
104561 internationalization variables.104562 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
104563 characters (for example, single-byte as opposed to multi-byte characters in
104564 arguments and input files) and to identify printable strings.

- 104565 **LC_MESSAGES**
- 104566 Determine the locale that should be used to affect the format and contents of
- 104567 diagnostic messages written to standard error.
- 104568 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 104569 **ASYNCHRONOUS EVENTS**
- 104570 Default.
- 104571 **STDOUT**
- 104572 Strings found shall be written to the standard output, one per line.
- 104573 When the **-t** option is not specified, the format of the output shall be:
- 104574 "%s", <string>
- 104575 With the **-t o** option, the format of the output shall be:
- 104576 "%o %s", <byte offset>, <string>
- 104577 With the **-t x** option, the format of the output shall be:
- 104578 "%x %s", <byte offset>, <string>
- 104579 With the **-t d** option, the format of the output shall be:
- 104580 "%d %s", <byte offset>, <string>
- 104581 **STDERR**
- 104582 The standard error shall be used only for diagnostic messages.
- 104583 **OUTPUT FILES**
- 104584 None.
- 104585 **EXTENDED DESCRIPTION**
- 104586 None.
- 104587 **EXIT STATUS**
- 104588 The following exit values shall be returned:
- 104589 0 Successful completion.
- 104590 >0 An error occurred.
- 104591 **CONSEQUENCES OF ERRORS**
- 104592 Default.
- 104593 **APPLICATION USAGE**
- 104594 By default the data area (as opposed to the text, "bss", or header areas) of a binary executable
- 104595 file is scanned. Implementations document which areas are scanned.
- 104596 Some historical implementations do not require NUL or <newline> terminators for strings to
- 104597 permit those languages that do not use NUL as a string terminator to have their strings written.
- 104598 **EXAMPLES**
- 104599 None.
- 104600 **RATIONALE**
- 104601 Apart from rationalizing the option syntax and slight difficulties with object and executable
- 104602 binary files, *strings* is specified to match historical practice closely. The **-a** and **-n** options were
- 104603 introduced to replace the non-conforming **-** and **-number** options. These options are no longer
- 104604 specified by POSIX.1-200x but may be present in some implementations.
- 104605 The **-o** option historically means different things on different implementations. Some use it to
- 104606 mean "offset in decimal", while others use it as "offset in octal". Instead of trying to decide which
- 104607 way would be least objectionable, the **-t** option was added. It was originally named **-O** to mean

104608 “offset”, but was changed to `-t` to be consistent with *od*.

104609 The ISO C standard function *isprint()* is restricted to a domain of **unsigned char**. This volume of
104610 POSIX.1-200x requires implementations to write strings as defined by the current locale.

104611 FUTURE DIRECTIONS

104612 None.

104613 SEE ALSO

104614 *localedef*, *nm*

104615 XBD Chapter 8 (on page 159), Section 12.2 (on page 201) +

104616 CHANGE HISTORY

104617 First released in Issue 4.

104618 Issue 6

104619 This utility is marked as part of the User Portability Utilities option.

104620 The obsolescent SYNOPSIS is removed.

104621 The normative text is reworded to avoid use of the term “must” for application requirements.

104622 Issue 7

104623 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying the behavior if the first
104624 argument is ‘-’.

104625 The *strings* utility is moved from the User Portability Utilities option to the Base. User
104626 Portability Utilities is now an option for interactive utilities.

104627 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

104628 **NAME**
 104629 strip — remove unnecessary information from strippable files (**DEVELOPMENT**)

104630 **SYNOPSIS**
 104631 SD strip *file...*

104632 **DESCRIPTION**
 104633 XSI A strippable file is defined as a relocatable, object, or executable file. On XSI-conformant
 104634 systems, a strippable file can also be an archive of object or relocatable files.
 104635 The *strip* utility shall remove from strippable files named by the *file* operands any information
 104636 the implementor deems unnecessary for execution of those files. The nature of that information
 104637 is unspecified. The effect of *strip* on object and executable files shall be similar to the use of the
 104638 XSI *-s* option to *c99* or *fort77*. The effect of *strip* on an archive of object files shall be similar to the
 104639 use of the *-s* option to *c99* or *fort77* for each object file in the archive.

104640 **OPTIONS**
 104641 None.

104642 **OPERANDS**
 104643 The following operand shall be supported:
 104644 *file* A pathname referring to a strippable file.

104645 **STDIN**
 104646 Not used.

104647 **INPUT FILES**
 104648 The input files shall be in the form of strippable files successfully produced by any compiler
 104649 XSI defined by this volume of POSIX.1-200x or produced by creating or updating an archive of such
 104650 files using the *ar* utility.

104651 **ENVIRONMENT VARIABLES**
 104652 The following environment variables shall affect the execution of *strip*:

104653 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 104654 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 104655 variables used to determine the values of locale categories.)

104656 *LC_ALL* If set to a non-empty string value, override the values of all the other
 104657 internationalization variables.

104658 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 104659 characters (for example, single-byte as opposed to multi-byte characters in
 104660 arguments).

104661 *LC_MESSAGES*
 104662 Determine the locale that should be used to affect the format and contents of
 104663 diagnostic messages written to standard error.

104664 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

104665 **ASYNCHRONOUS EVENTS**
 104666 Default.

- 104667 **STDOUT**
- 104668 Not used.
- 104669 **STDERR**
- 104670 The standard error shall be used only for diagnostic messages.
- 104671 **OUTPUT FILES**
- 104672 The *strip* utility shall produce strippable files of unspecified format.
- 104673 **EXTENDED DESCRIPTION**
- 104674 None.
- 104675 **EXIT STATUS**
- 104676 The following exit values shall be returned:
- 104677 0 Successful completion.
- 104678 >0 An error occurred.
- 104679 **CONSEQUENCES OF ERRORS**
- 104680 Default.
- 104681 **APPLICATION USAGE**
- 104682 None.
- 104683 **EXAMPLES**
- 104684 None.
- 104685 **RATIONALE**
- 104686 Historically, this utility has been used to remove the symbol table from a strippable file. It was
- 104687 included since it is known that the amount of symbolic information can amount to several
- 104688 megabytes; the ability to remove it in a portable manner was deemed important, especially for
- 104689 smaller systems.
- 104690 The behavior of *strip* on object and executable files is said to be the same as the `-s` option to a
- 104691 compiler. While the end result is essentially the same, it is not required to be identical.
- 104692 XSI-conformant systems support use of *strip* on archive files containing object files or relocatable
- 104693 files.
- 104694 **FUTURE DIRECTIONS**
- 104695 None.
- 104696 **SEE ALSO**
- 104697 *ar*, *c99*, *fort77*
- 104698 XBD [Chapter 8](#) (on page 159)
- 104699 **CHANGE HISTORY**
- 104700 First released in Issue 2.
- 104701 **Issue 6**
- 104702 This utility is marked as part of the Software Development Utilities option.
- 104703 **Issue 7**
- 104704 Austin Group Interpretation 1003.1-2001 #103 is applied.

104705 **NAME**
 104706 `stty` — set the options for a terminal

104707 **SYNOPSIS**
 104708 `stty [-a|-g]`
 104709 `stty operand...`

104710 **DESCRIPTION**
 104711 The `stty` utility shall set or report on terminal I/O characteristics for the device that is its
 104712 standard input. Without options or operands specified, it shall report the settings of certain
 104713 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it
 104714 shall modify the terminal state according to the specified operands. Detailed information about
 104715 the modes listed in the first five groups below are described in XBD [Chapter 11](#) (on page 185).
 104716 Operands in the Combination Modes group (see [Combination Modes](#), on page 3112) are
 104717 implemented using operands in the previous groups. Some combinations of operands are
 104718 mutually-exclusive on some terminal types; the results of using such combinations are
 104719 unspecified.

104720 Typical implementations of this utility require a communications line configured to use the
 104721 **termios** interface defined in the System Interfaces volume of POSIX.1-200x. On systems where
 104722 none of these lines are available, and on lines not currently configured to support the **termios**
 104723 interface, some of the operands need not affect terminal characteristics.

104724 **OPTIONS**
 104725 The `stty` utility shall conform to XBD [Section 12.2](#) (on page 201).

104726 The following options shall be supported:

- 104727 **-a** Write to standard output all the current settings for the terminal.
- 104728 **-g** Write to standard output all the current settings in an unspecified form that can be
 104729 used as arguments to another invocation of the `stty` utility on the same system. The
 104730 form used shall not contain any characters that would require quoting to avoid
 104731 word expansion by the shell; see [Section 2.6](#) (on page 2253).

104732 **OPERANDS**
 104733 The following operands shall be supported to set the terminal characteristics.

104734 **Control Modes**

104735 **parenb** (**-parenb**) Enable (disable) parity generation and detection. This shall have the effect of
 104736 setting (not setting) PARENB in the **termios** `c_flag` field, as defined in XBD
 104737 [Chapter 11](#) (on page 185).

104738 **parodd** (**-parodd**)
 104739 Select odd (even) parity. This shall have the effect of setting (not setting)
 104740 PARODD in the **termios** `c_flag` field, as defined in XBD [Chapter 11](#) (on page
 104741 185).

104742 **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,
 104743 CS7, and CS8, respectively, in the **termios** `c_flag` field, as defined in XBD
 104744 [Chapter 11](#) (on page 185).

104745 *number* Set terminal baud rate to the number given, if possible. If the baud rate is set
 104746 to zero, the modem control lines shall no longer be asserted. This shall have
 104747 the effect of setting the input and output **termios** baud rate values as defined
 104748 in XBD [Chapter 11](#) (on page 185).

104749	ispeed <i>number</i>	Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate shall be specified by the value of the output baud rate. This shall have the effect of setting the input termios baud rate values as defined in XBD Chapter 11 (on page 185).
104750		
104751		
104752		
104753	ospeed <i>number</i>	Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the output termios baud rate values as defined in XBD Chapter 11 (on page 185).
104754		
104755		
104756		
104757	hupcl (-hupcl)	Stop asserting modem control lines (do not stop asserting modem control lines) on last close. This shall have the effect of setting (not setting) HUPCL in the termios <i>c_cflag</i> field, as defined in XBD Chapter 11 (on page 185).
104758		
104759		
104760	hup (-hup)	Equivalent to hupcl (-hupcl).
104761	cstopb (-cstopb)	Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the termios <i>c_cflag</i> field, as defined in XBD Chapter 11 (on page 185).
104762		
104763		
104764	cread (-cread)	Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the termios <i>c_cflag</i> field, as defined in XBD Chapter 11 (on page 185).
104765		
104766		
104767	clocal (-clocal)	Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the termios <i>c_cflag</i> field, as defined in XBD Chapter 11 (on page 185).
104768		
104769		
104770	It is unspecified whether <i>stty</i> shall report an error if an attempt to set a Control Mode fails.	
104771	Input Modes	
104772	ignbrk (-ignbrk)	Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104773		
104774		
104775	brkint (-brkint)	Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104776		
104777		
104778	ignpar (-ignpar)	Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104779		
104780		
104781	parmrk (-parmrk)	Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104782		
104783		
104784		
104785	inpck (-inpck)	Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104786		
104787		
104788	istrip (-istrip)	Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104789		
104790		
104791	inlcr (-inlcr)	Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104792		
104793		

104794		igncr (-igncr)	Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104795			
104796			
104797		icrnl (-icrnl)	Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104798			
104799			
104800		ixon (-ixon)	Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104801			
104802			
104803			
104804	XSI	ixany (-ixany)	Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104805			
104806			
104807		ixoff (-ixoff)	Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the termios <i>c_iflag</i> field, as defined in XBD Chapter 11 (on page 185).
104808			
104809			
104810			
104811		Output Modes	
104812		opost (-opost)	Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104813			
104814			
104815	XSI	ocrnl (-ocrnl)	Map (do not map) CR to NL on output This shall have the effect of setting (not setting) OCRNL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104816			
104817			
104818	XSI	onocr (-onocr)	Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104819			
104820			
104821	XSI	onlret (-onlret)	The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104822			
104823			
104824	XSI	ofill (-ofill)	Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104825			
104826			
104827	XSI	ofdel (-ofdel)	Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104828			
104829			
104830	XSI	cr0 cr1 cr2 cr3	Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104831			
104832			
104833	XSI	nl0 nl1	Select the style of delay for NL. This shall have the effect of setting NLDLY to NL0 or NL1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104834			
104835			
104836	XSI	tab0 tab1 tab2 tab3	Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185). Note that TAB3 has the effect of expanding <tab>s to <space>s.
104837			
104838			
104839			
104840			

104841	XSI	tabs (-tabs)	Synonym for tab0 (tab3).
104842	XSI	bs0 bs1	Select the style of delay for backspaces. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104843			
104844			
104845	XSI	ff0 ff1	Select the style of delay for form-feeds. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104846			
104847			
104848	XSI	vt0 vt1	Select the style of delay for vertical-tabs. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the termios <i>c_oflag</i> field, as defined in XBD Chapter 11 (on page 185).
104849			
104850			
104851		Local Modes	
104852		isig (-isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104853			
104854			
104855			
104856		icanon (-icanon)	Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104857			
104858			
104859		iexten (-iexten)	Enable (disable) any implementation-defined special control characters not currently controlled by icanon , isig , ixon , or ixoff . This shall have the effect of setting (not setting) IEXTEN in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104860			
104861			
104862			
104863		echo (-echo)	Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104864			
104865			
104866		echoe (-echoe)	The ERASE character visually erases (does not erase) the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104867			
104868			
104869			
104870		echok (-echok)	Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104871			
104872			
104873		echonl (-echonl)	Echo (do not echo) NL, even if echo is disabled. This shall have the effect of setting (not setting) ECHONL in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104874			
104875			
104876		noflsh (-noflsh)	Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of setting (not setting) NOFLSH in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104877			
104878			
104879		tostop (-tostop)	Send SIGTTOU for background output. This shall have the effect of setting (not setting) TOSTOP in the termios <i>c_lflag</i> field, as defined in XBD Chapter 11 (on page 185).
104880			
104881			

104882

Special Control Character Assignments

104883

<control>-character string

104884

Set *<control>-character* to *string*. If *<control>-character* is one of the character sequences in the first column of the following table, the corresponding XBD Chapter 11 (on page 185) control character from the second column shall be recognized. This has the effect of setting the corresponding element of the **termios** *c_cc* array (see XBD Chapter 13 (on page 205), **<termios.h>**).

104885

104886

104887

104888

104889

Table 4-19 Control Character Names in *stty*

104890

Control Character	c_cc Subscript	Description
eof	VEOF	EOF character
eol	VEOL	EOL character
erase	VERASE	ERASE character
intr	VINTR	INTR character
kill	VKILL	KILL character
quit	VQUIT	QUIT character
susp	VSUSP	SUSP character
start	VSTART	START character
stop	VSTOP	STOP character

104891

104892

104893

104894

104895

104896

104897

104898

104899

104900

If *string* is a single character, the control character shall be set to that character. If *string* is the two-character sequence "^-" or the string *undef*, the control character shall be set to `_POSIX_VDISABLE`, if it is in effect for the device; if `_POSIX_VDISABLE` is not in effect for the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character sequence beginning with circumflex ('^'), and the second character is one of those listed in the "^c" column of the following table, the control character shall be set to the corresponding character value in the Value column of the table.

104901

104902

104903

104904

104905

104906

104907

Table 4-20 Circumflex Control Characters in *stty*

104908

^c	Value	^c	Value	^c	Value
a, A	<SOH>	l, L	<FF>	w, W	<ETB>
b, B	<STX>	m, M	<CR>	x, X	<CAN>
c, C	<ETX>	n, N	<SO>	y, Y	
d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
e, E	<ENQ>	p, P	<DLE>	[<ESC>
f, F	<ACK>	q, Q	<DC1>	\	<FS>
g, G	<BEL>	r, R	<DC2>]	<GS>
h, H	<BS>	s, S	<DC3>	^	<RS>
i, I	<HT>	t, T	<DC4>	_	<US>
j, J	<LF>	u, U	<NAK>	?	
k, K	<VT>	v, V	<SYN>		

104909

104910

104911

104912

104913

104914

104915

104916

104917

104918

104919

104920

min number

104921

Set the value of MIN to *number*. MIN is used in non-canonical mode input processing (**icanon**).

104922

104923

time number

104924

Set the value of TIME to *number*. TIME is used in non-canonical mode input processing (**icanon**).

104925

- 104926 **Combination Modes**
- 104927 *saved settings*
- 104928 Set the current terminal characteristics to the saved settings produced by the **-g** option.
- 104929 **evenp** or **parity**
- 104930 Enable **parenb** and **cs7**; disable **parodd**.
- 104931 **oddp**
- 104932 Enable **parenb**, **cs7**, and **parodd**.
- 104933 **-parity**, **-evenp**, or **-oddp**
- 104934 Disable **parenb**, and set **cs8**.
- 104935 XSI **raw** (**-raw** or **cooked**)
- 104936 Enable (disable) raw input and output. Raw mode shall be equivalent to setting:
- 104937 `stty cs8 erase ^- kill ^- intr ^- \`
- 104938 `quit ^- eof ^- eol ^- -post -inpck`
- 104939 **nl** (**-nl**)
- 104940 Disable (enable) **icrnl**. In addition, **-nl** unsets **inlcr** and **igncr**.
- 104941 **ek** Reset ERASE and KILL characters back to system defaults.
- 104942 **sane**
- 104943 Reset all modes to some reasonable, unspecified, values.
- 104944 **STDIN**
- 104945 Although no input is read from standard input, standard input shall be used to get the current
- 104946 terminal I/O characteristics and to set new terminal I/O characteristics.
- 104947 **INPUT FILES**
- 104948 None.
- 104949 **ENVIRONMENT VARIABLES**
- 104950 The following environment variables shall affect the execution of *stty*:
- 104951 **LANG** Provide a default value for the internationalization variables that are unset or null. |
- 104952 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |
- 104953 variables used to determine the values of locale categories.)
- 104954 **LC_ALL** If set to a non-empty string value, override the values of all the other
- 104955 internationalization variables.
- 104956 **LC_CTYPE** This variable determines the locale for the interpretation of sequences of bytes of
- 104957 text data as characters (for example, single-byte as opposed to multi-byte
- 104958 characters in arguments) and which characters are in the class **print**.
- 104959 **LC_MESSAGES**
- 104960 Determine the locale that should be used to affect the format and contents of
- 104961 diagnostic messages written to standard error.
- 104962 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.
- 104963 **ASYNCHRONOUS EVENTS**
- 104964 Default.

104965 **STDOUT**

104966 If operands are specified, no output shall be produced.

104967 If the **-g** option is specified, *stty* shall write to standard output the current settings in a form that
104968 can be used as arguments to another instance of *stty* on the same system.

104969 If the **-a** option is specified, all of the information as described in the OPERANDS section shall
104970 be written to standard output. Unless otherwise specified, this information shall be written as
104971 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified
104972 number of tokens per line. Additional information may be written.

104973 If no options or operands are specified, an unspecified subset of the information written for the
104974 **-a** option shall be written.

104975 If speed information is written as part of the default output, or if the **-a** option is specified and if
104976 the terminal input speed and output speed are the same, the speed information shall be written
104977 as follows:

104978 "speed %d baud;", <speed>

104979 Otherwise, speeds shall be written as:

104980 "ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>

104981 In locales other than the POSIX locale, the word **baud** may be changed to something more
104982 appropriate in those locales.

104983 If control characters are written as part of the default output, or if the **-a** option is specified,
104984 control characters shall be written as:

104985 "%s = %s;", <control-character name>, <value>

104986 where <value> is either the character, or some visual representation of the character if it is non-
104987 printable, or the string *undef* if the character is disabled.

104988 **STDERR**

104989 The standard error shall be used only for diagnostic messages.

104990 **OUTPUT FILES**

104991 None.

104992 **EXTENDED DESCRIPTION**

104993 None.

104994 **EXIT STATUS**

104995 The following exit values shall be returned:

104996 0 The terminal options were read or set successfully.

104997 >0 An error occurred.

104998 **CONSEQUENCES OF ERRORS**

104999 Default.

105000 **APPLICATION USAGE**

105001 The **-g** flag is designed to facilitate the saving and restoring of terminal state from the shell level.
105002 For example, a program may:

```
105003 saveterm="$(stty -g)"           # save terminal state
105004 stty (new settings)           # set new state
105005 ...                           # ...
105006 stty $saveterm                # restore terminal state
```

105007 Since the format is unspecified, the saved value is not portable across systems.

105008 Since the `-a` format is so loosely specified, scripts that save and restore terminal settings should
 105009 use the `-g` option.

105010 EXAMPLES

105011 None.

105012 RATIONALE

105013 The original *stty* description was taken directly from System V and reflected the System V
 105014 terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

105015 Output modes are specified only for XSI-conformant systems. All implementations are expected
 105016 to provide *stty* operands corresponding to all of the output modes they support.

105017 The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the
 105018 preferred ERASE and KILL characters. As an application programming utility, *stty* can be used
 105019 within shell scripts to alter the terminal settings for the duration of the script.

105020 The **termios** section states that individual disabling of control characters is possible through the
 105021 option `_POSIX_VDISABLE`. If enabled, two conventions currently exist for specifying this:
 105022 System V uses `"^-"`, and BSD uses *undef*. Both are accepted by *stty* in this volume of
 105023 POSIX.1-200x. The other BSD convention of using the letter 'u' was rejected because it conflicts
 105024 with the actual letter 'u', which is an acceptable value for a control character.

105025 Early proposals did not specify the mapping of `"^c"` to control characters because the control
 105026 characters were not specified in the POSIX locale character set description file requirements. The
 105027 control character set is now specified in XBD [Chapter 3](#) (on page 33), so the historical mapping is
 105028 specified. Note that although the mapping corresponds to control-character key assignments on
 105029 many terminals that use the ISO/IEC 646:1991 standard (or ASCII) character encodings, the
 105030 mapping specified here is to the control characters, not their keyboard encodings.

105031 Since **termios** supports separate speeds for input and output, two new options were added to
 105032 specify each distinctly.

105033 Some historical implementations use standard input to get and set terminal characteristics;
 105034 others use standard output. Since input from a login TTY is usually restricted to the owner while
 105035 output to a TTY is frequently open to anyone, using standard input provides fewer chances of
 105036 accidentally (or maliciously) altering the terminal settings of other users. Using standard input
 105037 also allows *stty -a* and *stty -g* output to be redirected for later use. Therefore, usage of standard
 105038 input is required by this volume of POSIX.1-200x.

105039 FUTURE DIRECTIONS

105040 None.

105041 SEE ALSO

105042 [Chapter 2](#) (on page 2245)

105043 XBD [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185), [Section 12.2](#) (on page 201), [<termios.h>](#)

105044 CHANGE HISTORY

105045 First released in Issue 2.

105046 Issue 5

105047 The description of **tabs** is clarified.

105048 The FUTURE DIRECTIONS section is added.

105049 Issue 6

105050 The LEGACY items **iucl(-iucl)**, **xcase**, **olcuc(-olcuc)**, **lcase(-lcase)**, and **LCASE(-LCASE)** are
 105051 removed.

105052 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/37 is applied, applying IEEE PASC
 105053 Interpretation 1003.2 #133, fixing an error in the OPERANDS section for the Combination Modes

105054

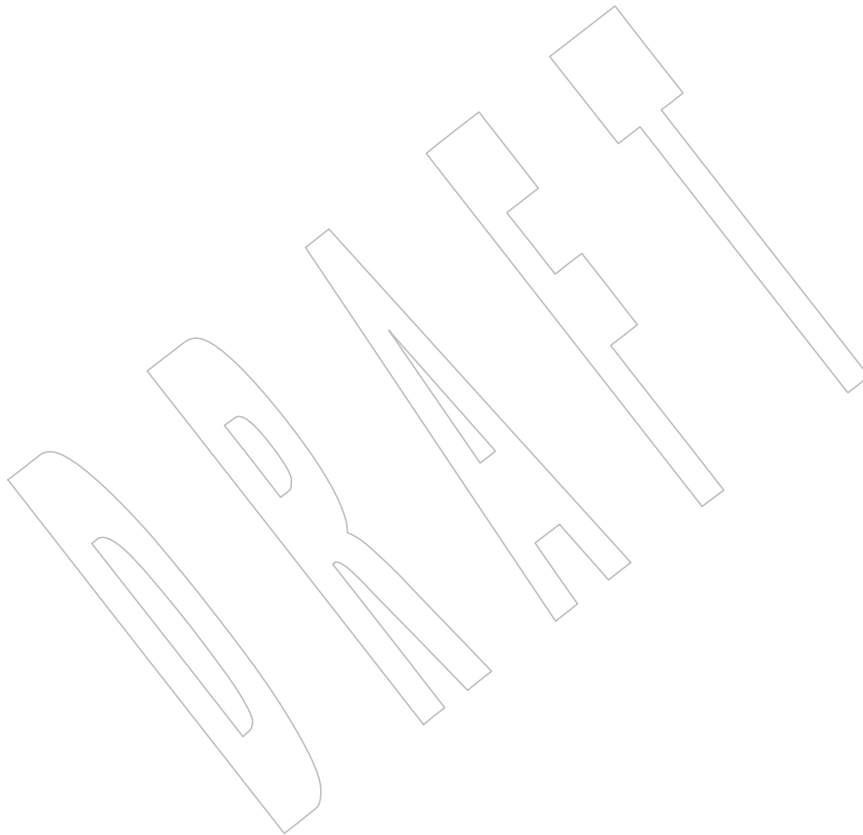
nl(-nl).

105055

Issue 7

105056

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



105057 **NAME**

105058 tabs — set terminal tabs

105059 **SYNOPSIS**

105060 XSI tabs [-n|-a|-a2|-c|-c2|-c3|-f|-p|-s|-u] [-T type]

105061 tabs [-T type] n[[sep[+]n]...]

105062 **DESCRIPTION**

105063 The *tabs* utility shall display a series of characters that first clears the hardware terminal tab
 105064 settings and then initializes the tab stops at the specified positions and optionally adjusts the
 105065 margin.

105066 The phrase “tab-stop position *N*” shall be taken to mean that, from the start of a line of output,
 105067 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position
 105068 on that line. The maximum number of tab stops allowed is terminal-dependent.

105069 It need not be possible to implement *tabs* on certain terminals. If the terminal type obtained from
 105070 the *TERM* environment variable or *-T* option represents such a terminal, an appropriate
 105071 diagnostic message shall be written to standard error and *tabs* shall exit with a status greater
 105072 than zero.

105073 **OPTIONS**

105074 XSI The *tabs* utility shall conform to XBD Section 12.2 (on page 201), except for various extensions:
 105075 the options *-a2*, *-c2*, and *-c3* are multi-character.

105076 The following options shall be supported:

105077 *-n* Specify repetitive tab stops separated by a uniform number of column positions, *n*,
 105078 where *n* is a single-digit decimal number. The default usage of *tabs* with no
 105079 arguments shall be equivalent to *tabs -8*. When *-0* is used, the tab stops shall be
 105080 cleared and no new ones set.

105081 XSI *-a* 1,10,16,36,72

105082 Assembler, applicable to some mainframes.

105083 XSI *-a2* 1,10,16,40,72

105084 Assembler, applicable to some mainframes.

105085 XSI *-c* 1,8,12,16,20,55

105086 COBOL, normal format.

105087 XSI *-c2* 1,6,10,14,49

105088 COBOL, compact format (columns 1 to 6 omitted).

105089 XSI *-c3* 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67105090 COBOL compact format (columns 1 to 6 omitted), with more tabs than *-c2*.105091 XSI *-f* 1,7,11,15,19,23

105092 FORTRAN

105093 XSI *-p* 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61

105094 PL/1

105095 XSI *-s* 1,10,55

105096 SNOBOL

105097 XSI *-u* 1,12,20,44

105098 Assembler, applicable to some mainframes.

105099 -T *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable
105100 is unset or null, an unspecified default terminal type shall be used. The setting of
105101 *type* shall take precedence over the value in *TERM*.

105102 OPERANDS

105103 The following operand shall be supported:

105104 *n*[[*sep*[+]*n*]...] A single command line argument that consists of one or more tab-stop values (*n*) |
105105 separated by a separator character (*sep*) which is either a comma or a <blank> |
105106 character. The application shall ensure that the tab-stop values are positive decimal |
105107 integers in strictly ascending order. If any tab-stop value (except the first one) is |
105108 preceded by a plus sign, it is taken as an increment to be added to the previous |
105109 value. For example, the tab lists 1,10,20,30 and "1 10 +10 +10" are considered |
105110 to be identical.

105111 STDIN

105112 Not used.

105113 INPUT FILES

105114 None.

105115 ENVIRONMENT VARIABLES

105116 The following environment variables shall affect the execution of *tabs*:

105117 *LANG* Provide a default value for the internationalization variables that are unset or null. |
105118 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
105119 variables used to determine the values of locale categories.)

105120 *LC_ALL* If set to a non-empty string value, override the values of all the other |
105121 internationalization variables.

105122 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as |
105123 characters (for example, single-byte as opposed to multi-byte characters in |
105124 arguments).

105125 *LC_MESSAGES*

105126 Determine the locale that should be used to affect the format and contents of |
105127 diagnostic messages written to standard error.

105128 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

105129 *TERM* Determine the terminal type. If this variable is unset or null, and if the -T option is |
105130 not specified, an unspecified default terminal type shall be used.

105131 ASYNCHRONOUS EVENTS

105132 Default.

105133 STDOUT

105134 If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be |
105135 written to standard output in an unspecified format. If standard output is not a terminal, |
105136 undefined results occur.

105137 STDERR

105138 The standard error shall be used only for diagnostic messages.

105139 OUTPUT FILES

105140 None.

105141 **EXTENDED DESCRIPTION**

105142 None.

105143 **EXIT STATUS**

105144 The following exit values shall be returned:

105145 0 Successful completion.

105146 >0 An error occurred.

105147 **CONSEQUENCES OF ERRORS**

105148 Default.

105149 **APPLICATION USAGE**105150 This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

105151 This utility is not recommended for application use.

105152 Some integrated display units might not have escape sequences to set tab stops, but may be set
 105153 by internal system calls. On these terminals, *tabs* works if standard output is directed to the
 105154 terminal; if output is directed to another file, however, *tabs* fails.

105155 **EXAMPLES**

105156 None.

105157 **RATIONALE**

105158 Consideration was given to having the *tput* utility handle all of the functions described in *tabs*.
 105159 However, the separate *tabs* utility was retained because it seems more intuitive to use a
 105160 command named *tabs* than *tput* with a new option. The *tput* utility does not support setting or
 105161 clearing tabs, and no known historical version of *tabs* supports the capability of setting arbitrary
 105162 tab stops.

105163 The System V *tabs* interface is very complex; the version in this volume of POSIX.1-200x has a
 105164 reduced feature list, but many of the features omitted were restored as part of the XSI option
 105165 even though the supported languages and coding styles are primarily historical.

105166 There was considerable sentiment for specifying only a means of resetting the tabs back to a
 105167 known state—presumably the “standard” of tabs every eight positions. The following features
 105168 were omitted:

- 105169 • Setting tab stops via the first line in a file, using *--file*. Since even the SVID has no
 105170 complete explanation of this feature, it is doubtful that it is in widespread use.

105171 In an early proposal, a *-t tablist* option was added for consistency with *expand*; this was later
 105172 removed when inconsistencies with the historical list of tabs were identified.

105173 Consideration was given to adding a *-p* option that would output the current tab settings so
 105174 that they could be saved and then later restored. This was not accepted because querying the tab
 105175 stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be
 105176 supported on a wide range of terminals.

105177 **FUTURE DIRECTIONS**

105178 None.

105179 **SEE ALSO**105180 *expand*, *stty*, *tput*, *unexpand*

105181 XBD Chapter 8 (on page 159), Section 12.2 (on page 201)

+

105182

CHANGE HISTORY

105183

First released in Issue 2.

105184

Issue 6

105185

This utility is marked as part of the User Portability Utilities option.

105186

The normative text is reworded to avoid use of the term “must” for application requirements.

105187

Issue 7

105188

The *tabs* utility is removed from the User Portability Utilities option. User Portability Utilities is now an option for interactive utilities.

105189

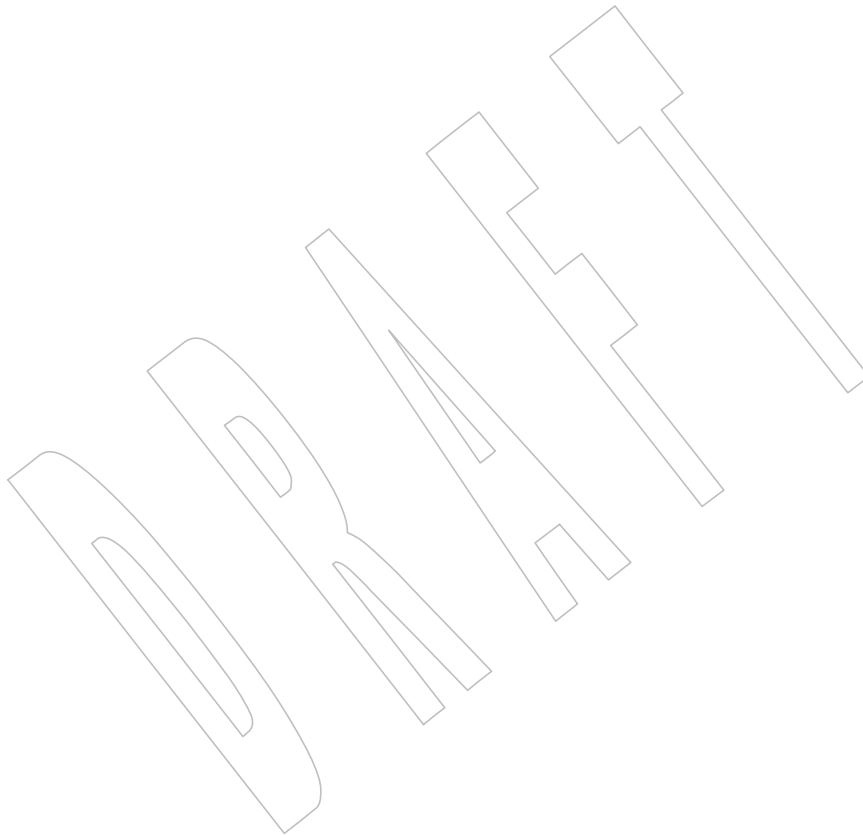
105190

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105191

The SYNOPSIS and OPERANDS sections are updated.

+



105192 **NAME**
 105193 tail — copy the last part of a file

105194 **SYNOPSIS**
 105195 tail [-f] [-c *number*|-n *number*] [*file*]

105196 **DESCRIPTION**
 105197 The *tail* utility shall copy its input file to the standard output beginning at a designated place.
 105198 Copying shall begin at the point in the file indicated by the *-c number* or *-n number* options. The
 105199 option-argument *number* shall be counted in units of lines or bytes, according to the options *-n*
 105200 and *-c*. Both line and byte counts start from 1.

105201 Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in
 105202 length. Such a buffer, if any, shall be no smaller than {LINE_MAX}*10 bytes.

105203 **OPTIONS**
 105204 The *tail* utility shall conform to XBD Section 12.2 (on page 201), except that '+' may be
 105205 recognized as an option delimiter as well as '-'. |

105206 The following options shall be supported:

105207 *-c number* The application shall ensure that the *number* option-argument is a decimal integer
 105208 whose sign affects the location in the file, measured in bytes, to begin the copying:

Sign	Copying Starts
+	Relative to the beginning of the file.
-	Relative to the end of the file.
<i>none</i>	Relative to the end of the file.

105213 The application shall ensure that if the sign of the *number* option-argument is '+',
 105214 the *number* option-argument is a positive decimal integer.

105215 The origin for counting shall be 1; that is, *-c +1* represents the first byte of the file,
 105216 *-c -1* the last.

105217 *-f* If the input file is a regular file or if the *file* operand specifies a FIFO, do not
 105218 terminate after the last line of the input file has been copied, but read and copy
 105219 further bytes from the input file when they become available. If no *file* operand is
 105220 specified and standard input is a pipe or FIFO, the *-f* option shall be ignored. If the
 105221 input file is not a FIFO, pipe, or regular file, it is unspecified whether or not the *-f*
 105222 option shall be ignored.

105223 *-n number* This option shall be equivalent to *-c number*, except the starting location in the file
 105224 shall be measured in lines instead of bytes. The origin for counting shall be 1; that
 105225 is, *-n +1* represents the first line of the file, *-n -1* the last.

105226 If neither *-c* nor *-n* is specified, *-n 10* shall be assumed.

105227 **OPERANDS**
 105228 The following operand shall be supported:

105229 *file* A pathname of an input file. If no *file* operands are specified, the standard input
 105230 shall be used.

105231 **STDIN**

105232 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
105233 section.

105234 **INPUT FILES**

105235 If the `-c` option is specified, the input file can contain arbitrary data; otherwise, the input file
105236 shall be a text file.

105237 **ENVIRONMENT VARIABLES**

105238 The following environment variables shall affect the execution of *tail*:

105239 **LANG** Provide a default value for the internationalization variables that are unset or null. |
105240 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
105241 variables used to determine the values of locale categories.)

105242 **LC_ALL** If set to a non-empty string value, override the values of all the other
105243 internationalization variables.

105244 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
105245 characters (for example, single-byte as opposed to multi-byte characters in
105246 arguments and input files).

105247 **LC_MESSAGES**

105248 Determine the locale that should be used to affect the format and contents of
105249 diagnostic messages written to standard error.

105250 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.

105251 **ASYNCHRONOUS EVENTS**

105252 Default.

105253 **STDOUT**

105254 The designated portion of the input file shall be written to standard output.

105255 **STDERR**

105256 The standard error shall be used only for diagnostic messages.

105257 **OUTPUT FILES**

105258 None.

105259 **EXTENDED DESCRIPTION**

105260 None.

105261 **EXIT STATUS**

105262 The following exit values shall be returned:

105263 0 Successful completion.

105264 >0 An error occurred.

105265 **CONSEQUENCES OF ERRORS**

105266 Default.

105267 **APPLICATION USAGE**

105268 The `-c` option should be used with caution when the input is a text file containing multi-byte
105269 characters; it may produce output that does not start on a character boundary.

105270 Although the input file to *tail* can be any type, the results might not be what would be expected
105271 on some character special device files or on file types not described by the System Interfaces
105272 volume of POSIX.1-200x. Since this volume of POSIX.1-200x does not specify the block size used
105273 when doing input, *tail* need not read all of the data from devices that only perform block
105274 transfers.

EXAMPLES

The `-f` option can be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail -f -c 15 fred
```

prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between the time *tail* is initiated and killed.

RATIONALE

This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The historical `-b` option was omitted because of the general non-portability of block-sized units of text. The `-c` option historically meant “characters”, but this volume of POSIX.1-200x indicates that it means “bytes”. This was selected to allow reasonable implementations when multi-byte characters are possible; it was not named `-b` to avoid confusion with the historical `-b`.

The origin of counting both lines and bytes is 1, matching all widespread historical implementations. Hence *tail -n +0* is not conforming usage because it attempts to output line zero; but note that *tail -n 0* does conform, and outputs nothing.

Earlier versions of this standard allowed the following forms in the SYNOPSIS:

```
tail -[number][b|c|l][f] [file]
tail +[number][b|c|l][f] [file]
```

These forms are no longer specified by POSIX.1-200x, but may be present in some implementations.

The restriction on the internal buffer is a compromise between the historical System V implementation of 4096 bytes and the BSD 32768 bytes.

The `-f` option has been implemented as a loop that sleeps for 1 second and copies any bytes that are available. This is sufficient, but if more efficient methods of determining when new data are available are developed, implementations are encouraged to use them.

Historical documentation indicates that *tail* ignores the `-f` option if the input file is a pipe (pipe and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System V-based systems, this was true when input was taken from standard input, but it did not ignore the `-f` flag if a FIFO was named as the *file* operand. Since the `-f` option is not useful on pipes and all historical implementations ignore `-f` if no *file* operand is specified and standard input is a pipe, this volume of POSIX.1-200x requires this behavior. However, since the `-f` option is useful on a FIFO, this volume of POSIX.1-200x also requires that if a FIFO is named, the `-f` option shall not be ignored. Earlier versions of this standard did not state any requirement for the case where no *file* operand is specified and standard input is a FIFO. The standard has been updated to reflect current practice which is to treat this case the same as a pipe on standard input. Although historical behavior does not ignore the `-f` option for other file types, this is unspecified so that implementations are allowed to ignore the `-f` option if it is known that the file cannot be extended.

This was changed to the current form based on comments noting that `-c` was almost never used without specifying a number and that there was no need to specify `-l` if `-n number` was given.

105318 **FUTURE DIRECTIONS**

105319 None.

105320 **SEE ALSO**105321 *head*105322 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +105323 **CHANGE HISTORY**

105324 First released in Issue 2.

105325 **Issue 6**

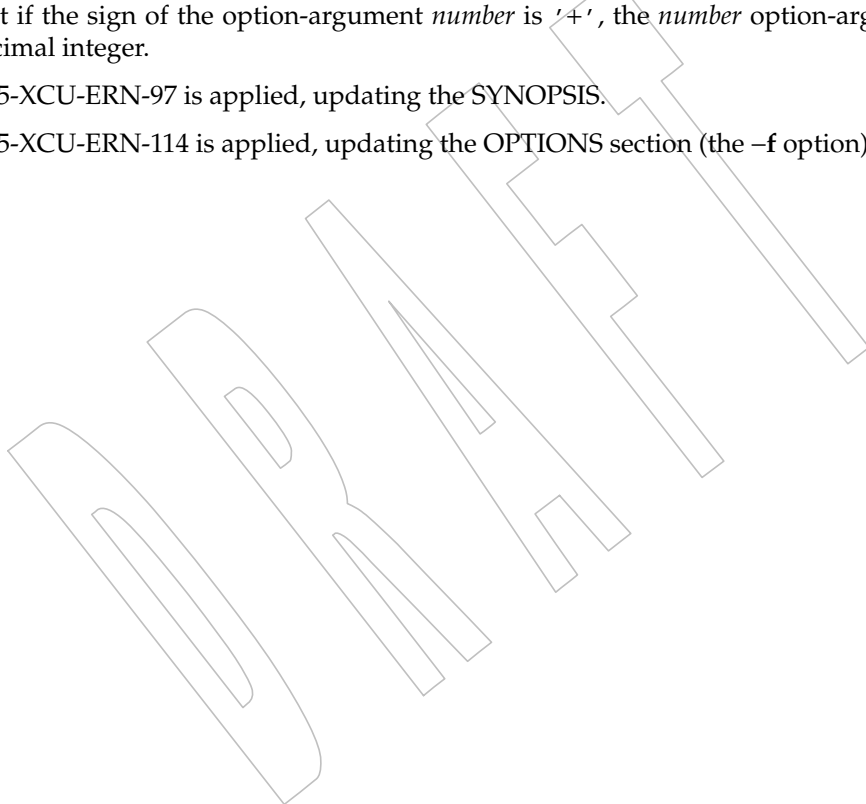
105326 The obsolescent SYNOPSIS lines and associated text are removed.

105327 The normative text is reworded to avoid use of the term “must” for application requirements.

105328 **Issue 7**105329 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that ‘+’ may be recognized
105330 as an option delimiter in the OPTIONS section.105331 Austin Group Interpretation 1003.1-2001 #100 is applied, adding the requirement on applications
105332 that if the sign of the option-argument *number* is ‘+’, the *number* option-argument is a positive
105333 decimal integer.

105334 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

105335 SD5-XCU-ERN-114 is applied, updating the OPTIONS section (the -f option).



105336 **NAME**

105337 talk — talk to another user

105338 **SYNOPSIS**105339 UP `talk address [terminal]`105340 **DESCRIPTION**105341 The *talk* utility is a two-way, screen-oriented communication program.105342 When first invoked, *talk* shall send a message similar to:

105343 Message from <unspecified string>
 105344 talk: connection requested by *your_address*
 105345 talk: respond with: talk *your_address*

105346 to the specified *address*. At this point, the recipient of the message can reply by typing:105347 `talk your_address`

105348 Once communication is established, the two parties can type simultaneously, with their output
 105349 displayed in separate regions of the screen. Characters shall be processed as follows:

- 105350 • Typing the alert character shall alert the recipient's terminal.
- 105351 • Typing <control>-L shall cause the sender's screen regions to be refreshed.
- 105352 • Typing the erase and kill characters shall affect the sender's terminal in the manner
 105353 described by the **termios** interface in XBD [Chapter 11](#) (on page 185).
- 105354 • Typing the interrupt or end-of-file characters shall terminate the local *talk* utility. Once the
 105355 *talk* session has been terminated on one side, the other side of the *talk* session shall be
 105356 notified that the *talk* session has been terminated and shall be able to do nothing except
 105357 exit.
- 105358 • Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those
 105359 characters to be sent to the recipient's terminal.
- 105360 • When and only when the *stty iexten* local mode is enabled, the existence and processing of
 105361 additional special control characters and multi-byte or single-byte functions shall be
 105362 implementation-defined.
- 105363 • Typing other non-printable characters shall cause implementation-defined sequences of
 105364 printable characters to be sent to the recipient's terminal.

105365 Permission to be a recipient of a *talk* message can be denied or granted by use of the *mesg* utility.
 105366 However, a user's privilege may further constrain the domain of accessibility of other users'
 105367 terminals. The *talk* utility shall fail when the user lacks the appropriate privileges to perform the
 105368 requested action.

105369 Certain block-mode terminals do not have all the capabilities necessary to support the
 105370 simultaneous exchange of messages required for *talk*. When this type of exchange cannot be
 105371 supported on such terminals, the implementation may support an exchange with reduced levels
 105372 of simultaneous interaction or it may report an error describing the terminal-related deficiency.

105373 **OPTIONS**

105374 None.

105375 **OPERANDS**

105376 The following operands shall be supported:

105377 *address* The recipient of the *talk* session. One form of *address* is the *<user name>*, as returned
 105378 by the *who* utility. Other address formats and how they are handled are
 105379 unspecified.

105380 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to
 105381 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message
 105382 shall be displayed on one or more accessible terminals in use by the recipient. The
 105383 format of *terminal* shall be the same as that returned by the *who* utility.

105384 **STDIN**

105385 Characters read from standard input shall be copied to the recipient's terminal in an unspecified
 105386 manner. If standard input is not a terminal, *talk* shall write a diagnostic message and exit with a
 105387 non-zero status.

105388 **INPUT FILES**

105389 None.

105390 **ENVIRONMENT VARIABLES**105391 The following environment variables shall affect the execution of *talk*:

105392 *LANG* Provide a default value for the internationalization variables that are unset or null.
 105393 (See XBD Section 8.2 (on page 160) for the precedence of internationalization
 105394 variables used to determine the values of locale categories.)

105395 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105396 internationalization variables.

105397 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105398 characters (for example, single-byte as opposed to multi-byte characters in
 105399 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
 105400 equivalent to the sender's, the results are undefined.

105401 *LC_MESSAGES*

105402 Determine the locale that should be used to affect the format and contents of
 105403 diagnostic messages written to standard error and informative messages written to
 105404 standard output.

105405 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

105406 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,
 105407 an unspecified default terminal type shall be used.

105408 **ASYNCHRONOUS EVENTS**

105409 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero
 105410 status. It shall take the standard action for all other signals.

105411 **STDOUT**

105412 If standard output is a terminal, characters copied from the recipient's standard input may be
 105413 written to standard output. Standard output also may be used for diagnostic messages. If
 105414 standard output is not a terminal, *talk* shall exit with a non-zero status.

105415 **STDERR**

105416 None.

105417 **OUTPUT FILES**

105418 None.

105419 **EXTENDED DESCRIPTION**

105420 None.

105421 **EXIT STATUS**

105422 The following exit values shall be returned:

105423 0 Successful completion.

105424 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.105425 **CONSEQUENCES OF ERRORS**

105426 Default.

105427 **APPLICATION USAGE**

105428 Because the handling of non-printable, non-`<space>`s is tied to the *stty* description of **iexten**,
 105429 implementation extensions within the terminal driver can be accessed. For example, some
 105430 implementations provide line editing functions with certain control character sequences.

105431 **EXAMPLES**

105432 None.

105433 **RATIONALE**

105434 The *write* utility was included in this volume of POSIX.1-200x since it can be implemented on all
 105435 terminal types. The *talk* utility, which cannot be implemented on certain terminals, was
 105436 considered to be a “better” communications interface. Both of these programs are in widespread
 105437 use on historical implementations. Therefore, both utilities have been specified.

105438 All references to networking abilities (*talking* to a user on another system) were removed as
 105439 being outside the scope of this volume of POSIX.1-200x.

105440 Historical BSD and System V versions of *talk* terminate both of the conversations when either
 105441 user breaks out of the session. This can lead to adverse consequences if a user unwittingly
 105442 continues to enter text that is interpreted by the shell when the other terminates the session.
 105443 Therefore, the version of *talk* specified by this volume of POSIX.1-200x requires both users to
 105444 terminate their end of the session explicitly.

105445 Only messages sent to the terminal of the invoking user can be internationalized in any way:

- 105446 • The original “Message from *<unspecified string>* ...” message sent to the terminal of the
 105447 recipient cannot be internationalized because the environment of the recipient is as yet
 105448 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.
- 105449 • Subsequent communication between the two parties cannot be internationalized because
 105450 the two parties may specify different languages in their environment (and non-portable
 105451 characters cannot be mapped from one language to another).
- 105452 • Neither party can be required to communicate in a language other than C and/or the one
 105453 specified by their environment because unavailable terminal hardware support (for
 105454 example, fonts) may be required.

105455 The text in the STDOUT section reflects the usage of the verb “display” in this section; some *talk*
 105456 implementations actually use standard output to write to the terminal, but this volume of
 105457 POSIX.1-200x does not require that to be the case.

105458 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
 105459 require that they all use or accept the same format.

105460 The handling of non-printable characters is partially implementation-defined because the details
 105461 of mapping them to printable sequences is not needed by the user. Historical implementations,
 105462 for security reasons, disallow the transmission of non-printable characters that may send
 105463 commands to the other terminal.

- 105464 **FUTURE DIRECTIONS**
- 105465 None.
- 105466 **SEE ALSO**
- 105467 *mesg, stty, who, write*
- 105468 XBD [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185)
- 105469 **CHANGE HISTORY**
- 105470 First released in Issue 4.
- 105471 **Issue 6**
- 105472 This utility is marked as part of the User Portability Utilities option.



105473 **NAME**

105474 tee — duplicate standard input

105475 **SYNOPSIS**

105476 tee [-ai] [*file...*]

105477 **DESCRIPTION**

105478 The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.

105479 The *tee* utility shall not buffer output.

105480 If the **-a** option is not specified, output files shall be written (see [Section 1.1.1.4](#) (on page 2228)).

105481 **OPTIONS**

105482 The *tee* utility shall conform to XBD [Section 12.2](#) (on page 201).

105483 The following options shall be supported:

105484 **-a** Append the output to the files.

105485 **-i** Ignore the SIGINT signal.

105486 **OPERANDS**

105487 The following operands shall be supported:

105488 *file* A pathname of an output file. Processing of at least 13 *file* operands shall be

105489 supported.

105490 **STDIN**

105491 The standard input can be of any type.

105492 **INPUT FILES**

105493 None.

105494 **ENVIRONMENT VARIABLES**

105495 The following environment variables shall affect the execution of *tee*:

105496 **LANG** Provide a default value for the internationalization variables that are unset or null. |

105497 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |

105498 variables used to determine the values of locale categories.)

105499 **LC_ALL** If set to a non-empty string value, override the values of all the other

105500 internationalization variables.

105501 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as

105502 characters (for example, single-byte as opposed to multi-byte characters in

105503 arguments).

105504 **LC_MESSAGES**

105505 Determine the locale that should be used to affect the format and contents of

105506 diagnostic messages written to standard error.

105507 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC_MESSAGES**.

105508 **ASYNCHRONOUS EVENTS**

105509 Default, except that if the **-i** option was specified, SIGINT shall be ignored.

105510 **STDOUT**

105511 The standard output shall be a copy of the standard input.

105512 **STDERR**

105513 The standard error shall be used only for diagnostic messages.

105514 **OUTPUT FILES**105515 If any *file* operands are specified, the standard input shall be copied to each named file.105516 **EXTENDED DESCRIPTION**

105517 None.

105518 **EXIT STATUS**

105519 The following exit values shall be returned:

105520 0 The standard input was successfully copied to all output files.

105521 >0 An error occurred.

105522 **CONSEQUENCES OF ERRORS**105523 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*
105524 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,
105525 the default actions specified in [Section 1.4](#) (on page 2235) apply.105526 **APPLICATION USAGE**105527 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.105528 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.105529 **EXAMPLES**

105530 Save an unsorted intermediate form of the data in a pipeline:

105531

```
... | tee unsorted | sort > sorted
```

105532 **RATIONALE**105533 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or
105534 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.105535 It should be noted that early versions of BSD ignore any invalid options and accept a single '-'
105536 as an alternative to -i. They also print a message if unable to open a file:105537

```
"tee: cannot access %s\n", <pathname>
```

105538 Historical implementations ignore write errors. This is explicitly not permitted by this volume of
105539 POSIX.1-200x.105540 Some historical implementations use O_APPEND when providing append mode; others use the
105541 *lseek()* function to seek to the end-of-file after opening the file without O_APPEND. This volume
105542 of POSIX.1-200x requires functionality equivalent to using O_APPEND; see [Section 1.1.1.4](#) (on
105543 page 2228).105544 **FUTURE DIRECTIONS**

105545 None.

105546 **SEE ALSO**105547 [Chapter 1](#) (on page 2227), *cat*105548 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)105549 XSH *lseek()*105550 **CHANGE HISTORY**

105551 First released in Issue 2.

105552

Issue 6

105553

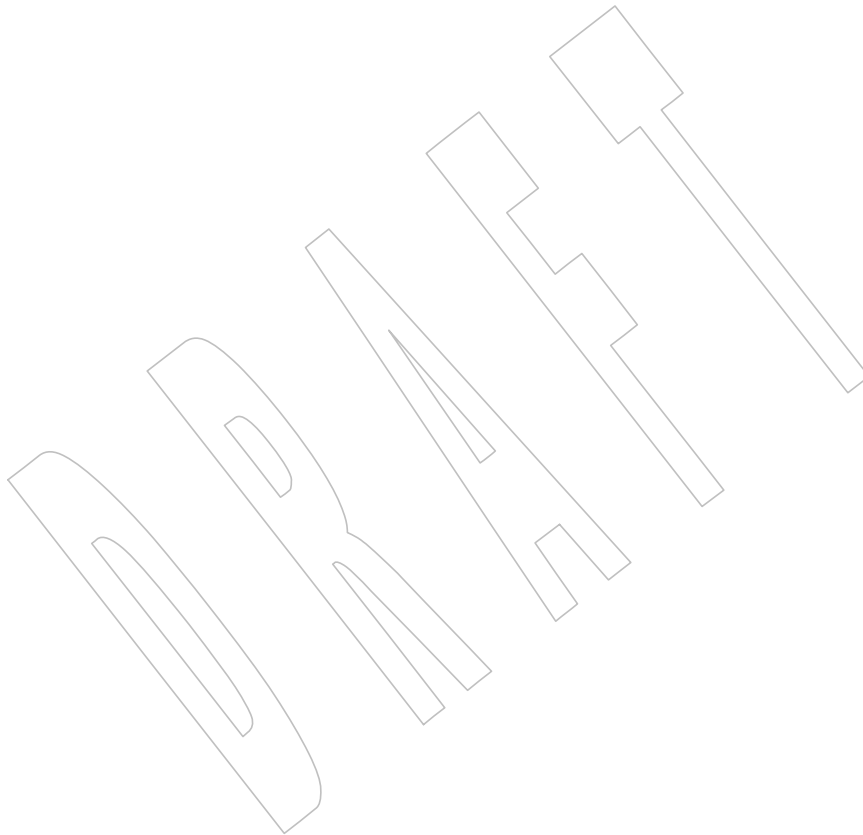
IEEE PASC Interpretation 1003.2 #168 is applied.

105554

Issue 7

105555

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



105556 **NAME**105557 `test` — evaluate expression105558 **SYNOPSIS**105559 `test` [*expression*]105560 [[*expression*]]105561 **DESCRIPTION**

105562 The *test* utility shall evaluate the *expression* and indicate the result of the evaluation by its exit
 105563 status. An exit status of zero indicates that the expression evaluated as true and an exit status of
 105564 1 indicates that the expression evaluated as false.

105565 In the second form of the utility, which uses "[]" rather than *test*, the application shall ensure
 105566 that the square brackets are separate arguments.

105567 **OPTIONS**

105568 The *test* utility shall not recognize the "--" argument in the manner specified by Guideline 10 in
 105569 XBD Section 12.2 (on page 201).

105570 No options shall be supported.

105571 **OPERANDS**

105572 The application shall ensure that all operators and elements of primaries are presented as
 105573 separate arguments to the *test* utility.

105574 The following primaries can be used to construct *expression*:

105575 **-b** *pathname* True if *pathname* resolves to a file that exists and is a block special file. False if
 105576 *pathname* cannot be resolved, or if *pathname* resolves to a file that exists but is not a
 105577 block special file.

105578 **-c** *pathname* True if *pathname* resolves to a file that exists and is a character special file. False if
 105579 *pathname* cannot be resolved, or if *pathname* resolves to a file that exists but is not a
 105580 character special file.

105581 **-d** *pathname* True if *pathname* resolves to a file that exists and is a directory. False if *pathname*
 105582 cannot be resolved, or if *pathname* resolves to a file that exists but is not a directory.

105583 **-e** *pathname* True if *pathname* resolves to a file that exists. False if *pathname* cannot be resolved.

105584 **-f** *pathname* True if *pathname* resolves to a file that exists and is a regular file. False if *pathname*
 105585 cannot be resolved, or if *pathname* resolves to a file that exists but is not a regular
 105586 file.

105587 **-g** *pathname* True if *pathname* resolves to a file that exists and has its set-group-ID flag set. False
 105588 if *pathname* cannot be resolved, or if *pathname* resolves to a file that exists but does
 105589 not have its set-group-ID flag set.

105590 **-h** *pathname* True if *pathname* resolves to a file that exists and is a symbolic link. False if
 105591 *pathname* cannot be resolved, or if *pathname* resolves to a file that exists but is not a
 105592 symbolic link. If the final component of *pathname* is a symlink, that symlink is not
 105593 followed.

105594 **-L** *pathname* True if *pathname* resolves to a file that exists and is a symbolic link. False if
 105595 *pathname* cannot be resolved, or if *pathname* resolves to a file that exists but is not a
 105596 symbolic link. If the final component of *pathname* is a symlink, that symlink is not
 105597 followed.

105598	-n <i>string</i>	True if the length of <i>string</i> is non-zero; otherwise, false.
105599	-p <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and is a FIFO. False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file that exists but is not a FIFO.
105600		
105601	-r <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and for which permission to read from the file will be granted, as defined in Section 1.1.1.4 (on page 2228). False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file for which permission to read from the file will not be granted.
105602		
105603		
105604		
105605	-S <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and is a socket. False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file that exists but is not a socket.
105606		
105607	-s <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and has a size greater than zero. False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file that exists but does not have a size greater than zero.
105608		
105609		
105610	-t <i>file_descriptor</i>	
105611		True if file descriptor number <i>file_descriptor</i> is open and is associated with a terminal. False if <i>file_descriptor</i> is not a valid file descriptor number, or if file descriptor number <i>file_descriptor</i> is not open, or if it is open but is not associated with a terminal.
105612		
105613		
105614		
105615	-u <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and has its set-user-ID flag set. False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file that exists but does not have its set-user-ID flag set.
105616		
105617		
105618	-w <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and for which permission to write to the file will be granted, as defined in Section 1.1.1.4 (on page 2228). False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file for which permission to write to the file will not be granted.
105619		
105620		
105621		
105622	-x <i>pathname</i>	True if <i>pathname</i> resolves to a file that exists and for which permission to execute the file (or search it, if it is a directory) will be granted, as defined in Section 1.1.1.4 (on page 2228). False if <i>pathname</i> cannot be resolved, or if <i>pathname</i> resolves to a file for which permission to execute (or search) the file will not be granted.
105623		
105624		
105625		
105626	-z <i>string</i>	True if the length of string <i>string</i> is zero; otherwise, false.
105627	<i>string</i>	True if the string <i>string</i> is not the null string; otherwise, false.
105628	<i>s1 = s2</i>	True if the strings <i>s1</i> and <i>s2</i> are identical; otherwise, false.
105629	<i>s1 != s2</i>	True if the strings <i>s1</i> and <i>s2</i> are not identical; otherwise, false.
105630	<i>n1 -eq n2</i>	True if the integers <i>n1</i> and <i>n2</i> are algebraically equal; otherwise, false.
105631	<i>n1 -ne n2</i>	True if the integers <i>n1</i> and <i>n2</i> are not algebraically equal; otherwise, false.
105632	<i>n1 -gt n2</i>	True if the integer <i>n1</i> is algebraically greater than the integer <i>n2</i> ; otherwise, false.
105633	<i>n1 -ge n2</i>	True if the integer <i>n1</i> is algebraically greater than or equal to the integer <i>n2</i> ; otherwise, false.
105634		
105635	<i>n1 -lt n2</i>	True if the integer <i>n1</i> is algebraically less than the integer <i>n2</i> ; otherwise, false.
105636	<i>n1 -le n2</i>	True if the integer <i>n1</i> is algebraically less than or equal to the integer <i>n2</i> ; otherwise, false.
105637		
105638	OB XSI expression1 -a expression2	
105639		True if both <i>expression1</i> and <i>expression2</i> are true; otherwise, false. The -a binary primary is left associative. It has a higher precedence than -o .
105640		

- 105641 OB XSI `expression1 -o expression2`
 105642 True if either *expression1* or *expression2* is true; otherwise, false. The `-o` binary
 105643 primary is left associative.
- 105644 With the exception of the `-h file` and `-L file` primaries, if a *file* argument is a symbolic link, *test*
 105645 shall evaluate the expression by resolving the symbolic link and using the file referenced by the
 105646 link.
- 105647 These primaries can be combined with the following operators:
- 105648 `! expression` True if *expression* is false. False if *expression* is true.
- 105649 OB XSI `(expression)` True if *expression* is true. False if *expression* is false. The parentheses can be used to
 105650 alter the normal precedence and associativity.
- 105651 The primaries with two elements of the form: -
 105652 `-primary_operator primary_operand`
- 105653 are known as *unary primaries*. The primaries with three elements in either of the two forms: -
 105654 `primary_operand -primary_operator primary_operand`
 105655 `primary_operand primary_operator primary_operand`
- 105656 are known as *binary primaries*. Additional implementation-defined operators and
 105657 *primary_operators* may be provided by implementations. They shall be of the form `-operator`
 105658 where the first character of *operator* is not a digit.
- 105659 The algorithm for determining the precedence of the operators and the return value that shall be
 105660 generated is based on the number of arguments presented to *test*. (However, when using the
 105661 "`[. . .]`" form, the right-bracket final argument shall not be counted in this algorithm.)
- 105662 In the following list, \$1, \$2, \$3, and \$4 represent the arguments presented to *test*:
- 105663 0 arguments: Exit false (1).
- 105664 1 argument: Exit true (0) if \$1 is not null; otherwise, exit false.
- 105665 2 arguments:
- If \$1 is '!', exit true if \$2 is null, false if \$2 is not null.
 - If \$1 is a unary primary, exit true if the unary test is true, false if the unary test is false.
 - Otherwise, produce unspecified results.
- 105666
- 105667
- 105668
- 105669 3 arguments:
- If \$2 is a binary primary, perform the binary test of \$1 and \$3.
 - If \$1 is '!', negate the two-argument test of \$2 and \$3.
 - If \$1 is '(', and \$3 is ')', perform the unary test of \$2. On systems that do not support the XSI option, the results are unspecified if \$1 is ' (' and \$3 is ') '.
 - Otherwise, produce unspecified results.
- 105670
- 105671 OB XSI
- 105672
- 105673
- 105674
- 105675 4 arguments:
- If \$1 is '!', negate the three-argument test of \$2, \$3, and \$4.
 - If \$1 is ' (' and \$4 is ') ', perform the two-argument test of \$2 and \$3. On systems that do not support the XSI option, the results are unspecified if \$1 is ' (' and \$4 is ') '.
 - Otherwise, the results are unspecified.
- 105676 OB XSI
- 105677
- 105678
- 105679
- 105680 >4 arguments: The results are unspecified.

test

Utilities

105681 OB XSI On XSI-conformant systems, combinations of primaries and operators shall be
 105682 evaluated using the precedence and associativity rules described previously.
 105683 In addition, the string comparison binary primaries '=' and '! =' shall have
 105684 a higher precedence than any unary primary.

105685 **STDIN**

105686 Not used.

105687 **INPUT FILES**

105688 None.

105689 **ENVIRONMENT VARIABLES**105690 The following environment variables shall affect the execution of *test*:

105691 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 105692 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 105693 variables used to determine the values of locale categories.)

105694 *LC_ALL* If set to a non-empty string value, override the values of all the other
 105695 internationalization variables.

105696 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
 105697 characters (for example, single-byte as opposed to multi-byte characters in
 105698 arguments).

105699 *LC_MESSAGES*

105700 Determine the locale that should be used to affect the format and contents of
 105701 diagnostic messages written to standard error.

105702 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

105703 **ASYNCHRONOUS EVENTS**

105704 Default.

105705 **STDOUT**

105706 Not used.

105707 **STDERR**

105708 The standard error shall be used only for diagnostic messages.

105709 **OUTPUT FILES**

105710 None.

105711 **EXTENDED DESCRIPTION**

105712 None.

105713 **EXIT STATUS**

105714 The following exit values shall be returned:

105715 0 *expression* evaluated to true.105716 1 *expression* evaluated to false or *expression* was missing.

105717 >1 An error occurred.

105718 **CONSEQUENCES OF ERRORS**

105719 Default.

APPLICATION USAGE

The XSI extensions specifying the `-a` and `-o` binary primaries and the `'(' and ')'` operators have been marked obsolescent. (Many expressions using them are ambiguously defined by the grammar depending on the specific expressions being evaluated.) Scripts using these expressions should be converted to the forms given below. Even though many implementations will continue to support these obsolescent forms, scripts should be extremely careful when dealing with user-supplied input that could be confused with these and other primaries and operators. Unless the application developer knows all the cases that produce input to the script, invocations like:

```
test "$1" -a "$2"
```

should be written as:

```
test "$1" && test "$2"
```

to avoid problems if a user supplied values such as \$1 set to `'!'` and \$2 set to the null string. That is, in cases where maximal portability is of concern, replace:

```
test expr1 -a expr2
```

with:

```
test expr1 && test expr2
```

and replace:

```
test expr1 -o expr2
```

with:

```
test expr1 || test expr2
```

but note that, in *test*, `-a` has higher precedence than `-o` while `"&&"` and `"||"` have equal precedence in the shell.

Parentheses or braces can be used in the shell command language to effect grouping.

Parentheses must be escaped when using *sh*; for example:

```
test \( expr1 -a expr2 \) -o expr3
```

This command is not always portable even on XSI-conformant systems depending on the expressions specified by *expr1*, *expr2*, and *expr3*. The following form can be used instead:

```
( test expr1 && test expr2 ) || test expr3
```

The two commands:

```
test "$1"
test ! "$1"
```

could not be used reliably on some historical systems. Unexpected results would occur if such a *string* expression were used and \$1 expanded to `'!'`, `'('`, or a known unary primary. Better constructs are:

```
test -n "$1"
test -z "$1"
```

respectively.

Historical systems have also been unreliable given the common construct:

```
test "$response" = "expected string"
```

One of the following is a more reliable form:

```
105761 test "X$response" = "Xexpected string"
105762 test "expected string" = "$response"
```

105763 Note that the second form assumes that *expected string* could not be confused with any unary
 105764 primary. If *expected string* starts with '-', '(', '!', or even '=', the first form should be used
 105765 instead. Using the preceding rules without the XSI marked extensions, any of the three
 105766 comparison forms is reliable, given any input. (However, note that the strings are quoted in all
 105767 cases.)

105768 Because the string comparison binary primaries, '=' and '!=', have a higher precedence than
 105769 any unary primary in the greater than 4 argument case, unexpected results can occur if
 105770 arguments are not properly prepared. For example, in:

```
105771 test -d $1 -o -d $2
```

105772 If \$1 evaluates to a possible directory name of '=', the first three arguments are considered a
 105773 string comparison, which shall cause a syntax error when the second -d is encountered. One of
 105774 the following forms prevents this; the second is preferred:

```
105775 test \( -d "$1" \) -o \( -d "$2" \)
105776 test -d "$1" || test -d "$2"
```

105777 Also in the greater than 4 argument case:

```
105778 test "$1" = "bat" -a "$2" = "ball"
```

105779 syntax errors occur if \$1 evaluates to '(' or '!'. One of the following forms prevents this; the
 105780 third is preferred:

```
105781 test "X$1" = "Xbat" -a "X$2" = "Xball"
105782 test "$1" = "bat" && test "$2" = "ball"
105783 test "X$1" = "Xbat" && test "X$2" = "Xball"
```

105784 EXAMPLES

- 105785 1. Exit if there are not two or three arguments (two variations):

```
105786 if [ $# -ne 2 ] && [ $# -ne 3 ]; then exit 1; fi
105787 if [ $# -lt 2 ] || [ $# -gt 3 ]; then exit 1; fi
```

- 105788 2. Perform a *mkdir* if a directory does not exist:

```
105789 test ! -d tempdir && mkdir tempdir
```

- 105790 3. Wait for a file to become non-readable:

```
105791 while test -r thefile
105792 do
105793     sleep 30
105794 done
105795 echo "thefile" is no longer readable'
```

- 105796 4. Perform a command if the argument is one of three strings (two variations):

```
105797 if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
105798 then
105799     command
105800 fi
105801 case "$1" in
105802     pear|grape|apple) command ;;
105803 esac
```


RATIONALE

The KornShell-derived conditional command (double bracket `[[]]`) was removed from the shell command language description in an early proposal. Objections were raised that the real problem is misuse of the `test` command (`I`), and putting it into the shell is the wrong way to fix the problem. Instead, proper documentation and a new shell reserved word (`!`) are sufficient.

Tests that require multiple `test` operations can be done at the shell level using individual invocations of the `test` command and shell logicals, rather than using the error-prone `-o` flag of `test`.

XSI-conformant systems support more than four arguments.

XSI-conformant systems support the combining of primaries with the following constructs:

expression1 `-a` *expression2*

True if both *expression1* and *expression2* are true.

expression1 `-o` *expression2*

True if at least one of *expression1* and *expression2* are true.

(*expression*)

True if *expression* is true.

In evaluating these more complex combined expressions, the following precedence rules are used:

- The unary primaries have higher precedence than the algebraic binary primaries.
- The unary primaries have lower precedence than the string binary primaries.
- The unary and binary primaries have higher precedence than the unary *string* primary.
- The `!` operator has higher precedence than the `-a` operator, and the `-a` operator has higher precedence than the `-o` operator.
- The `-a` and `-o` operators are left associative.
- The parentheses can be used to alter the normal precedence and associativity.

The BSD and System V versions of `-f` are not the same. The BSD definition was:

`-f file` True if *file* exists and is not a directory.

The SVID version (true if the file exists and is a regular file) was chosen for this volume of POSIX.1-200x because its use is consistent with the `-b`, `-c`, `-d`, and `-p` operands (*file* exists and is a specific file type).

The `-e` primary, possessing similar functionality to that provided by the C shell, was added because it provides the only way for a shell script to find out if a file exists without trying to open the file. Since implementations are allowed to add additional file types, a portable script cannot use:

```
test -b foo -o -c foo -o -d foo -o -f foo -o -p foo
```

to find out if `foo` is an existing file. On historical BSD systems, the existence of a file could be determined by:

```
test -f foo -o -d foo
```

but there was no easy way to determine that an existing file was a regular file. An early proposal used the KornShell `-a` primary (with the same meaning), but this was changed to `-e` because there were concerns about the high probability of humans confusing the `-a` primary with the `-a` binary operator.

The following options were not included in this volume of POSIX.1-200x, although they are

105847 provided by some implementations. These operands should not be used by new
105848 implementations for other purposes:

105849 **-k file** True if *file* exists and its sticky bit is set.

105850 **-C file** True if *file* is a contiguous file.

105851 **-V file** True if *file* is a version file.

105852 The following option was not included because it was undocumented in most implementations,
105853 has been removed from some implementations (including System V), and the functionality is
105854 provided by the shell (see [Section 2.6.2](#) (on page 2254).

105855 **-l string** The length of the string *string*.

105856 The **-b**, **-c**, **-g**, **-p**, **-u**, and **-x** operands are derived from the SVID; historical BSD does not
105857 provide them. The **-k** operand is derived from System V; historical BSD does not provide it.

105858 On historical BSD systems, *test -w directory* always returned false because *test* tried to open the
105859 directory for writing, which always fails.

105860 Some additional primaries newly invented or from the KornShell appeared in an early proposal
105861 as part of the conditional command (`[[]]`): *s1 > s2*, *s1 < s2*, *str = pattern*, *str != pattern*, *f1 -nt f2*, *f1*
105862 *-ot f2*, and *f1 -ef f2*. They were not carried forward into the *test* utility when the conditional
105863 command was removed from the shell because they have not been included in the *test* utility
105864 built into historical implementations of the *sh* utility.

105865 The **-t file_descriptor** primary is shown with a mandatory argument because the grammar is
105866 ambiguous if it can be omitted. Historical implementations have allowed it to be omitted,
105867 providing a default of 1.

105868 FUTURE DIRECTIONS

105869 None.

105870 SEE ALSO

105871 [Section 1.1.1.4](#) (on page 2228), *find*

105872 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

105873 CHANGE HISTORY

105874 First released in Issue 2.

105875 Issue 5

105876 The FUTURE DIRECTIONS section is added.

105877 Issue 6

105878 The **-h** operand is added for symbolic links, and access permission requirements are clarified for
105879 the **-r**, **-w**, and **-x** operands to align with the IEEE P1003.2b draft standard.

105880 The normative text is reworded to avoid use of the term “must” for application requirements.

105881 The **-L** and **-S** operands are added for symbolic links and sockets.

105882 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/38 is applied, adding XSI margin
105883 marking and shading to a line in the OPERANDS section referring to the use of parentheses as
105884 arguments to the *test* utility.

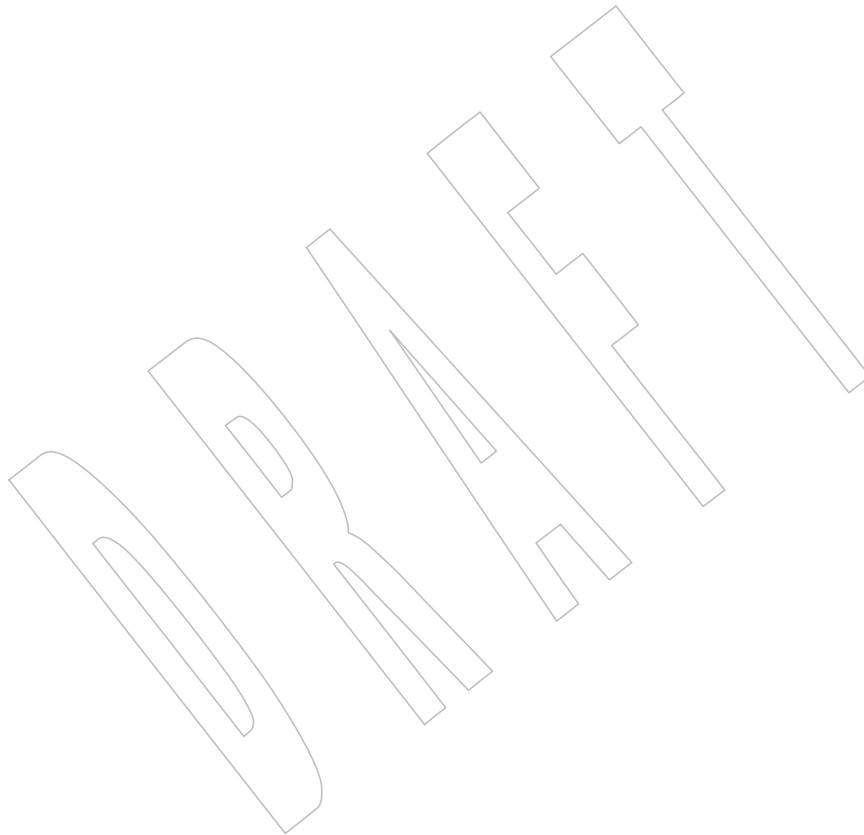
105885 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/30 is applied, rewording the existence
105886 primaries for the *test* utility.

105887

Issue 7

105888

Austin Group Interpretation 1003.1-2001 #107 is applied.



105889 **NAME**105890 `time` — `time` a simple command105891 **SYNOPSIS**105892 `time [-p] utility [argument...]`105893 **DESCRIPTION**105894 The `time` utility shall invoke the utility named by the `utility` operand with arguments supplied as
105895 the `argument` operands and write a message to standard error that lists timing statistics for the
105896 utility. The message shall include the following information:

- 105897 • The elapsed (real) time between invocation of `utility` and its termination.
- 105898 • The User CPU time, equivalent to the sum of the `tms_utime` and `tms_cutime` fields returned
105899 by the `times()` function defined in the System Interfaces volume of POSIX.1-200x for the
105900 process in which `utility` is executed.
- 105901 • The System CPU time, equivalent to the sum of the `tms_stime` and `tms_cstime` fields
105902 returned by the `times()` function for the process in which `utility` is executed.

105903 The precision of the timing shall be no less than the granularity defined for the size of the clock
105904 tick unit on the system, but the results shall be reported in terms of standard time units (for
105905 example, 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

105906 When `time` is used as part of a pipeline, the times reported are unspecified, except when it is the
105907 sole command within a grouping command (see [Section 2.9.4.1](#), on page 2268) in that pipeline.
105908 For example, the commands on the left are unspecified; those on the right report on utilities `a`
105909 and `c`, respectively:

```
105910 time a | b | c      { time a; } | b | c
105911 a | b | time c     a | b | (time c)
```

105912 **OPTIONS**105913 The `time` utility shall conform to XBD [Section 12.2](#) (on page 201).

105914 The following option shall be supported:

105915 `-p` Write the timing output to standard error in the format shown in the `STDERR`
105916 section.

105917 **OPERANDS**

105918 The following operands shall be supported:

105919 `utility` The name of a utility that is to be invoked. If the `utility` operand names any of the
105920 special built-in utilities in [Section 2.14](#) (on page 2280), the results are undefined.

105921 `argument` Any string to be supplied as an argument when invoking the utility named by the
105922 `utility` operand.

105923 **STDIN**

105924 Not used.

105925 **INPUT FILES**

105926 None.

105927 **ENVIRONMENT VARIABLES**105928 The following environment variables shall affect the execution of `time`:

105929 `LANG` Provide a default value for the internationalization variables that are unset or null.
105930 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
105931 variables used to determine the values of locale categories.)

105932		<i>LC_ALL</i>	If set to a non-empty string value, override the values of all the other internationalization variables.
105933			
105934		<i>LC_CTYPE</i>	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
105935			
105936			
105937		<i>LC_MESSAGES</i>	
105938			Determine the locale that should be used to affect the format and contents of diagnostic and informative messages written to standard error.
105939			
105940		<i>LC_NUMERIC</i>	
105941			Determine the locale for numeric formatting.
105942	XSI	<i>NLSPATH</i>	Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .
105943		<i>PATH</i>	Determine the search path that shall be used to locate the utility to be invoked; see XBD Chapter 8 (on page 159).
105944			

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used to write the timing statistics. If **-p** is specified, the following format shall be used in the POSIX locale:

```
"real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,
  <system seconds>
```

where each floating-point number shall be expressed in seconds. The precision used may be less than the default six digits of %f, but shall be sufficiently precise to accommodate the size of the clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits shall follow the radix character). The number of digits following the radix character shall be no less than one, even if this always results in a trailing zero. The implementation may append white space and additional information following the format shown here.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise, the *time* utility shall exit with one of the following values:

- 1-125 An error occurred in the *time* utility.
- 126 The utility specified by *utility* was found but could not be invoked.
- 127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication”. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

EXAMPLES

It is frequently desirable to apply *time* to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the *time* applies to everything in the file.

Alternatively, the following command can be used to apply *time* to a complex command:

```
time sh -c 'complex-command-line'
```

RATIONALE

When the *time* utility was originally proposed to be included in the ISO POSIX-2:1993 standard, questions were raised about its suitability for inclusion on the grounds that it was not useful for conforming applications, specifically:

- The underlying CPU definitions from the System Interfaces volume of POSIX.1-200x are vague, so the numeric output could not be compared accurately between systems or even between invocations.
- The creation of portable benchmark programs was outside the scope this volume of POSIX.1-200x.

However, *time* does fit in the scope of user portability. Human judgement can be applied to the analysis of the output, and it could be very useful in hands-on debugging of applications or in providing subjective measures of system performance. Hence it has been included in this volume of POSIX.1-200x.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting this numeric output. The `-p` option was invented to provide scripts with a common means of obtaining this information.

In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather than just a simple command. The POSIX definition has been worded to allow this implementation. Consideration was given to invalidating this approach because of the historical model from the C shell and System V shell. However, since the System V *time* utility historically has not produced accurate results in pipeline timing (because the constituent processes are not all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to break historical KornShell usage.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

FUTURE DIRECTIONS

None.

106017 **SEE ALSO**106018 [Chapter 2](#) (on page 2245), *sh*106019 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)106020 XSH *times()*106021 **CHANGE HISTORY**

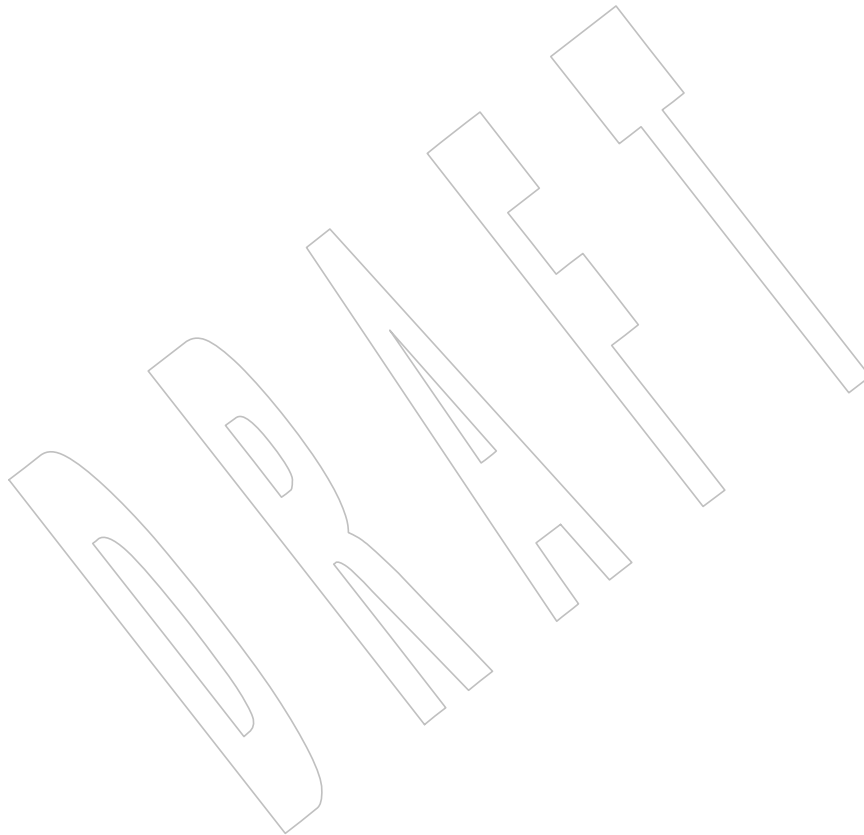
106022 First released in Issue 2.

106023 **Issue 6**

106024 This utility is marked as part of the User Portability Utilities option.

106025 **Issue 7**106026 The *time* utility is moved from the User Portability Utilities option to the Base. User Portability
106027 Utilities is now an option for interactive utilities.

106028 SD5-XCU-ERN-115 is applied, updating the example in the DESCRIPTION.



106029 **NAME**
 106030 touch — change file access and modification times

106031 **SYNOPSIS**
 106032 touch [-acm] [-r *ref_file*|-t *time*] *file*...

106033 **DESCRIPTION**
 106034 The *touch* utility shall change the last data modification timestamps, the last data access
 106035 timestamps, or both.

106036 The time used can be specified by the **-t *time*** option-argument, the corresponding time fields of
 106037 the file referenced by the **-r *ref_file*** option-argument, or the *date_time* operand, as specified in the
 106038 following sections. If none of these are specified, *touch* shall use the current time (the value
 106039 returned by the equivalent of the *time()* function defined in the System Interfaces volume of
 106040 POSIX.1-200x).

106041 For each *file* operand, *touch* shall perform actions equivalent to the following functions defined
 106042 in the System Interfaces volume of POSIX.1-200x:

- 106043 1. If *file* does not exist:
 - 106044 a. The *creat()* function is called with the following arguments:
 - 106045 — The *file* operand is used as the *path* argument.
 - 106046 — The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP,
 106047 S_IWGRP, S_IROTH, and S_IWOTH is used as the *mode* argument.
 - 106048 b. The *futimens()* function is called with the following arguments:
 - 106049 — The file descriptor opened in step 1a.
 - 106050 — The access time and the modification time, set as described in the OPTIONS
 106051 section, are used as the first and second elements of the *times* array argument,
 106052 respectively.
- 106053 2. If *file* exists, the *utimensat()* function is called with the following arguments:
 - 106054 a. The AT_FDCWD special value is used as the *fd* argument. +
 - 106055 b. The *file* operand is used as the *path* argument.
 - 106056 c. The access time and the modification time, set as described in the OPTIONS +
 106057 section, are used as the first and second elements of the *times* array argument, +
 106058 respectively. +
 - 106059 d. The *flag* argument is set to zero. |

106060 **OPTIONS**
 106061 The *touch* utility shall conform to XBD Section 12.2 (on page 201). |

106062 The following options shall be supported:

- 106063 **-a** Change the access time of *file*. Do not change the modification time unless **-m** is
 106064 also specified.
- 106065 **-c** Do not create a specified *file* if it does not exist. Do not write any diagnostic
 106066 messages concerning this condition.
- 106067 **-m** Change the modification time of *file*. Do not change the access time unless **-a** is
 106068 also specified.

- 106069 **-r** *ref_file* Use the corresponding time of the file named by the pathname *ref_file* instead of
106070 the current time.
- 106071 **-t** *time* Use the specified *time* instead of the current time. The option-argument shall be a
106072 decimal number of the form:
- 106073 [[CC]YY]MMDDhhmm[.SS]
- 106074 where each two digits represents the following:
- 106075 *MM* The month of the year [01,12].
- 106076 *DD* The day of the month [01,31].
- 106077 *hh* The hour of the day [00,23].
- 106078 *mm* The minute of the hour [00,59].
- 106079 *CC* The first two digits of the year (the century).
- 106080 *YY* The second two digits of the year.
- 106081 *SS* The second of the minute [00,60].

106082 Both *CC* and *YY* shall be optional. If neither is given, the current year shall be
106083 assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

If YY is:	CC becomes:
[69,99]	19
[00,68]	20

106087 **Note:** It is expected that in a future version of this standard the default century inferred
106088 from a 2-digit year will change. (This would apply to all commands accepting a
106089 2-digit year as input.)

106090 The resulting time shall be affected by the value of the *TZ* environment variable. If
106091 the resulting time value precedes the Epoch, the behavior is implementation-
106092 defined. If the time is out of range for the file's timestamp, *touch* shall exit
106093 immediately with an error status. The range of valid times past the Epoch is
106094 implementation-defined, but it shall extend to at least the time 0 hours, 0 minutes,
106095 0 seconds, January 1, 2038, Coordinated Universal Time. Some implementations
106096 may not be able to represent dates beyond January 18, 2038, because they use
106097 **signed int** as a time holder.

106098 The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60,
106099 and the resulting time, as affected by the *TZ* environment variable, does not refer
106100 to a leap second, the resulting time shall be one second after a time where *SS* is 59.
106101 If *SS* is not given a value, it is assumed to be zero.

106102 If neither the **-a** nor **-m** options were specified, *touch* shall behave as if both the **-a** and **-m**
106103 options were specified.

106104 OPERANDS

106105 The following operands shall be supported:

106106 *file* A pathname of a file whose times shall be modified.

106107 STDIN

106108 Not used.

touch

Utilities

106109 **INPUT FILES**

106110 None.

106111 **ENVIRONMENT VARIABLES**106112 The following environment variables shall affect the execution of *touch*:

106113 *LANG* Provide a default value for the internationalization variables that are unset or null. |
 106114 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |
 106115 variables used to determine the values of locale categories.)

106116 *LC_ALL* If set to a non-empty string value, override the values of all the other |
 106117 internationalization variables.

106118 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as |
 106119 characters (for example, single-byte as opposed to multi-byte characters in |
 106120 arguments).

106121 *LC_MESSAGES*

106122 Determine the locale that should be used to affect the format and contents of |
 106123 diagnostic messages written to standard error.

106124 *XSIPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

106125 *TZ* Determine the timezone to be used for interpreting the *time* option-argument. If *TZ* |
 106126 is unset or null, an unspecified default timezone shall be used.

106127 **ASYNCHRONOUS EVENTS**

106128 Default.

106129 **STDOUT**

106130 Not used.

106131 **STDERR**

106132 The standard error shall be used only for diagnostic messages.

106133 **OUTPUT FILES**

106134 None.

106135 **EXTENDED DESCRIPTION**

106136 None.

106137 **EXIT STATUS**

106138 The following exit values shall be returned:

106139 0 The utility executed successfully and all requested changes were made.

106140 >0 An error occurred.

106141 **CONSEQUENCES OF ERRORS**

106142 Default.

106143 **APPLICATION USAGE**

106144 The interpretation of time is taken to be *seconds since the Epoch* (see XBD Section 4.15, on page |
 106145 100). It should be noted that implementations conforming to the System Interfaces volume of |
 106146 POSIX.1-200x do not take leap seconds into account when computing seconds since the Epoch. |
 106147 When *SS=60* is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time |
 106148 when *SS=59*.

106149 Although the *-t time* option-argument specifies values in 1969, the access time and modification |
 106150 time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC). |
 106151 Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid |
 106152 hours in 1969 and there need not be any valid times in 1969.

106153 One ambiguous situation occurs if *-t time* is not specified, *-r ref_file* is not specified, and the first

106154 operand is an eight or ten-digit decimal number. A portable script can avoid this problem by
 106155 using:

106156 touch -- file

106157 or:

106158 touch ./file

106159 in this case.

106160 EXAMPLES

106161 None.

106162 RATIONALE

106163 The functionality of *touch* is described almost entirely through references to functions in the
 106164 System Interfaces volume of POSIX.1-200x. In this way, there is no duplication of effort required
 106165 for describing such side effects as the relationship of user IDs to the user database, permissions,
 106166 and so on.

106167 There are some significant differences between the *touch* utility in this volume of POSIX.1-200x
 106168 and those in System V and BSD systems. They are upwards-compatible for historical
 106169 applications from both implementations:

106170 1. In System V, an ambiguity exists when a pathname that is a decimal number leads the
 106171 operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be
 106172 touched to the current time. The `-t time` construct solves these problems for future
 106173 conforming applications (note that the `-t` option is not historical practice).

106174 2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is
 106175 treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates
 106176 following the Epoch was included as recognition that some implementations are not able
 106177 to represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

106178 The `-r` option was added because several comments requested this capability. This option was
 106179 named `-f` in an early proposal, but was changed because the `-f` option is used in the BSD
 106180 version of *touch* with a different meaning.

106181 At least one historical implementation of *touch* incremented the exit code if `-c` was specified and
 106182 the file did not exist. This volume of POSIX.1-200x requires exit status zero if no errors occur.

106183 In previous editions of the standard, if at least two operands are specified, and the first operand
 106184 is an eight or ten-digit decimal integer, the first operand was assumed to be a *date_time* operand.
 106185 This usage was removed in this edition of the standard since it had been marked obsolescent
 106186 previously.

106187 FUTURE DIRECTIONS

106188 None.

106189 SEE ALSO

106190 *date*

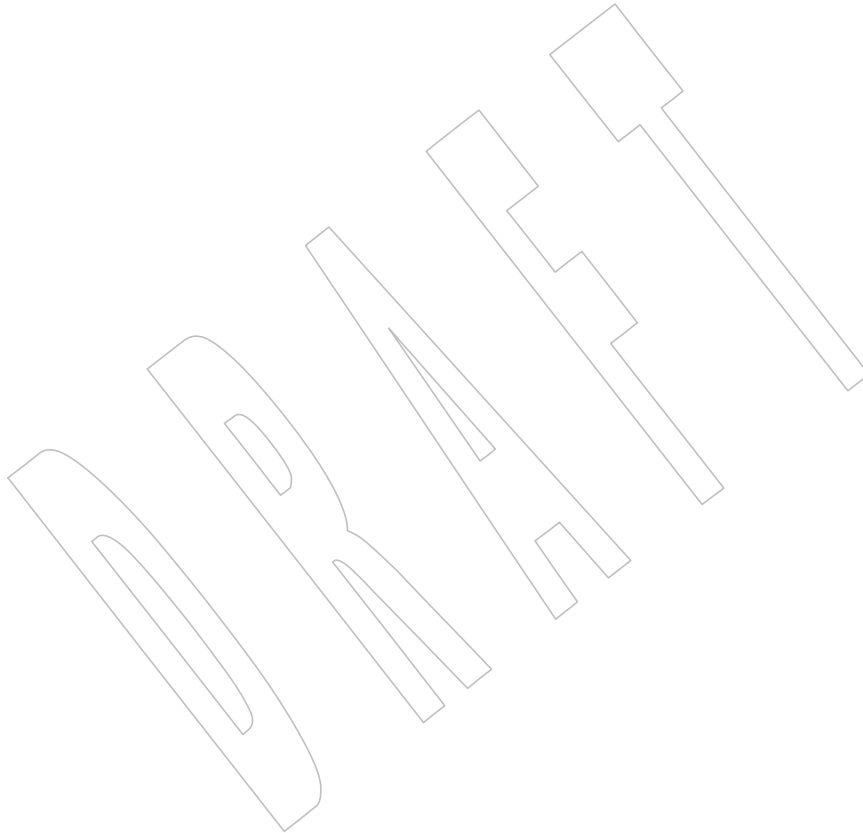
106191 XBD [Section 4.15](#) (on page 100), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201), [<sys/stat.h>](#)

106192 XSH [creat\(\)](#), [futimens\(\)](#), [time](#), [utime\(\)](#)

106193 CHANGE HISTORY

106194 First released in Issue 2.

106195	Issue 6	
106196		The obsolescent <i>date_time</i> operand is removed.
106197		The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.
106198		
106199		
106200		The range for seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899:1999 standard, and to allow for positive leap seconds.
106201		
106202	Issue 7	
106203		Austin Group Interpretation 1003.1-2001 #118 is applied. +
106204		SD5-XCU-ERN-45 is applied, adding a new paragraph to the RATIONALE.
106205		SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.
106206		SD5-XCU-ERN-110 is applied, updating the OPTIONS section.
106207		Changes are made related to support for finegrained timestamps. +



106208 **NAME**
 106209 tput — change terminal characteristics

106210 **SYNOPSIS**
 106211 tput [-T *type*] *operand*...

106212 **DESCRIPTION**
 106213 The *tput* utility shall display terminal-dependent information. The manner in which this
 106214 information is retrieved is unspecified. The information displayed shall clear the terminal
 106215 screen, initialize the user's terminal, or reset the user's terminal, depending on the operand
 106216 given. The exact consequences of displaying this information are unspecified.

106217 **OPTIONS**
 106218 The *tput* utility shall conform to XBD [Section 12.2](#) (on page 201).

106219 The following option shall be supported:

106220 **-T *type*** Indicate the type of terminal. If this option is not supplied and the *TERM* variable
 106221 is unset or null, an unspecified default terminal type shall be used. The setting of
 106222 *type* shall take precedence over the value in *TERM*.

106223 **OPERANDS**
 106224 The following strings shall be supported as operands by the implementation in the POSIX locale:

106225 **clear** Display the clear-screen sequence.
 106226 **init** Display the sequence that initializes the user's terminal in an implementation-
 106227 defined manner.
 106228 **reset** Display the sequence that resets the user's terminal in an implementation-defined
 106229 manner.

106230 If a terminal does not support any of the operations described by these operands, this shall not
 106231 be considered an error condition.

106232 **STDIN**
 106233 Not used.

106234 **INPUT FILES**
 106235 None.

106236 **ENVIRONMENT VARIABLES**
 106237 The following environment variables shall affect the execution of *tput*:

106238 **LANG** Provide a default value for the internationalization variables that are unset or null.
 106239 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization
 106240 variables used to determine the values of locale categories.)

106241 **LC_ALL** If set to a non-empty string value, override the values of all the other
 106242 internationalization variables.

106243 **LC_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as
 106244 characters (for example, single-byte as opposed to multi-byte characters in
 106245 arguments).

106246 **LC_MESSAGES**
 106247 Determine the locale that should be used to affect the format and contents of
 106248 diagnostic messages written to standard error.

- 106249 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC_MESSAGES*.
- 106250 **TERM** Determine the terminal type. If this variable is unset or null, and if the **-T** option is
106251 not specified, an unspecified default terminal type shall be used.

ASYNCHRONOUS EVENTS

- 106252 Default.

STDOUT

- 106255 If standard output is a terminal device, it may be used for writing the appropriate sequence to
106256 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,
106257 undefined results occur.

STDERR

- 106258 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

- 106260 None.

EXTENDED DESCRIPTION

- 106262 None.

EXIT STATUS

- 106265 The following exit values shall be returned:
- 106266 0 The requested string was written successfully.
- 106267 1 Unspecified.
- 106268 2 Usage error.
- 106269 3 No information is available about the specified terminal type.
- 106270 4 The specified operand is invalid.
- 106271 >4 An error occurred.

CONSEQUENCES OF ERRORS

- 106272 If one of the operands is not available for the terminal, *tput* continues processing the remaining
106273 operands.

APPLICATION USAGE

- 106275 The difference between resetting and initializing a terminal is left unspecified, as they vary
106276 greatly based on hardware types. In general, resetting is a more severe action.

- 106278 Some terminals use control characters to perform the stated functions, and on such terminals it
106279 might make sense to use *tput* to store the initialization strings in a file or environment variable
106280 for later use. However, because other terminals might rely on system calls to do this work, the
106281 standard output cannot be used in a portable manner, such as the following non-portable
106282 constructs:

- ```
106283 ClearVar='tput clear`
106284 tput reset | mailx -s "Wake Up" ddg
```

**EXAMPLES**

- 106286 1. Initialize the terminal according to the type of terminal in the environmental variable  
106287 *TERM*. This command can be included in a **.profile** file.
- ```
106288 tput init
```
- 106289 2. Reset a 450 terminal.
- ```
106290 tput -T 450 reset
```

**RATIONALE**

The list of operands was reduced to a minimum for the following reasons:

- The only features chosen were those that were likely to be used by human users interacting with a terminal.
- Specifying the full *terminfo* set was not considered desirable, but the standard developers did not want to select among operands.
- This volume of POSIX.1-200x does not attempt to provide applications with sophisticated terminal handling capabilities, as that falls outside of its assigned scope and intersects with the responsibilities of other standards bodies.

The difference between resetting and initializing a terminal is left unspecified as this varies greatly based on hardware types. In general, resetting is a more severe action.

The exit status of 1 is historically reserved for finding out if a Boolean operand is not set. Although the operands were reduced to a minimum, the exit status of 1 should still be reserved for the Boolean operands, for those sites that wish to support them.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*stty*, *tabs*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

**CHANGE HISTORY**

First released in Issue 4.

**Issue 6**

This utility is marked as part of the User Portability Utilities option.

**Issue 7**

The *tput* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

106317 **NAME**

106318 tr — translate characters

106319 **SYNOPSIS**106320 tr [-c|-C] [-s] *string1 string2*106321 tr -s [-c|-C] *string1*106322 tr -d [-c|-C] *string1*106323 tr -ds [-c|-C] *string1 string2*106324 **DESCRIPTION**

106325 The *tr* utility shall copy the standard input to the standard output with substitution or deletion  
 106326 of selected characters. The options specified and the *string1* and *string2* operands shall control  
 106327 translations that occur while copying characters and single-character collating elements.

106328 **OPTIONS**106329 The *tr* utility shall conform to XBD [Section 12.2](#) (on page 201).

106330 The following options shall be supported:

106331 **-c** Complement the set of values specified by *string1*. See the EXTENDED  
 106332 DESCRIPTION section.

106333 **-C** Complement the set of characters specified by *string1*. See the EXTENDED  
 106334 DESCRIPTION section.

106335 **-d** Delete all occurrences of input characters that are specified by *string1*.

106336 **-s** Replace instances of repeated characters with a single character, as described in the  
 106337 EXTENDED DESCRIPTION section.

106338 **OPERANDS**

106339 The following operands shall be supported:

106340 *string1, string2*

106341 Translation control strings. Each string shall represent a set of characters to be  
 106342 converted into an array of characters used for the translation. For a detailed  
 106343 description of how the strings are interpreted, see the EXTENDED DESCRIPTION  
 106344 section.

106345 **STDIN**

106346 The standard input can be any type of file.

106347 **INPUT FILES**

106348 None.

106349 **ENVIRONMENT VARIABLES**106350 The following environment variables shall affect the execution of *tr*:

106351 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 106352 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 106353 variables used to determine the values of locale categories.)

106354 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 106355 internationalization variables.

106356 **LC\_COLLATE**

106357 Determine the locale for the behavior of range expressions and equivalence classes.



106358 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 106359 characters (for example, single-byte as opposed to multi-byte characters in  
 106360 arguments) and the behavior of character classes.

106361 *LC\_MESSAGES*  
 106362 Determine the locale that should be used to affect the format and contents of  
 106363 diagnostic messages written to standard error.

106364 xSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 106365 ASYNCHRONOUS EVENTS

106366 Default.

## 106367 STDOUT

106368 The *tr* output shall be identical to the input, with the exception of the specified transformations.

## 106369 STDERR

106370 The standard error shall be used only for diagnostic messages.

## 106371 OUTPUT FILES

106372 None.

## 106373 EXTENDED DESCRIPTION

106374 The operands *string1* and *string2* (if specified) define two arrays of characters. The constructs in  
 106375 the following list can be used to specify characters or single-character collating elements. If any  
 106376 of the constructs result in multi-character collating elements, *tr* shall exclude, without a  
 106377 diagnostic, those multi-character elements from the resulting array.

106378 *character* Any character not described by one of the conventions below shall represent itself.

106379 *\octal* Octal sequences can be used to represent characters with specific coded values. An  
 106380 octal sequence shall consist of a backslash followed by the longest sequence of one,  
 106381 two, or three-octal-digit characters (01234567). The sequence shall cause the value  
 106382 whose encoding is represented by the one, two, or three-digit octal integer to be  
 106383 placed into the array. Multi-byte characters require multiple, concatenated escape  
 106384 sequences of this type, including the leading '*\*' for each byte.

106385 *\character* The backslash-escape sequences in XBD Table 5-1 (on page 108) ('*\\*', '*\a*',  
 106386 '*\b*', '*\f*', '*\n*', '*\r*', '*\t*', '*\v*') shall be supported. The results of using any  
 106387 other character, other than an octal digit, following the backslash are unspecified.

106388 *c-c* In the POSIX locale, this construct shall represent the range of collating elements  
 106389 between the range endpoints (as long as neither endpoint is an octal sequence of  
 106390 the form *\octal*), inclusive, as defined by the collation sequence. The characters or  
 106391 collating elements in the range shall be placed in the array in ascending collation  
 106392 sequence. If the second endpoint precedes the starting endpoint in the collation  
 106393 sequence, it is unspecified whether the range of collating elements is empty, or this  
 106394 construct is treated as invalid. In locales other than the POSIX locale, this construct  
 106395 has unspecified behavior.

106396 If either or both of the range endpoints are octal sequences of the form *\octal*, this  
 106397 shall represent the range of specific coded values between the two range  
 106398 endpoints, inclusive.

106399 *[:class:]* Represents all characters belonging to the defined character class, as defined by the  
 106400 current setting of the *LC\_CTYPE* locale category. The following character class  
 106401 names shall be accepted when specified in *string1*:

|              |              |              |              |              |               |
|--------------|--------------|--------------|--------------|--------------|---------------|
| <b>alnum</b> | <b>blank</b> | <b>digit</b> | <b>lower</b> | <b>punct</b> | <b>upper</b>  |
| <b>alpha</b> | <b>cntrl</b> | <b>graph</b> | <b>print</b> | <b>space</b> | <b>xdigit</b> |

|                                                                                                                      |                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------------------------------------------------------------------------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 106404<br>106405<br>106406                                                                                           | XSI                      | In addition, character class expressions of the form <code>[:name:]</code> shall be recognized in those locales where the <code>name</code> keyword has been given a <b>charclass</b> definition in the <code>LC_CTYPE</code> category.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 106407<br>106408<br>106409<br>106410<br>106411<br>106412<br>106413<br>106414<br>106415<br>106416<br>106417<br>106418 |                          | When both the <code>-d</code> and <code>-s</code> options are specified, any of the character class names shall be accepted in <code>string2</code> . Otherwise, only character class names <b>lower</b> or <b>upper</b> are valid in <code>string2</code> and then only if the corresponding character class ( <b>upper</b> and <b>lower</b> , respectively) is specified in the same relative position in <code>string1</code> . Such a specification shall be interpreted as a request for case conversion. When <code>[:lower:]</code> appears in <code>string1</code> and <code>[:upper:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>toupper</b> mapping in the <code>LC_CTYPE</code> category of the current locale. When <code>[:upper:]</code> appears in <code>string1</code> and <code>[:lower:]</code> appears in <code>string2</code> , the arrays shall contain the characters from the <b>tolower</b> mapping in the <code>LC_CTYPE</code> category of the current locale. The first character from each mapping pair shall be in the array for <code>string1</code> and the second character from each mapping pair shall be in the array for <code>string2</code> in the same relative position. |
| 106419<br>106420                                                                                                     |                          | Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 106421<br>106422                                                                                                     |                          | If the name specified for <code>class</code> does not define a valid character class in the current locale, the behavior is undefined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 106423<br>106424<br>106425<br>106426<br>106427                                                                       | [ <code>=equiv=</code> ] | Represents all characters or collating elements belonging to the same equivalence class as <code>equiv</code> , as defined by the current setting of the <code>LC_COLLATE</code> locale category. An equivalence class expression shall be allowed only in <code>string1</code> , or in <code>string2</code> when it is being used by the combined <code>-d</code> and <code>-s</code> options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 106428<br>106429<br>106430<br>106431<br>106432<br>106433                                                             | [ <code>x*n</code> ]     | Represents <code>n</code> repeated occurrences of the character <code>x</code> . Because this expression is used to map multiple characters to one, it is only valid when it occurs in <code>string2</code> . If <code>n</code> is omitted or is zero, it shall be interpreted as large enough to extend the <code>string2</code> -based sequence to the length of the <code>string1</code> -based sequence. If <code>n</code> has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 106434                                                                                                               |                          | When the <code>-d</code> option is not specified:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 106435<br>106436<br>106437                                                                                           |                          | • Each input character found in the array specified by <code>string1</code> shall be replaced by the character in the same relative position in the array specified by <code>string2</code> . When the array specified by <code>string2</code> is shorter than the one specified by <code>string1</code> , the results are unspecified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 106438<br>106439<br>106440<br>106441                                                                                 |                          | • If the <code>-C</code> option is specified, the complements of the characters specified by <code>string1</code> (the set of all characters in the current character set, as defined by the current setting of <code>LC_CTYPE</code> , except for those actually specified in the <code>string1</code> operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of <code>LC_COLLATE</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 106442<br>106443                                                                                                     |                          | • If the <code>-c</code> option is specified, the complement of the values specified by <code>string1</code> shall be placed in the array in ascending order by binary value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 106444<br>106445<br>106446<br>106447                                                                                 |                          | • Because the order in which characters specified by character class expressions or equivalence class expressions is undefined, such expressions should only be used if the intent is to map several characters into one. An exception is case conversion, as described previously.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 106448                                                                                                               |                          | When the <code>-d</code> option is specified:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 106449                                                                                                               |                          | • Input characters found in the array specified by <code>string1</code> shall be deleted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

- 106450 • When the `-C` option is specified with `-d`, all characters except those specified by *string1*
- 106451 shall be deleted. The contents of *string2* are ignored, unless the `-s` option is also specified.
- 106452 • When the `-c` option is specified with `-d`, all values except those specified by *string1* shall
- 106453 be deleted. The contents of *string2* shall be ignored, unless the `-s` option is also specified.
- 106454 • The same string cannot be used for both the `-d` and the `-s` option; when both options are
- 106455 specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be
- 106456 required.

106457 When the `-s` option is specified, after any deletions or translations have taken place, repeated

106458 sequences of the same character shall be replaced by one occurrence of the same character, if the

106459 character is found in the array specified by the last operand. If the last operand contains a

106460 character class, such as the following example:

```
106461 tr -s '[:space:]'
```

106462 the last operand's array shall contain all of the characters in that character class. However, in a

106463 case conversion, as described previously, such as:

```
106464 tr -s '[:upper:]' '[:lower:]'
```

106465 the last operand's array shall contain only those characters defined as the second characters in

106466 each of the **toupper** or **tolower** character pairs, as appropriate.

106467 An empty string used for *string1* or *string2* produces undefined results.

#### 106468 EXIT STATUS

106469 The following exit values shall be returned:

106470 0 All input was processed successfully.

106471 >0 An error occurred.

#### 106472 CONSEQUENCES OF ERRORS

106473 Default.

#### 106474 APPLICATION USAGE

106475 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

106476 If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must

106477 use the full three digits to avoid ambiguity.

106478 When *string2* is shorter than *string1*, a difference results between historical System V and BSD

106479 systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible

106480 to do the following:

```
106481 tr 0123456789 d
```

106482 which would translate all digits to the letter 'd'. Since this area is specifically unspecified in

106483 this volume of POSIX.1-200x, both the BSD and System V behaviors are allowed, but a

106484 conforming application cannot rely on the BSD behavior. It would have to code the example in

106485 the following way:

```
106486 tr 0123456789 '[d*]'
```

106487 It should be noted that, despite similarities in appearance, the string operands used by *tr* are not

106488 regular expressions.

106489 Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL

106490 characters in its input stream. NUL characters can be stripped by using:

```
106491 tr -d '\000'
```

## EXAMPLES

1. The following example creates a list of all words in **file1** one per line in **file2**, where a word is taken to be a maximal string of letters.

```
tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

2. The next example translates all lowercase characters in **file1** to uppercase and writes the results to standard output.

```
tr "[:lower:]" "[:upper:]" <file1
```

3. This example uses an equivalence class to identify accented variants of the base character 'e' in **file1**, which are stripped of diacritical marks and written to **file2**.

```
tr "[=e=]" "[e*]" <file1 >file2
```

## RATIONALE

In some early proposals, an explicit option `-n` was added to disable the historical behavior of stripping NUL characters from the input. It was considered that automatically stripping NUL characters from the input was not correct functionality. However, the removal of `-n` in a later proposal does not remove the requirement that `tr` correctly process NUL characters in its input stream. NUL characters can be stripped by using `tr -d '\000'`.

Historical implementations of `tr` differ widely in syntax and behavior. For example, the BSD version has not needed the bracket characters for the repetition sequence. The `tr` utility syntax is based more closely on the System V and XPG3 model while attempting to accommodate historical BSD implementations. In the case of the short `string2` padding, the decision was to unspecified the behavior and preserve System V and XPG3 scripts, which might find difficulty with the BSD method. The assumption was made that BSD users of `tr` have to make accommodations to meet the syntax defined here. Since it is possible to use the repetition sequence to duplicate the desired behavior, whereas there is no simple way to achieve the System V method, this was the correct, if not desirable, approach.

The use of octal values to specify control characters, while having historical precedents, is not portable. The introduction of escape sequences for control characters should provide the necessary portability. It is recognized that this may cause some historical scripts to break.

An early proposal included support for multi-character collating elements. It was pointed out that, while `tr` does employ some syntactical elements from REs, the aim of `tr` is quite different; ranges, for example, do not have a similar meaning ("any of the chars in the range matches", *versus* "translate each character in the range to the output counterpart"). As a result, the previously included support for multi-character collating elements has been removed. What remains are ranges in current collation order (to support, for example, accented characters), character classes, and equivalence classes.

In XPG3 the `[:class:]` and `[=equiv=]` conventions are shown with double brackets, as in RE syntax. However, `tr` does not implement RE principles; it just borrows part of the syntax. Consequently, `[:class:]` and `[=equiv=]` should be regarded as syntactical elements on a par with `[x*n]`, which is not an RE bracket expression.

The standard developers will consider changes to `tr` that allow it to translate characters between different character encodings, or they will consider providing a new utility to accomplish this.

On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
tr '[a-z]' '[A-Z]'
```

However, BSD-based systems did not require the brackets, and this convention is used here to avoid breaking large numbers of BSD scripts:

```
tr a-z A-Z
```

106538 The preceding System V script will continue to work because the brackets, treated as regular  
 106539 characters, are translated to themselves. However, any System V script that relied on "a-z"  
 106540 representing the three characters 'a', '-', and 'z' have to be rewritten as "az-".

106541 The ISO POSIX-2:1993 standard had a -c option that behaved similarly to the -C option, but did  
 106542 not supply functionality equivalent to the -c option specified in POSIX.1-200x. This meant that  
 106543 historical practice of being able to specify *tr -cd\000-\177* (which would delete all bytes with  
 106544 the top bit set) would have no effect because, in the C locale, bytes with the values octal 200 to  
 106545 octal 377 are not characters.

106546 The earlier version also said that octal sequences referred to collating elements and could be  
 106547 placed adjacent to each other to specify multi-byte characters. However, it was noted that this  
 106548 caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were  
 106549 intending to specify multi-byte characters or multiple single byte characters. POSIX.1-200x  
 106550 specifies that octal sequences always refer to single byte binary values when used to specify an  
 106551 endpoint of a range of collating elements.

106552 Earlier versions of this standard allowed for implementations with bytes other than eight bits, |  
 106553 but this has been modified in this version.

#### 106554 FUTURE DIRECTIONS

106555 None.

#### 106556 SEE ALSO

106557 *sed*

106558 XBD [Table 5-1](#) (on page 108), [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

#### 106559 CHANGE HISTORY

106560 First released in Issue 2.

##### 106561 Issue 6

106562 The -C operand is added, and the description of the -c operand is changed to align with the  
 106563 IEEE P1003.2b draft standard.

106564 The normative text is reworded to avoid use of the term "must" for application requirements.

106565 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/31 is applied, removing text describing  
 106566 behavior on systems with bytes consisting of more than eight bits.

106567 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/32 is applied, updating an example in the  
 106568 EXAMPLES section to avoid using unspecified behavior.

106569 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/33 is applied, making a correction to the  
 106570 RATIONALE.

##### 106571 Issue 7

106572 SD5-XCU-ERN-30 is applied.

106573 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

106574 **NAME**  
 106575       true — return true value

106576 **SYNOPSIS**  
 106577       true

106578 **DESCRIPTION**  
 106579       The *true* utility shall return with exit code zero.

106580 **OPTIONS**  
 106581       None.

106582 **OPERANDS**  
 106583       None.

106584 **STDIN**  
 106585       Not used.

106586 **INPUT FILES**  
 106587       None.

106588 **ENVIRONMENT VARIABLES**  
 106589       None.

106590 **ASYNCHRONOUS EVENTS**  
 106591       Default.

106592 **STDOUT**  
 106593       Not used.

106594 **STDERR**  
 106595       Not used.

106596 **OUTPUT FILES**  
 106597       None.

106598 **EXTENDED DESCRIPTION**  
 106599       None.

106600 **EXIT STATUS**  
 106601       Zero.

106602 **CONSEQUENCES OF ERRORS**  
 106603       None.

106604 **APPLICATION USAGE**  
 106605       This utility is typically used in shell scripts, as shown in the EXAMPLES section. The special  
 106606       built-in utility `:` is sometimes more efficient than *true*.

106607 **EXAMPLES**  
 106608       This command is executed forever:  
 106609       while true  
 106610       do  
 106611             command  
 106612       done

106613  
106614  
106615  
106616  
  
106617  
106618  
  
106619  
106620  
  
106621  
106622  
  
106623  
106624  
106625  
106626

**RATIONALE**

The *true* utility has been retained in this volume of POSIX.1-200x, even though the shell special built-in `:` provides similar functionality, because *true* is widely used in historical scripts and is less cryptic to novice script readers.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

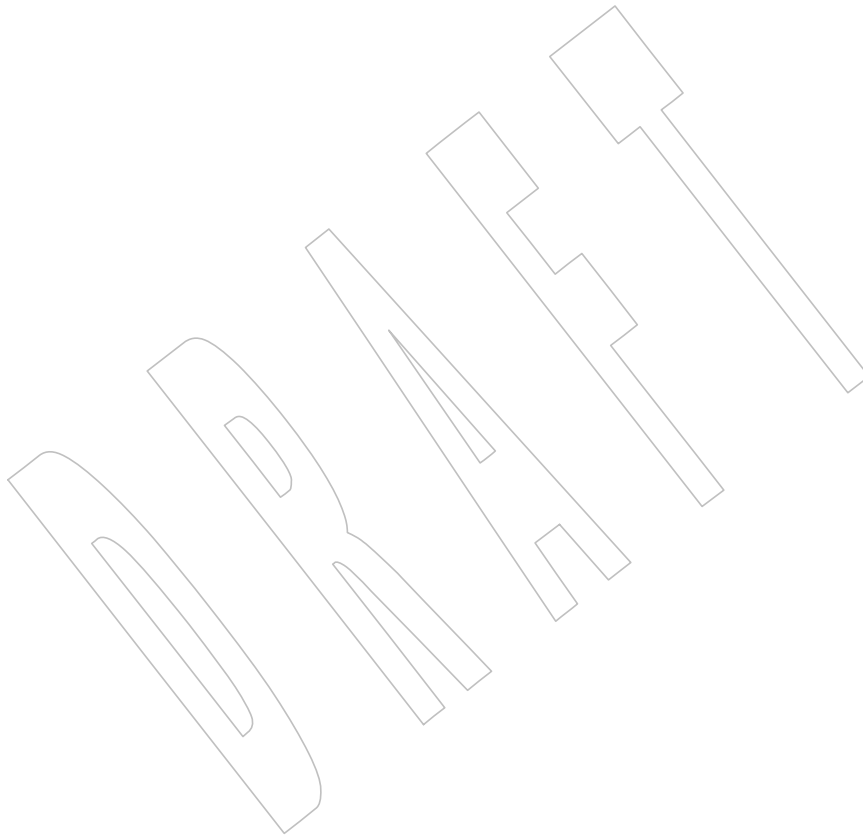
[Section 2.9](#) (on page 2263), *false*

**CHANGE HISTORY**

First released in Issue 2.

**Issue 6**

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/39 is applied, replacing the terms “None” and “Default” from the `STDERR` and `EXIT STATUS` sections, respectively, with terms as defined in [Section 1.4](#) (on page 2235).



106627 **NAME**106628 `tsort` — topological sort106629 **SYNOPSIS**106630 `tsort` [*file*]106631 **DESCRIPTION**106632 The *tsort* utility shall write to standard output a totally ordered list of items consistent with a  
106633 partial ordering of items contained in the input.106634 The application shall ensure that the input consists of pairs of items (non-empty strings)  
106635 separated by <blank>s. Pairs of different items indicate ordering. Pairs of identical items  
106636 indicate presence, but not ordering.106637 **OPTIONS**

106638 None.

106639 **OPERANDS**

106640 The following operand shall be supported:

106641 *file* A pathname of a text file to order. If no *file* operand is given, the standard input  
106642 shall be used.106643 **STDIN**106644 The standard input shall be a text file that is used if no *file* operand is given.106645 **INPUT FILES**106646 The input file named by the *file* operand is a text file.106647 **ENVIRONMENT VARIABLES**106648 The following environment variables shall affect the execution of *tsort*:106649 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
106650 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
106651 variables used to determine the values of locale categories.)106652 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
106653 internationalization variables.106654 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
106655 characters (for example, single-byte as opposed to multi-byte characters in  
106656 arguments and input files).106657 *LC\_MESSAGES*106658 Determine the locale that should be used to affect the format and contents of  
106659 diagnostic messages written to standard error.106660 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.106661 **ASYNCHRONOUS EVENTS**

106662 Default.

106663 **STDOUT**106664 The standard output shall be a text file consisting of the order list produced from the partially  
106665 ordered input.



106666 **STDERR**

106667 The standard error shall be used only for diagnostic messages.

106668 **OUTPUT FILES**

106669 None.

106670 **EXTENDED DESCRIPTION**

106671 None.

106672 **EXIT STATUS**

106673 The following exit values shall be returned:

106674 0 Successful completion.

106675 &gt;0 An error occurred.

106676 **CONSEQUENCES OF ERRORS**

106677 Default.

106678 **APPLICATION USAGE**106679 The *LC\_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not  
106680 lexicographic, but depends on the pairs of items given as input.106681 **EXAMPLES**

106682 The command:

106683 `tsort <<EOF`  
106684 `a b c c d e`  
106685 `g g`  
106686 `f g e f`  
106687 `h h`  
106688 `EOF`

106689 produces the output:

106690 `a`  
106691 `b`  
106692 `c`  
106693 `d`  
106694 `e`  
106695 `f`  
106696 `g`  
106697 `h`106698 **RATIONALE**

106699 None.

106700 **FUTURE DIRECTIONS**

106701 None.

106702 **SEE ALSO**106703 XBD [Chapter 8](#) (on page 159)106704 **CHANGE HISTORY**

106705 First released in Issue 2.

106706 **Issue 6**

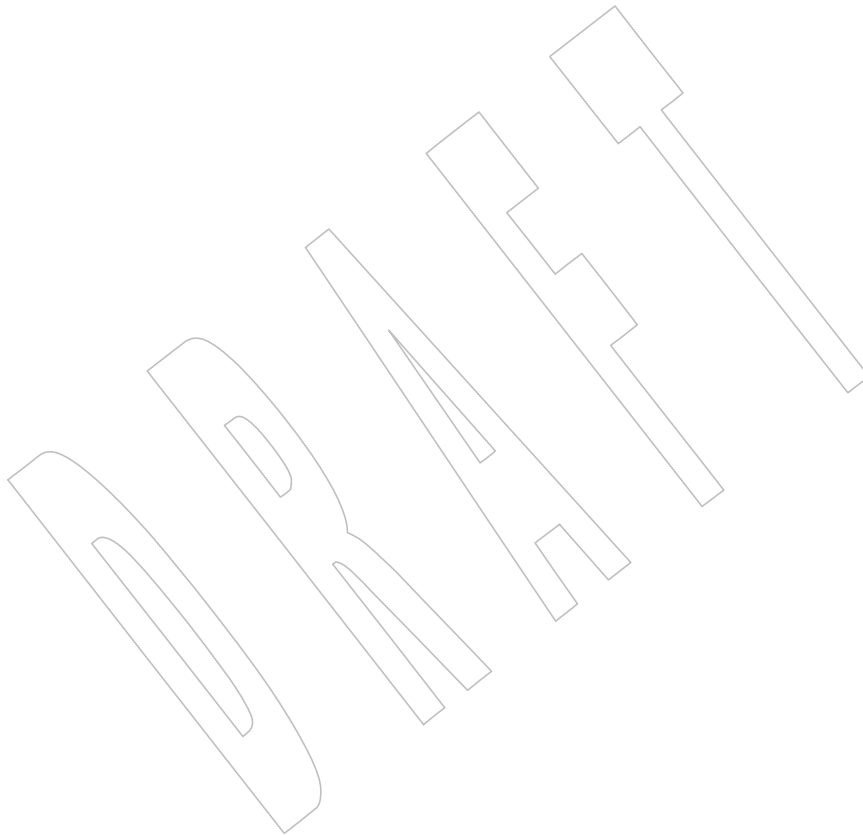
106707 The normative text is reworded to avoid use of the term “must” for application requirements.

106708

**Issue 7**

106709

The *tsort* utility is moved from the XSI option to the Base.



106710 **NAME**

106711 tty — return user's terminal name

106712 **SYNOPSIS**

106713 tty

106714 **DESCRIPTION**

106715 The *tty* utility shall write to the standard output the name of the terminal that is open as  
 106716 standard input. The name that is used shall be equivalent to the string that would be returned by  
 106717 the *ttyname()* function defined in the System Interfaces volume of POSIX.1-200x.

106718 **OPTIONS**106719 The *tty* utility shall conform to XBD [Section 12.2](#) (on page 201).106720 **OPERANDS**

106721 None.

106722 **STDIN**

106723 While no input is read from standard input, standard input shall be examined to determine  
 106724 whether or not it is a terminal, and, if so, to determine the name of the terminal.

106725 **INPUT FILES**

106726 None.

106727 **ENVIRONMENT VARIABLES**106728 The following environment variables shall affect the execution of *tty*:

106729 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 106730 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 106731 variables used to determine the values of locale categories.)

106732 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 106733 internationalization variables.

106734 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 106735 characters (for example, single-byte as opposed to multi-byte characters in  
 106736 arguments).

106737 **LC\_MESSAGES**

106738 Determine the locale that should be used to affect the format and contents of  
 106739 diagnostic messages written to standard error and informative messages written to  
 106740 standard output.

106741 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

106742 **ASYNCHRONOUS EVENTS**

106743 Default.

106744 **STDOUT**

106745 If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname()*  
 106746 function defined in the System Interfaces volume of POSIX.1-200x shall be written in the  
 106747 following format:

106748 "%s\n", &lt;terminal name&gt;

106749 Otherwise, a message shall be written indicating that standard input is not connected to a  
 106750 terminal. In the POSIX locale, the *tty* utility shall use the format:

106751 "not a tty\n"

106752 **STDERR**

106753 The standard error shall be used only for diagnostic messages.

106754 **OUTPUT FILES**

106755 None.

106756 **EXTENDED DESCRIPTION**

106757 None.

106758 **EXIT STATUS**

106759 The following exit values shall be returned:

106760 0 Standard input is a terminal.

106761 1 Standard input is not a terminal.

106762 &gt;1 An error occurred.

106763 **CONSEQUENCES OF ERRORS**

106764 Default.

106765 **APPLICATION USAGE**

106766 This utility checks the status of the file open as standard input against that of an  
 106767 implementation-defined set of files. It is possible that no match can be found, or that the match  
 106768 found need not be the same file as that which was opened for standard input (although they are  
 106769 the same device).

106770 **EXAMPLES**

106771 None.

106772 **RATIONALE**

106773 None.

106774 **FUTURE DIRECTIONS**

106775 None.

106776 **SEE ALSO**106777 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)106778 XSH [isatty\(\)](#), [ttyname\(\)](#)106779 **CHANGE HISTORY**

106780 First released in Issue 2.

106781 **Issue 5**

106782 The SYNOPSIS is changed to indicate two forms of the command, with the second form marked  
 106783 as obsolete. This is a clarification and does not change the functionality published in previous  
 106784 issues.

106785 **Issue 6**106786 The obsolescent `-s` option is removed.

106787 **NAME**

106788           type — write a description of command type

106789 **SYNOPSIS**

106790 XSI        type name...

106791 **DESCRIPTION**106792       The *type* utility shall indicate how each argument would be interpreted if used as a command  
106793       name.106794 **OPTIONS**

106795       None.

106796 **OPERANDS**

106797       The following operand shall be supported:

106798       *name*        A name to be interpreted.106799 **STDIN**

106800       Not used.

106801 **INPUT FILES**

106802       None.

106803 **ENVIRONMENT VARIABLES**106804       The following environment variables shall affect the execution of *type*:106805       *LANG*        Provide a default value for the internationalization variables that are unset or null. |  
106806                    (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
106807                    variables used to determine the values of locale categories.)106808       *LC\_ALL*       If set to a non-empty string value, override the values of all the other  
106809                    internationalization variables.106810       *LC\_CTYPE*   Determine the locale for the interpretation of sequences of bytes of text data as  
106811                    characters (for example, single-byte as opposed to multi-byte characters in  
106812                    arguments).106813       *LC\_MESSAGES*106814                    Determine the locale that should be used to affect the format and contents of  
106815                    diagnostic messages written to standard error.106816       *NLSPATH*     Determine the location of message catalogs for the processing of *LC\_MESSAGES*.106817       *PATH*        Determine the location of *name*, as described in XBD [Chapter 8](#) (on page 159). |106818 **ASYNCHRONOUS EVENTS**

106819       Default.

106820 **STDOUT**106821       The standard output of *type* contains information about each operand in an unspecified format.  
106822       The information provided typically identifies the operand as a shell built-in, function, alias, or  
106823       keyword, and where applicable, may display the operand's pathname.106824 **STDERR**

106825       The standard error shall be used only for diagnostic messages.

106826 **OUTPUT FILES**

106827 None.

106828 **EXTENDED DESCRIPTION**

106829 None.

106830 **EXIT STATUS**

106831 The following exit values shall be returned:

106832 0 Successful completion.

106833 &gt;0 An error occurred.

106834 **CONSEQUENCES OF ERRORS**

106835 Default.

106836 **APPLICATION USAGE**

106837 Since *type* must be aware of the contents of the current shell execution environment (such as the  
 106838 lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell  
 106839 regular built-in. If it is called in a separate utility execution environment, such as one of the  
 106840 following:

106841 `nohup type writer`106842 `find . -type f | xargs type`

106843 it might not produce accurate results.

106844 **EXAMPLES**

106845 None.

106846 **RATIONALE**

106847 None.

106848 **FUTURE DIRECTIONS**

106849 None.

106850 **SEE ALSO**106851 *command*, *hash*106852 XBD [Chapter 8](#) (on page 159)106853 **CHANGE HISTORY**

106854 First released in Issue 2.

106855 **NAME**106856 `ulimit` — set or report file size limit106857 **SYNOPSIS**106858 XSI `ulimit [-f] [blocks]`106859 **DESCRIPTION**106860 The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the  
106861 shell and its child processes (files of any size may be read). Only a process with appropriate  
106862 privileges can increase the limit.106863 **OPTIONS**106864 The *ulimit* utility shall conform to XBD [Section 12.2](#) (on page 201). |

106865 The following option shall be supported:

106866 `-f` Set (or report, if no *blocks* operand is present), the file size limit in blocks. The `-f`  
106867 option shall also be the default case.106868 **OPERANDS**

106869 The following operand shall be supported:

106870 *blocks* The number of 512-byte blocks to use as the new file size limit.106871 **STDIN**

106872 Not used.

106873 **INPUT FILES**

106874 None.

106875 **ENVIRONMENT VARIABLES**106876 The following environment variables shall affect the execution of *ulimit*:106877 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
106878 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
106879 variables used to determine the values of locale categories.)106880 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
106881 internationalization variables.106882 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
106883 characters (for example, single-byte as opposed to multi-byte characters in  
106884 arguments).106885 *LC\_MESSAGES*  
106886 Determine the locale that should be used to affect the format and contents of  
106887 diagnostic messages written to standard error.106888 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.106889 **ASYNCHRONOUS EVENTS**

106890 Default.

106891 **STDOUT**106892 The standard output shall be used when no *blocks* operand is present. If the current number of  
106893 blocks is limited, the number of blocks in the current limit shall be written in the following  
106894 format:106895 `"%d\n", <number of 512-byte blocks>`

106896 If there is no current limit on the number of blocks, in the POSIX locale the following format  
106897 shall be used:

106898 "unlimited\n"

#### 106899 **STDERR**

106900 The standard error shall be used only for diagnostic messages.

#### 106901 **OUTPUT FILES**

106902 None.

#### 106903 **EXTENDED DESCRIPTION**

106904 None.

#### 106905 **EXIT STATUS**

106906 The following exit values shall be returned:

106907 0 Successful completion.

106908 >0 A request for a higher limit was rejected or an error occurred.

#### 106909 **CONSEQUENCES OF ERRORS**

106910 Default.

#### 106911 **APPLICATION USAGE**

106912 Since *ulimit* affects the current shell execution environment, it is always provided as a shell  
106913 regular built-in. If it is called in a separate utility execution environment, such as one of the  
106914 following:

```
106915 nohup ulimit -f 10000
106916 env ulimit 10000
```

106917 it does not affect the file size limit of the caller's environment.

106918 Once a limit has been decreased by a process, it cannot be increased (unless appropriate  
106919 privileges are involved), even back to the original system limit.

#### 106920 **EXAMPLES**

106921 Set the file size limit to 51 200 bytes:

```
106922 ulimit -f 100
```

#### 106923 **RATIONALE**

106924 None.

#### 106925 **FUTURE DIRECTIONS**

106926 None.

#### 106927 **SEE ALSO**

106928 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

106929 XSH [ulimit](#)

#### 106930 **CHANGE HISTORY**

106931 First released in Issue 2.

#### 106932 **Issue 7**

106933 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



106934 **NAME**  
 106935 `umask` — get or set the file mode creation mask

106936 **SYNOPSIS**  
 106937 `umask [-S] [mask]`

106938 **DESCRIPTION**  
 106939 The *umask* utility shall set the file mode creation mask of the current shell execution environment  
 106940 (see [Section 2.12](#), on page 2277) to the value specified by the *mask* operand. This mask shall affect  
 106941 the initial value of the file permission bits of subsequently created files. If *umask* is called in a  
 106942 subshell or separate utility execution environment, such as one of the following:

```
106943 (umask 002)
106944 nohup umask ...
106945 find . -exec umask ... \;
```

106946 it shall not affect the file mode creation mask of the caller's environment.

106947 If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of  
 106948 the file mode creation mask of the invoking process.

106949 **OPTIONS**  
 106950 The *umask* utility shall conform to XBD [Section 12.2](#) (on page 201).

106951 The following option shall be supported:

106952 **-S** Produce symbolic output.

106953 The default output style is unspecified, but shall be recognized on a subsequent invocation of  
 106954 *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

106955 **OPERANDS**  
 106956 The following operand shall be supported:

106957 *mask* A string specifying the new file mode creation mask. The string is treated in the  
 106958 same way as the *mode* operand described in the EXTENDED DESCRIPTION  
 106959 section for *chmod*.

106960 For a *symbolic\_mode* value, the new value of the file mode creation mask shall be  
 106961 the logical complement of the file permission bits portion of the file mode specified  
 106962 by the *symbolic\_mode* string.

106963 In a *symbolic\_mode* value, the permissions *op* characters '+' and '-' shall be  
 106964 interpreted relative to the current file mode creation mask; '+' shall cause the bits  
 106965 for the indicated permissions to be cleared in the mask; '-' shall cause the bits for  
 106966 the indicated permissions to be set in the mask.

106967 The interpretation of *mode* values that specify file mode bits other than the file  
 106968 permission bits is unspecified.

106969 In the octal integer form of *mode*, the specified bits are set in the file mode creation  
 106970 mask.

106971 The file mode creation mask shall be set to the resulting numeric value.

106972 The default output of a prior invocation of *umask* on the same system with no  
 106973 operand also shall be recognized as a *mask* operand.

**umask**

Utilities

106974 **STDIN**

106975 Not used.

106976 **INPUT FILES**

106977 None.

106978 **ENVIRONMENT VARIABLES**106979 The following environment variables shall affect the execution of *umask*:

106980 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
 106981 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 106982 variables used to determine the values of locale categories.)

106983 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 106984 internationalization variables.

106985 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 106986 characters (for example, single-byte as opposed to multi-byte characters in  
 106987 arguments).

106988 *LC\_MESSAGES*  
 106989 Determine the locale that should be used to affect the format and contents of  
 106990 diagnostic messages written to standard error.

106991 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

106992 **ASYNCHRONOUS EVENTS**

106993 Default.

106994 **STDOUT**

106995 When the *mask* operand is not specified, the *umask* utility shall write a message to standard  
 106996 output that can later be used as a *umask mask* operand.

106997 If **-S** is specified, the message shall be in the following format:

106998 "u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,  
 106999 <other permissions>

107000 where the three values shall be combinations of letters from the set {*r*, *w*, *x*}; the presence of a  
 107001 letter shall indicate that the corresponding bit is clear in the file mode creation mask.

107002 If a *mask* operand is specified, there shall be no output written to standard output.

107003 **STDERR**

107004 The standard error shall be used only for diagnostic messages.

107005 **OUTPUT FILES**

107006 None.

107007 **EXTENDED DESCRIPTION**

107008 None.

107009 **EXIT STATUS**

107010 The following exit values shall be returned:

107011 0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

107012 >0 An error occurred.

107013 **CONSEQUENCES OF ERRORS**

107014 Default.

**APPLICATION USAGE**

Since *umask* affects the current shell execution environment, it is generally provided as a shell regular built-in.

In contrast to the negative permission logic provided by the file mode creation mask and the octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those permissions that are left alone.

**EXAMPLES**

Either of the commands:

```
umask a=rx,ug+w
```

```
umask 002
```

sets the mode mask so that subsequently created files have their S\_IWOTH bit cleared.

After setting the mode mask with either of the above commands, the *umask* command can be used to write out the current value of the mode mask:

```
$ umask
0002
```

(The output format is unspecified, but historical implementations use the octal integer mode format.)

```
$ umask -S
u=rwx,g=rwx,o=rx
```

Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask* utility.

Assuming the mode mask is set as above, the command:

```
umask g-w
```

sets the mode mask so that subsequently created files have their S\_IWGRP and S\_IWOTH bits cleared.

The command:

```
umask -- -w
```

sets the mode mask so that subsequently created files have all their write bits cleared. Note that *mask* operands *-r*, *-w*, *-x* or anything beginning with a hyphen, must be preceded by "--" to keep it from being interpreted as an option.

**RATIONALE**

Since *umask* affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(umask 002)
nohup umask ...
find . -exec umask ... \;
```

it does not affect the file mode creation mask of the environment of the caller.

The description of the historical utility was modified to allow it to use the symbolic modes of *chmod*. The *-s* option used in early proposals was changed to *-S* because *-s* could be confused with a *symbolic\_mode* form of mask referring to the S\_ISUID and S\_ISGID bits.

The default output style is unspecified to permit implementors to provide migration to the new symbolic style at the time most appropriate to their users. A *-o* flag to force octal mode output was omitted because the octal mode may not be sufficient to specify all of the information that

107059 may be present in the file mode creation mask when more secure file access permission checks  
107060 are implemented.

107061 It has been suggested that trusted systems developers might appreciate ameliorating the  
107062 requirement that the mode mask “affects” the file access permissions, since it seems access  
107063 control lists might replace the mode mask to some degree. The wording has been changed to say  
107064 that it affects the file permission bits, and it leaves the details of the behavior of how they affect  
107065 the file access permissions to the description in the System Interfaces volume of POSIX.1-200x.

#### 107066 FUTURE DIRECTIONS

107067 None.

#### 107068 SEE ALSO

107069 [Chapter 2](#) (on page 2245), *chmod*

107070 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

107071 XSH *umask*

#### 107072 CHANGE HISTORY

107073 First released in Issue 2.

#### 107074 Issue 6

107075 The following new requirements on POSIX implementations derive from alignment with the  
107076 Single UNIX Specification:

- 107077 • The octal mode is supported.

107078 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/34 is applied, making a correction to the  
107079 RATIONALE.

#### 107080 Issue 7

107081 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107082 **NAME**  
 107083 unalias — remove alias definitions

107084 **SYNOPSIS**  
 107085 unalias *alias-name*...  
 107086 unalias -a

107087 **DESCRIPTION**  
 107088 The *unalias* utility shall remove the definition for each alias name specified. See [Section 2.3.1](#) (on  
 107089 page 2248). The aliases shall be removed from the current shell execution environment; see  
 107090 [Section 2.12](#) (on page 2277).

107091 **OPTIONS**  
 107092 The *unalias* utility shall conform to XBD [Section 12.2](#) (on page 201).  
 107093 The following option shall be supported:  
 107094 **-a** Remove all alias definitions from the current shell execution environment.

107095 **OPERANDS**  
 107096 The following operand shall be supported:  
 107097 *alias-name* The name of an alias to be removed.

107098 **STDIN**  
 107099 Not used.

107100 **INPUT FILES**  
 107101 None.

107102 **ENVIRONMENT VARIABLES**  
 107103 The following environment variables shall affect the execution of *unalias*:  
 107104 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 107105 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 107106 variables used to determine the values of locale categories.)  
 107107 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 107108 internationalization variables.  
 107109 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 107110 characters (for example, single-byte as opposed to multi-byte characters in  
 107111 arguments).  
 107112 **LC\_MESSAGES**  
 107113 Determine the locale that should be used to affect the format and contents of  
 107114 diagnostic messages written to standard error.  
 107115 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

107116 **ASYNCHRONOUS EVENTS**  
 107117 Default.

107118 **STDOUT**  
 107119 Not used.

- 107120 **STDERR**
- 107121 The standard error shall be used only for diagnostic messages.
- 107122 **OUTPUT FILES**
- 107123 None.
- 107124 **EXTENDED DESCRIPTION**
- 107125 None.
- 107126 **EXIT STATUS**
- 107127 The following exit values shall be returned:
- 107128 0 Successful completion.
- 107129 >0 One of the *alias-name* operands specified did not represent a valid alias definition, or an
- 107130 error occurred.
- 107131 **CONSEQUENCES OF ERRORS**
- 107132 Default.
- 107133 **APPLICATION USAGE**
- 107134 Since *unalias* affects the current shell execution environment, it is generally provided as a shell
- 107135 regular built-in.
- 107136 **EXAMPLES**
- 107137 None.
- 107138 **RATIONALE**
- 107139 The *unalias* description is based on that from historical KornShell implementations. Known
- 107140 differences exist between that and the C shell. The KornShell version was adopted to be
- 107141 consistent with all the other KornShell features in this volume of POSIX.1-200x, such as
- 107142 command line editing.
- 107143 The **-a** option is the equivalent of the *unalias \** form of the C shell and is provided to address
- 107144 security concerns about unknown aliases entering the environment of a user (or application)
- 107145 through the allowable implementation-defined predefined alias route or as a result of an *ENV*
- 107146 file. (Although *unalias* could be used to simplify the “secure” shell script shown in the *command*
- 107147 rationale, it does not obviate the need to quote all command names. An initial call to *unalias -a*
- 107148 would have to be quoted in case there was an alias for *unalias*.)
- 107149 **FUTURE DIRECTIONS**
- 107150 None.
- 107151 **SEE ALSO**
- 107152 [Chapter 2](#) (on page 2245), [alias](#)
- 107153 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +
- 107154 **CHANGE HISTORY**
- 107155 First released in Issue 4.
- 107156 **Issue 6**
- 107157 This utility is marked as part of the User Portability Utilities option.
- 107158 **Issue 7**
- 107159 The *unalias* utility is moved from the User Portability Utilities option to the Base. User
- 107160 Portability Utilities is now an option for interactive utilities.

107161 **NAME**  
 107162        *uname* — return system name

107163 **SYNOPSIS**  
 107164        *uname* [-amnrsv]

107165 **DESCRIPTION**  
 107166        By default, the *uname* utility shall write the operating system name to standard output. When  
 107167        options are specified, symbols representing one or more system characteristics shall be written to  
 107168        the standard output. The format and contents of the symbols are implementation-defined. On  
 107169        systems conforming to the System Interfaces volume of POSIX.1-200x, the symbols written shall  
 107170        be those supported by the *uname()* function as defined in the System Interfaces volume of  
 107171        POSIX.1-200x.

107172 **OPTIONS**  
 107173        The *uname* utility shall conform to XBD [Section 12.2](#) (on page 201).

107174        The following options shall be supported:

- 107175        **-a**        Behave as though all of the options **-mnrsv** were specified.
- 107176        **-m**        Write the name of the hardware type on which the system is running to standard  
 107177        output.
- 107178        **-n**        Write the name of this node within an implementation-defined communications  
 107179        network.
- 107180        **-r**        Write the current release level of the operating system implementation.
- 107181        **-s**        Write the name of the implementation of the operating system.
- 107182        **-v**        Write the current version level of this release of the operating system  
 107183        implementation.

107184        If no options are specified, the *uname* utility shall write the operating system name, as if the **-s**  
 107185        option had been specified.

107186 **OPERANDS**  
 107187        None.

107188 **STDIN**  
 107189        Not used.

107190 **INPUT FILES**  
 107191        None.

107192 **ENVIRONMENT VARIABLES**  
 107193        The following environment variables shall affect the execution of *uname*:

- 107194        **LANG**        Provide a default value for the internationalization variables that are unset or null.  
 107195        (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 107196        variables used to determine the values of locale categories.)
- 107197        **LC\_ALL**       If set to a non-empty string value, override the values of all the other  
 107198        internationalization variables.
- 107199        **LC\_CTYPE**     Determine the locale for the interpretation of sequences of bytes of text data as  
 107200        characters (for example, single-byte as opposed to multi-byte characters in  
 107201        arguments).

107202 *LC\_MESSAGES*  
 107203 Determine the locale that should be used to affect the format and contents of  
 107204 diagnostic messages written to standard error.

107205 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

**ASYNCHRONOUS EVENTS**

107206 Default.  
 107207

**STDOUT**

107208 By default, the output shall be a single line of the following form:

107209 "*%s\n*", *<sysname>*  
 107210

107211 If the *-a* option is specified, the output shall be a single line of the following form:

107212 "*%s %s %s %s %s\n*", *<sysname>*, *<nodename>*, *<release>*,  
 107213 *<version>*, *<machine>*

107214 Additional implementation-defined symbols may be written; all such symbols shall be written at  
 107215 the end of the line of output before the *<newline>*.

107216 If options are specified to select different combinations of the symbols, only those symbols shall  
 107217 be written, in the order shown above for the *-a* option. If a symbol is not selected for writing, its  
 107218 corresponding trailing *<blank>*s also shall not be written.

**STDERR**

107219 The standard error shall be used only for diagnostic messages.  
 107220

**OUTPUT FILES**

107221 None.  
 107222

**EXTENDED DESCRIPTION**

107223 None.  
 107224

**EXIT STATUS**

107225 The following exit values shall be returned:  
 107226

107227 0 The requested information was successfully written.

107228 >0 An error occurred.

**CONSEQUENCES OF ERRORS**

107229 Default.  
 107230

**APPLICATION USAGE**

107231 Note that any of the symbols could include embedded *<space>*s, which may affect parsing  
 107232 algorithms if multiple options are selected for output.  
 107233

107234 The node name is typically a name that the system uses to identify itself for inter-system  
 107235 communication addressing.

**EXAMPLES**

107236 The following command:  
 107237

107238 *uname -sr*

107239 writes the operating system name and release level, separated by one or more *<blank>*s.

**RATIONALE**

107240 It was suggested that this utility cannot be used portably since the format of the symbols is  
 107241 implementation-defined. The POSIX.1 working group could not achieve consensus on defining  
 107242 these formats in the underlying *uname()* function, and there was no expectation that this volume  
 107243 of POSIX.1-200x would be any more successful. Some applications may still find this historical  
 107244 utility of value. For example, the symbols could be used for system log entries or for comparison  
 107245



107246 with operator or user input.

107247 **FUTURE DIRECTIONS**

107248 None.

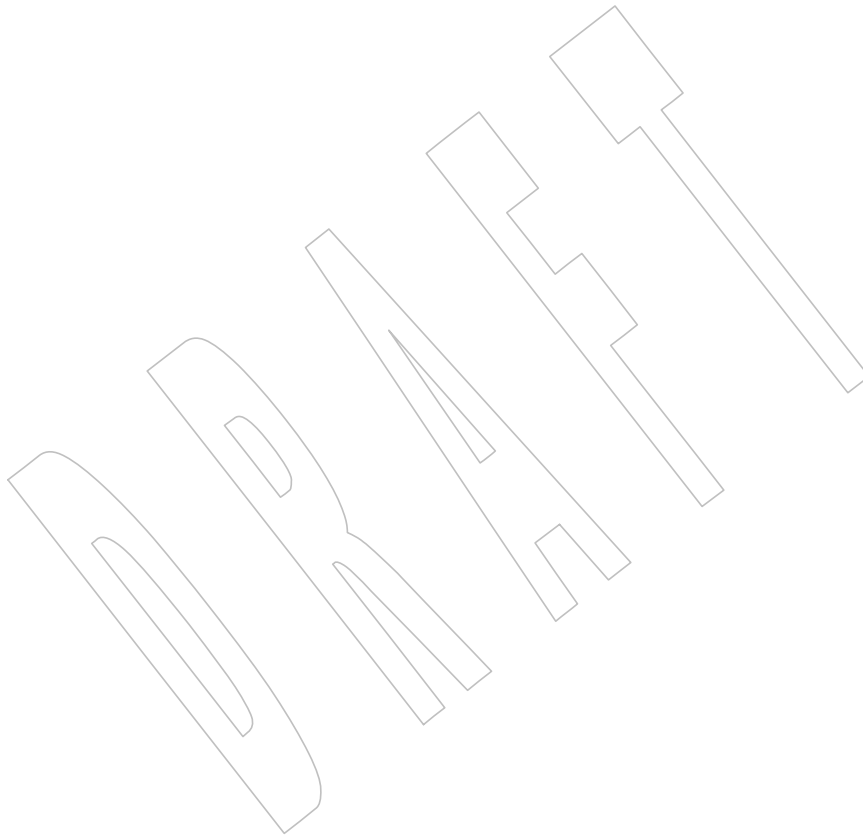
107249 **SEE ALSO**

107250 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

107251 XSH [uname](#)

107252 **CHANGE HISTORY**

107253 First released in Issue 2.



107254 **NAME**

107255 uncompress — expand compressed data

107256 **SYNOPSIS**107257 XSI uncompress [-cfv] [*file...*]107258 **DESCRIPTION**

107259 The *uncompress* utility shall restore files to their original state after they have been compressed  
 107260 using the *compress* utility. If no files are specified, the standard input shall be uncompressed to  
 107261 the standard output. If the invoking process has appropriate privileges, the ownership, modes,  
 107262 access time, and modification time of the original file shall be preserved.

107263 This utility shall support the uncompressing of any files produced by the *compress* utility on the  
 107264 same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to  
 107265 14-bit compression (see *compress*, **-b**); it is implementation-defined whether values of **-b** greater  
 107266 than 14 are supported.

107267 **OPTIONS**

107268 The *uncompress* utility shall conform to XBD [Section 12.2](#) (on page 201), except that Guideline 1  
 107269 does apply since the utility name has ten letters.

107270 The following options shall be supported:

- 107271 **-c** Write to standard output; no files are changed.
- 107272 **-f** Do not prompt for overwriting files. Except when run in the background, if **-f** is  
 107273 not given the user shall be prompted as to whether an existing file should be  
 107274 overwritten. If the standard input is not a terminal and **-f** is not given, *uncompress*  
 107275 shall write a diagnostic message to standard error and exit with a status greater  
 107276 than zero.
- 107277 **-v** Write messages to standard error concerning the expansion of each file.

107278 **OPERANDS**

107279 The following operand shall be supported:

- 107280 *file* A pathname of a file. If *file* already has the **.Z** suffix specified, it shall be used as the  
 107281 input file and the output file shall be named **file** with the **.Z** suffix removed.  
 107282 Otherwise, *file* shall be used as the name of the output file and **file** with the **.Z**  
 107283 suffix appended shall be used as the input file.

107284 **STDIN**

107285 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is **'-'**.

107286 **INPUT FILES**

107287 Input files shall be in the format produced by the *compress* utility.

107288 **ENVIRONMENT VARIABLES**

107289 The following environment variables shall affect the execution of *uncompress*:

- 107290 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 107291 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 107292 variables used to determine the values of locale categories.)
- 107293 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 107294 internationalization variables.

107295 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 107296 characters (for example, single-byte as opposed to multi-byte characters in  
 107297 arguments).

107298 **LC\_MESSAGES**  
 107299 Determine the locale that should be used to affect the format and contents of  
 107300 diagnostic messages written to standard error.

107301 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 107302 **ASYNCHRONOUS EVENTS**

107303 Default.

#### 107304 **STDOUT**

107305 When there are no *file* operands or the `-c` option is specified, the uncompressed output is written  
 107306 to standard output.

#### 107307 **STDERR**

107308 Prompts shall be written to the standard error output under the conditions specified in the  
 107309 DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their  
 107310 format is otherwise unspecified. Otherwise, the standard error output shall be used only for  
 107311 diagnostic messages.

#### 107312 **OUTPUT FILES**

107313 Output files are the same as the respective input files to *compress*.

#### 107314 **EXTENDED DESCRIPTION**

107315 None.

#### 107316 **EXIT STATUS**

107317 The following exit values shall be returned:

107318 0 Successful completion.

107319 >0 An error occurred.

#### 107320 **CONSEQUENCES OF ERRORS**

107321 The input file remains unmodified.

#### 107322 **APPLICATION USAGE**

107323 The limit of 14 on the *compress* `-b bits` argument is to achieve portability to all systems (within  
 107324 the restrictions imposed by the lack of an explicit published file format). Some implementations  
 107325 based on 16-bit architectures cannot support 15 or 16-bit uncompression.

#### 107326 **EXAMPLES**

107327 None.

#### 107328 **RATIONALE**

107329 None.

#### 107330 **FUTURE DIRECTIONS**

107331 None.

#### 107332 **SEE ALSO**

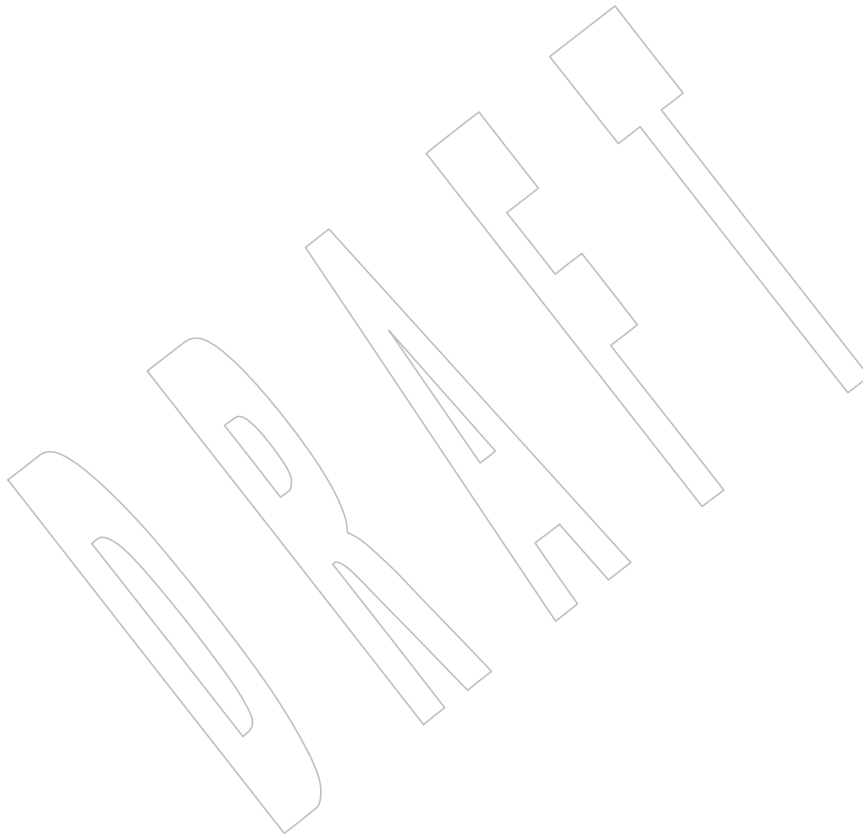
107333 *compress*, *zcat*

107334 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

#### 107335 **CHANGE HISTORY**

107336 First released in Issue 4.

- 107337 **Issue 6**
- 107338 The normative text is reworded to avoid use of the term “must” for application requirements.
- 107339 **Issue 7**
- 107340 SD5-XCU-ERN-26 is applied, clarifying that this utility is allowed to break the Utility Syntax
- 107341 Guidelines by having ten letters in its name.
- 107342 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



107343 **NAME**  
 107344 unexpand — convert spaces to tabs

107345 **SYNOPSIS**  
 107346 unexpand [-a|-t *tablist*] [*file...*]

107347 **DESCRIPTION**  
 107348 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>s  
 107349 at the beginning of each line into the maximum number of <tab>s followed by the minimum  
 107350 number of <space>s needed to fill the same column positions originally filled by the translated  
 107351 <blank>s. By default, tabstops shall be set at every eighth column position. Each <backspace>  
 107352 shall be copied to the output, and shall cause the column position count for tab calculations to be  
 107353 decremented; the count shall never be decremented to a value less than one.

107354 **OPTIONS**  
 107355 The *unexpand* utility shall conform to XBD Section 12.2 (on page 201).

107356 The following options shall be supported:

107357 **-a** In addition to translating <blank>s at the beginning of each line, translate all  
 107358 sequences of two or more <blank>s immediately preceding a tab stop to the  
 107359 maximum number of <tab>s followed by the minimum number of <space>s  
 107360 needed to fill the same column positions originally filled by the translated  
 107361 <blank>s.

107362 **-t *tablist*** Specify the tab stops. The application shall ensure that the *tablist* option-argument  
 107363 is a single argument consisting of a single positive decimal integer or multiple  
 107364 positive decimal integers, separated by <blank>s or commas, in ascending order. If  
 107365 a single number is given, tabs shall be set *tablist* column positions apart instead of  
 107366 the default 8. If multiple numbers are given, the tabs shall be set at those specific  
 107367 column positions.

107368 The application shall ensure that each tab-stop position *N* is an integer value  
 107369 greater than zero, and the list shall be in strictly ascending order. This is taken to  
 107370 mean that, from the start of a line of output, tabbing to position *N* shall cause the  
 107371 next character output to be in the (*N*+1)th column position on that line. When the  
 107372 **-t** option is not specified, the default shall be the equivalent of specifying **-t 8**  
 107373 (except for the interaction with **-a**, described below).

107374 No <space>-to-<tab> conversions shall occur for characters at positions beyond  
 107375 the last of those specified in a multiple tab-stop list.

107376 When **-t** is specified, the presence or absence of the **-a** option shall be ignored;  
 107377 conversion shall not be limited to the processing of leading <blank>s.

107378 **OPERANDS**  
 107379 The following operand shall be supported:

107380 *file* A pathname of a text file to be used as input.

107381 **STDIN**  
 107382 See the INPUT FILES section.

107383 **INPUT FILES**  
 107384 The input files shall be text files.

107385 **ENVIRONMENT VARIABLES**107386 The following environment variables shall affect the execution of *unexpand*:

107387 **LANG** Provide a default value for the internationalization variables that are unset or null. |  
 107388 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
 107389 variables used to determine the values of locale categories.)

107390 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 107391 internationalization variables.

107392 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 107393 characters (for example, single-byte as opposed to multi-byte characters in  
 107394 arguments and input files), the processing of <tab>s and <space>s, and for the  
 107395 determination of the width in column positions each character would occupy on  
 107396 an output device.

107397 **LC\_MESSAGES**  
 107398 Determine the locale that should be used to affect the format and contents of  
 107399 diagnostic messages written to standard error.

107400 **XSI** **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

107401 **ASYNCHRONOUS EVENTS**

107402 Default.

107403 **STDOUT**

107404 The standard output shall be equivalent to the input files with the specified <space>-to-<tab>  
 107405 conversions.

107406 **STDERR**

107407 The standard error shall be used only for diagnostic messages.

107408 **OUTPUT FILES**

107409 None.

107410 **EXTENDED DESCRIPTION**

107411 None.

107412 **EXIT STATUS**

107413 The following exit values shall be returned:

107414 0 Successful completion.

107415 &gt;0 An error occurred.

107416 **CONSEQUENCES OF ERRORS**

107417 Default.

107418 **APPLICATION USAGE**

107419 One non-intuitive aspect of *unexpand* is its restriction to leading spaces when neither **-a** nor **-t** is  
 107420 specified. Users who always want to convert all spaces in a file can easily alias *unexpand* to use  
 107421 the **-a** or **-t 8** option.

107422 **EXAMPLES**

107423 None.

107424 **RATIONALE**

107425 On several occasions, consideration was given to adding a **-t** option to the *unexpand* utility to  
 107426 complement the **-t** in *expand* (see *expand*). The historical intent of *unexpand* was to translate  
 107427 multiple <blank>s into tab stops, where tab stops were a multiple of eight column positions on  
 107428 most UNIX systems. An early proposal omitted **-t** because it seemed outside the scope of the  
 107429 User Portability Utilities option; it was not described in any of the base documents. However,  
 107430 hard-coding tab stops every eight columns was not suitable for the international community and

107431 broke historical precedents for some vendors in the FORTRAN community, so `-t` was restored in  
 107432 conjunction with the list of valid extension categories considered by the standard developers.  
 107433 Thus, *unexpand* is now the logical converse of *expand*.

#### 107434 FUTURE DIRECTIONS

107435 None.

#### 107436 SEE ALSO

107437 *expand*, *tabs*

107438 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

#### 107439 CHANGE HISTORY

107440 First released in Issue 4.

#### 107441 Issue 6

107442 This utility is marked as part of the User Portability Utilities option.

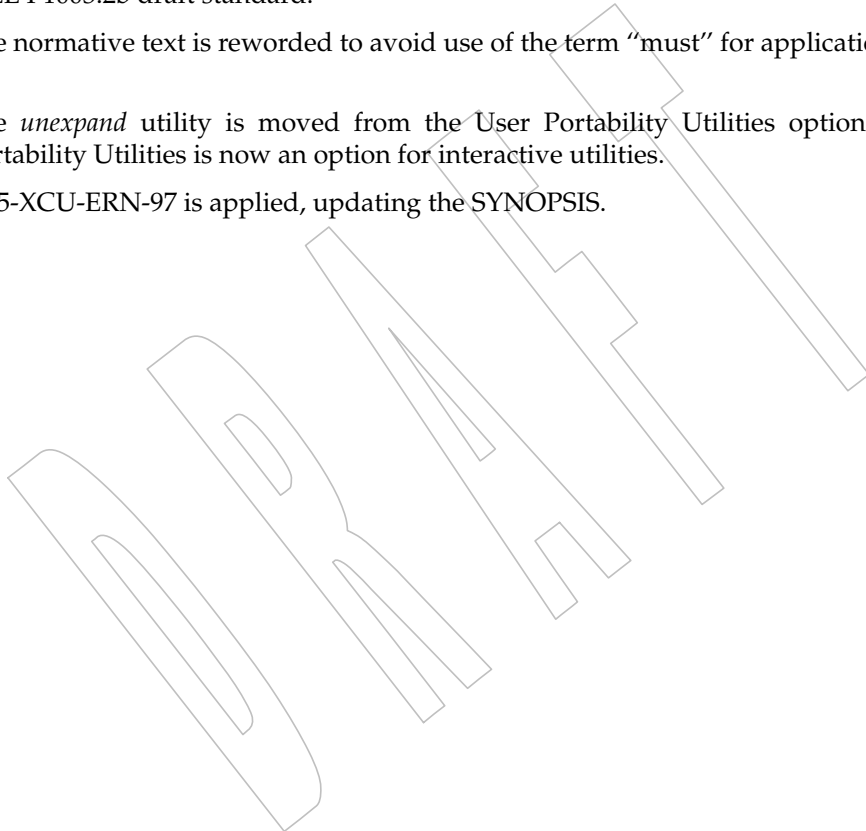
107443 The definition of the `LC_CTYPE` environment variable is changed to align with the  
 107444 IEEE P1003.2b draft standard.

107445 The normative text is reworded to avoid use of the term “must” for application requirements.

#### 107446 Issue 7

107447 The *unexpand* utility is moved from the User Portability Utilities option to the Base. User  
 107448 Portability Utilities is now an option for interactive utilities.

107449 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



107450 **NAME**  
 107451 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)

107452 **SYNOPSIS**  
 107453 XSI unget [-ns] [-r *SID*] *file...*

107454 **DESCRIPTION**  
 107455 The *unget* utility shall reverse the effect of a *get* -e done prior to creating the intended new delta.

107456 **OPTIONS**  
 107457 The *unget* utility shall conform to XBD [Section 12.2](#) (on page 201).

107458 The following options shall be supported:

107459 -r *SID* Uniquely identify which delta is no longer intended. (This would have been  
 107460 specified by *get* as the new delta.) The use of this option is necessary only if two or  
 107461 more outstanding *get* commands for editing on the same SCCS file were done by  
 107462 the same person (login name).

107463 -s Suppress the writing to standard output of the intended delta's SID.

107464 -n Retain the file that was obtained by *get*, which would normally be removed from  
 107465 the current directory.

107466 **OPERANDS**  
 107467 The following operands shall be supported:

107468 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*  
 107469 utility shall behave as though each file in the directory were specified as a named  
 107470 file, except that non-SCCS files (last component of the pathname does not begin  
 107471 with s.) and unreadable files shall be silently ignored.

107472 If exactly one *file* operand appears, and it is '-', the standard input shall be read;  
 107473 each line of the standard input shall be taken to be the name of an SCCS file to be  
 107474 processed. Non-SCCS files and unreadable files shall be silently ignored.

107475 **STDIN**  
 107476 The standard input shall be a text file used only when the *file* operand is specified as '-'. Each  
 107477 line of the text file shall be interpreted as an SCCS pathname.

107478 **INPUT FILES**  
 107479 Any SCCS files processed shall be files of an unspecified format.

107480 **ENVIRONMENT VARIABLES**  
 107481 The following environment variables shall affect the execution of *unget*:

107482 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 107483 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 107484 variables used to determine the values of locale categories.)

107485 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 107486 internationalization variables.

107487 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 107488 characters (for example, single-byte as opposed to multi-byte characters in  
 107489 arguments and input files).

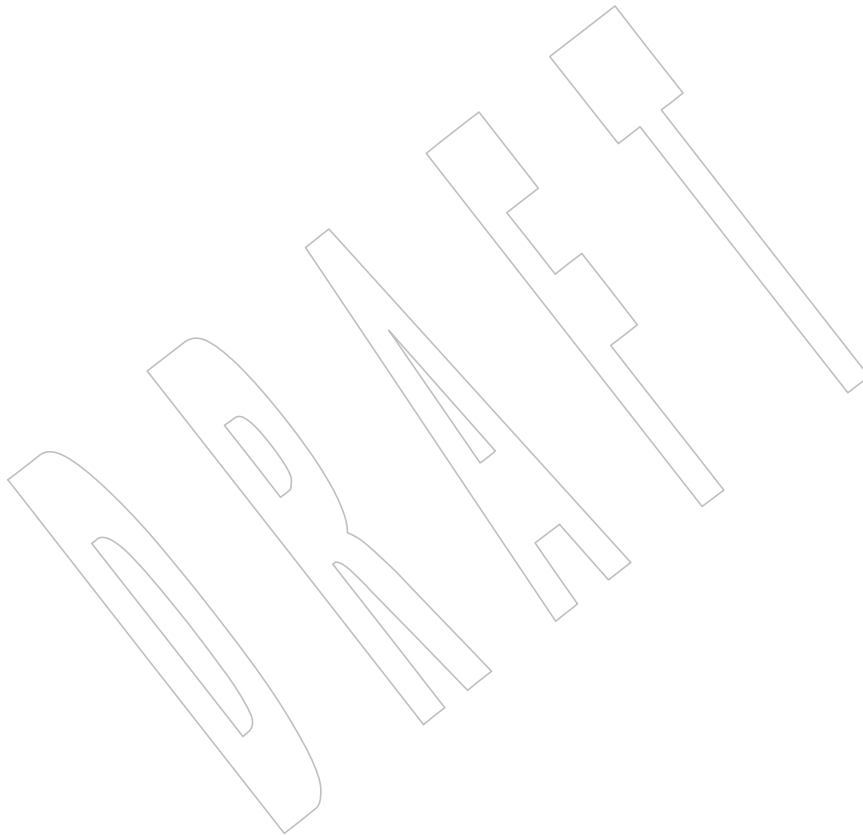


- 107490 **LC\_MESSAGES**
- 107491 Determine the locale that should be used to affect the format and contents of
- 107492 diagnostic messages written to standard error.
- 107493 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 107494 **ASYNCHRONOUS EVENTS**
- 107495 Default.
- 107496 **STDOUT**
- 107497 The standard output shall consist of a line for each file, in the following format:
- 107498 "%s\n", <*SID removed from file*>
- 107499 If there is more than one named file or if a directory or standard input is named, each pathname
- 107500 shall be written before each of the preceding lines:
- 107501 "\n%s:\n", <*pathname*>
- 107502 **STDERR**
- 107503 The standard error shall be used only for diagnostic messages.
- 107504 **OUTPUT FILES**
- 107505 Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a
- 107506 locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and
- 107507 deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.
- 107508 **EXTENDED DESCRIPTION**
- 107509 None.
- 107510 **EXIT STATUS**
- 107511 The following exit values shall be returned:
- 107512 0 Successful completion.
- 107513 >0 An error occurred.
- 107514 **CONSEQUENCES OF ERRORS**
- 107515 Default.
- 107516 **APPLICATION USAGE**
- 107517 None.
- 107518 **EXAMPLES**
- 107519 None.
- 107520 **RATIONALE**
- 107521 None.
- 107522 **FUTURE DIRECTIONS**
- 107523 None.
- 107524 **SEE ALSO**
- 107525 *delta*, *get*, *sact*
- 107526 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +
- 107527 **CHANGE HISTORY**
- 107528 First released in Issue 2.
- 107529 **Issue 6**
- 107530 The normative text is reworded to avoid use of the term “must” for application requirements.

107531  
107532

**Issue 7**

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



107533 **NAME**  
 107534 `uniq` — report or filter out repeated lines in a file

107535 **SYNOPSIS**  
 107536 `uniq [-c|-d|-u] [-f fields] [-s char] [input_file [output_file]]`

107537 **DESCRIPTION**  
 107538 The *uniq* utility shall read an input file comparing adjacent lines, and write one copy of each  
 107539 input line on the output. The second and succeeding copies of repeated adjacent input lines shall  
 107540 not be written.

107541 Repeated lines in the input shall not be detected if they are not adjacent.

107542 **OPTIONS**  
 107543 The *uniq* utility shall conform to XBD Section 12.2 (on page 201), except that '+' may be  
 107544 recognized as an option delimiter as well as '-'.  
 107545 The following options shall be supported:

107546 **-c** Precede each output line with a count of the number of times the line occurred in  
 107547 the input.

107548 **-d** Suppress the writing of lines that are not repeated in the input.

107549 **-f *fields*** Ignore the first *fields* fields on each input line when doing comparisons, where  
 107550 *fields* is a positive decimal integer. A field is the maximal string matched by the  
 107551 basic regular expression:

107552 `[[:blank:]]*^[[:blank:]]*`

107553 If the *fields* option-argument specifies more fields than appear on an input line, a  
 107554 null string shall be used for comparison.

107555 **-s *chars*** Ignore the first *chars* characters when doing comparisons, where *chars* shall be a  
 107556 positive decimal integer. If specified in conjunction with the **-f** option, the first  
 107557 *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-  
 107558 argument specifies more characters than remain on an input line, a null string shall  
 107559 be used for comparison.

107560 **-u** Suppress the writing of lines that are repeated in the input.

107561 **OPERANDS**  
 107562 The following operands shall be supported:

107563 *input\_file* A pathname of the input file. If the *input\_file* operand is not specified, or if the  
 107564 *input\_file* is '-', the standard input shall be used.

107565 *output\_file* A pathname of the output file. If the *output\_file* operand is not specified, the  
 107566 standard output shall be used. The results are unspecified if the file named by  
 107567 *output\_file* is the file named by *input\_file*.

107568 **STDIN**  
 107569 The standard input shall be used only if no *input\_file* operand is specified or if *input\_file* is '-'.  
 107570 See the INPUT FILES section.

107571 **INPUT FILES**  
 107572 The input file shall be a text file.

107573 **ENVIRONMENT VARIABLES**107574 The following environment variables shall affect the execution of *uniq*:

107575 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
 107576 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
 107577 variables used to determine the values of locale categories.)

107578 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 107579 internationalization variables.

107580 *LC\_COLLATE*  
 107581 Determine the locale for ordering rules.

107582 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 107583 characters (for example, single-byte as opposed to multi-byte characters in  
 107584 arguments and input files) and which characters constitute a <blank> in the  
 107585 current locale.

107586 *LC\_MESSAGES*  
 107587 Determine the locale that should be used to affect the format and contents of  
 107588 diagnostic messages written to standard error.

107589 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

107590 **ASYNCHRONOUS EVENTS**

107591 Default.

107592 **STDOUT**

107593 The standard output shall be used only if no *output\_file* operand is specified. See the OUTPUT  
 107594 FILES section.

107595 **STDERR**

107596 The standard error shall be used only for diagnostic messages.

107597 **OUTPUT FILES**107598 If the *-c* option is specified, the output file shall be empty or each line shall be of the form:

107599 "%d %s", &lt;number of duplicates&gt;, &lt;line&gt;

107600 otherwise, the output file shall be empty or each line shall be of the form:

107601 "%s", &lt;line&gt;

107602 **EXTENDED DESCRIPTION**

107603 None.

107604 **EXIT STATUS**

107605 The following exit values shall be returned:

107606 0 The utility executed successfully.

107607 &gt;0 An error occurred.

107608 **CONSEQUENCES OF ERRORS**

107609 Default.

**APPLICATION USAGE**

The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

**EXAMPLES**

The following input file data (but flushed left) was used for a test series on *uniq*:

```
#01 foo0 bar0 fool bar1
#02 bar0 fool bar1 fool
#03 foo0 bar0 fool bar1
#04
#05 foo0 bar0 fool bar1
#06 foo0 bar0 fool bar1
#07 bar0 fool bar1 foo0
```

What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the input data as a sequence of strings delimited by '\n'. Accordingly, for the *fields* member of the sequence, *uniq* interprets unique or repeated adjacent lines strictly relative to the *fields*+1th member.

1. This first example tests the line counting option, comparing each line of the input file data starting from the second field:

```
uniq -c -f 1 uniq_0I.t
 1 #01 foo0 bar0 fool bar1
 1 #02 bar0 fool bar1 foo0
 1 #03 foo0 bar0 fool bar1
 1 #04
 2 #05 foo0 bar0 fool bar1
 1 #07 bar0 fool bar1 foo0
```

The number '2', prefixing the fifth line of output, signifies that the *uniq* utility detected a pair of repeated lines. Given the input data, this can only be true when *uniq* is run using the *-f 1* option (which shall cause *uniq* to ignore the first field on each input line).

2. The second example tests the option to suppress unique lines, comparing each line of the input file data starting from the second field:

```
uniq -d -f 1 uniq_0I.t
#05 foo0 bar0 fool bar1
```

3. This test suppresses repeated lines, comparing each line of the input file data starting from the second field:

```
uniq -u -f 1 uniq_0I.t
#01 foo0 bar0 fool bar1
#02 bar0 fool bar1 fool
#03 foo0 bar0 fool bar1
#04
#07 bar0 fool bar1 foo0
```

4. This suppresses unique lines, comparing each line of the input file data starting from the third character:

```
uniq -d -s 2 uniq_0I.t
```

In the last example, the *uniq* utility found no input matching the above criteria.

107654  
 107655  
 107656  
 107657  
 107658  
 107659  
 107660  
 107661  
 107662  
 107663  
 107664  
 107665  
 107666  
 107667  
 107668  
 107669  
 107670  
 107671  
 107672  
 107673  
 107674  
 107675

**RATIONALE**

Some historical implementations have limited lines to be 1 080 bytes in length, which does not meet the implied {LINE\_MAX} limit.

Earlier versions of this standard allowed the *-number* and *+number* options. These options are no longer specified by POSIX.1-200x but may be present in some implementations.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*comm*, *sort*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

**CHANGE HISTORY**

First released in Issue 2.

**Issue 6**

The obsolescent SYNOPSIS and associated text are removed.

The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/40 is applied, adding *LC\_COLLATE* to the ENVIRONMENT VARIABLES section, and changing “the application shall ensure that” in the OUTPUT FILES section.

**Issue 7**

Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that ‘+’ may be recognized as an option delimiter in the OPTIONS section.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

107676 **NAME**  
 107677 unlink — call the *unlink()* function

107678 **SYNOPSIS**  
 107679 XSI unlink *file*

107680 **DESCRIPTION**  
 107681 The *unlink* utility shall perform the function call:

107682 unlink(*file*);

107683 A user may need appropriate privilege to invoke the *unlink* utility.

107684 **OPTIONS**  
 107685 None.

107686 **OPERANDS**  
 107687 The following operands shall be supported:  
 107688 *file* The pathname of an existing file.

107689 **STDIN**  
 107690 Not used.

107691 **INPUT FILES**  
 107692 Not used.

107693 **ENVIRONMENT VARIABLES**  
 107694 The following environment variables shall affect the execution of *unlink*:

107695 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
 107696 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 107697 variables used to determine the values of locale categories.)

107698 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 107699 internationalization variables.

107700 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 107701 characters (for example, single-byte as opposed to multi-byte characters in  
 107702 arguments).

107703 *LC\_MESSAGES*  
 107704 Determine the locale that should be used to affect the format and contents of  
 107705 diagnostic messages written to standard error.

107706 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

107707 **ASYNCHRONOUS EVENTS**  
 107708 Default.

107709 **STDOUT**  
 107710 None.

107711 **STDERR**  
 107712 The standard error shall be used only for diagnostic messages.

**unlink***Utilities*

107713 **OUTPUT FILES**  
107714 None.

107715 **EXTENDED DESCRIPTION**  
107716 None.

107717 **EXIT STATUS**  
107718 The following exit values shall be returned:  
107719 0 Successful completion.  
107720 >0 An error occurred.

107721 **CONSEQUENCES OF ERRORS**  
107722 Default.

107723 **APPLICATION USAGE**  
107724 None.

107725 **EXAMPLES**  
107726 None.

107727 **RATIONALE**  
107728 None.

107729 **FUTURE DIRECTIONS**  
107730 None.

107731 **SEE ALSO**  
107732 *link, rm*  
107733 XBD [Chapter 8](#) (on page 159)  
107734 XSH *unlink*

107735 **CHANGE HISTORY**  
107736 First released in Issue 5.



107737 **NAME**

107738 uucp — system-to-system copy

107739 **SYNOPSIS**

107740 UU uucp [-cCdfjmr] [-n user] source-file... destination-file

107741 **DESCRIPTION**107742 The *uucp* utility shall copy files named by the *source-file* argument to the *destination-file* argument.  
107743 The files named can be on local or remote systems.107744 The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For  
107745 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
107746 filenames need not be portable to non-internationalized systems, and so on. Under these  
107747 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991  
107748 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used,  
107749 and that only characters defined in the portable filename character set be used for naming files.  
107750 The protocol for transfer of files is unspecified by POSIX.1-200x.107751 Typical implementations of this utility require a communications line configured to use XBD  
107752 [Chapter 11](#) (on page 185), but other communications means may be used. On systems where  
107753 there are no available communications means (either temporarily or permanently), this utility  
107754 shall write an error message describing the problem and exit with a non-zero exit status.107755 **OPTIONS**107756 The *uucp* utility shall conform to XBD [Section 12.2](#) (on page 201).

107757 The following options shall be supported:

- 107758 **-c** Do not copy local file to the spool directory for transfer to the remote machine  
107759 (default).
- 107760 **-C** Force the copy of local files to the spool directory for transfer.
- 107761 **-d** Make all necessary directories for the file copy (default).
- 107762 **-f** Do not make intermediate directories for the file copy.
- 107763 **-j** Write the job identification string to standard output. This job identification can be  
107764 used by *uustat* to obtain the status or terminate a job.
- 107765 **-m** Send mail to the requester when the copy is completed.
- 107766 **-n user** Notify *user* on the remote system that a file was sent.
- 107767 **-r** Do not start the file transfer; just queue the job.

107768 **OPERANDS**

107769 The following operands shall be supported:

107770 *destination-file*, *source-file*107771 A pathname of a file to be copied to, or from, respectively. Either name can be a  
107772 pathname on the local machine, or can have the form:107773 *system-name!pathname*107774 where *system-name* is taken from a list of system names that *uucp* knows about.  
107775 The destination *system-name* can also be a list of names such as:107776 *system-name!system-name!...!system-name!pathname*

107777 in which case, an attempt is made to send the file via the specified route to the

107778 destination. Care should be taken to ensure that intermediate nodes in the route  
107779 are willing to forward information.

107780 The shell pattern matching notation characters '?', '\*', and "[...]" appearing  
107781 in *pathname* shall be expanded on the appropriate system.

107782 Pathnames can be one of:

- 107783 1. An absolute pathname.
- 107784 2. A pathname preceded by *~user* where *user* is a login name on the specified  
107785 system and is replaced by that user's login directory. Note that if an invalid  
107786 login is specified, the default is to the public directory (called *PUBDIR*; the  
107787 actual location of *PUBDIR* is implementation-defined).
- 107788 3. A pathname preceded by *~/destination* where *destination* is appended to  
107789 *PUBDIR*.

107790 **Note:** This destination is treated as a filename unless more than one file is being  
107791 transferred by this request or the destination is already a directory. To  
107792 ensure that it is a directory, follow the destination with a '/'. For  
107793 example, *~/dan/* as the destination makes the directory *PUBDIR/dan* if it  
107794 does not exist and puts the requested files in that directory.

- 107795 4. Anything else shall be prefixed by the current directory.

107796 If the result is an erroneous pathname for the remote system, the copy shall fail. If  
107797 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

107798 The read, write, and execute permissions given by *uucp* are implementation-  
107799 defined.

#### 107800 STDIN

107801 Not used.

#### 107802 INPUT FILES

107803 The files to be copied are regular files.

#### 107804 ENVIRONMENT VARIABLES

107805 The following environment variables shall affect the execution of *uucp*:

107806 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
107807 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
107808 variables used to determine the values of locale categories.)

107809 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
107810 internationalization variables.

107811 *LC\_COLLATE*

107812 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
107813 character collating elements within bracketed filename patterns.

107814 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
107815 characters (for example, single-byte as opposed to multi-byte characters in  
107816 arguments and input files) and the behavior of character classes within bracketed  
107817 filename patterns (for example, "[[:lower:]]\*").

107818 *LC\_MESSAGES*

107819 Determine the locale that should be used to affect the format and contents of  
107820 diagnostic messages written to standard error, and informative messages written  
107821 to standard output.

107822 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 107823 **ASYNCHRONOUS EVENTS**

107824 Default.

## 107825 **STDOUT**

107826 Not used.

## 107827 **STDERR**

107828 The standard error shall be used only for diagnostic messages.

## 107829 **OUTPUT FILES**

107830 The output files (which may be on other systems) are copies of the input files.

107831 If **-m** is used, mail files are modified.

## 107832 **EXTENDED DESCRIPTION**

107833 None.

## 107834 **EXIT STATUS**

107835 The following exit values shall be returned:

107836 0 Successful completion.

107837 >0 An error occurred.

## 107838 **CONSEQUENCES OF ERRORS**

107839 Default.

## 107840 **APPLICATION USAGE**

107841 This utility is part of the UUCP Utilities option and need not be supported by all  
107842 implementations.

107843 The domain of remotely accessible files can (and for obvious security reasons usually should) be  
107844 severely restricted.

107845 Note that the `'!'` character in addresses has to be escaped when using *csh* as a command  
107846 interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary,  
107847 but may be used.

107848 As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate  
107849 system. On an internationalized system, this is done under the control of local settings of  
107850 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
107851 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
107852 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
107853 need not be supported on non-internationalized systems.

## 107854 **EXAMPLES**

107855 None.

## 107856 **RATIONALE**

107857 None.

## 107858 **FUTURE DIRECTIONS**

107859 None.

## 107860 **SEE ALSO**

107861 *mailx*, *uuencode*, *uustat*, *uux*

107862 XBD [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185), [Section 12.2](#) (on page 201)

+

107863

**CHANGE HISTORY**

107864

First released in Issue 2.

107865

**Issue 6**

107866

The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

107867

The UN margin codes and associated shading are removed from the *-C*, *-f*, *-j*, *-n*, and *-r* options in response to The Open Group Base Resolution bwg2001-003.

107868

107869

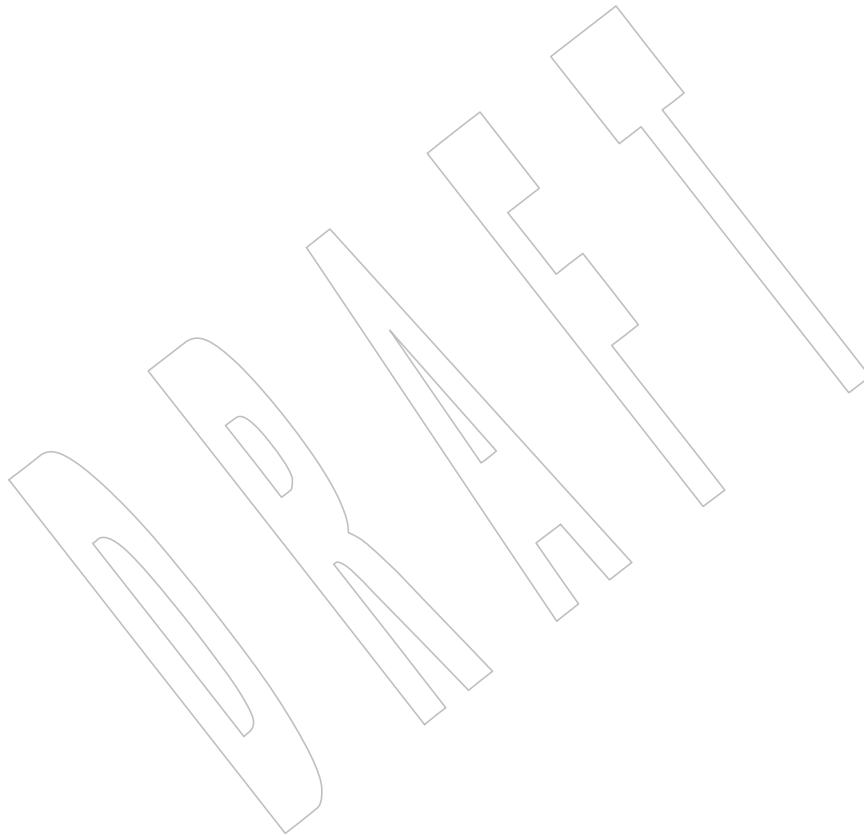
**Issue 7**

107870

SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.

107871

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



107872 **NAME**  
 107873 uudecode — decode a binary file

107874 **SYNOPSIS**  
 107875 uudecode [-o *outfile*] [*file*]

107876 **DESCRIPTION**  
 107877 The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data  
 107878 created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data  
 107879 compatible with one of the formats specified in *uuencode*, and attempt to create or overwrite the  
 107880 file described by the data (or overridden by the **-o** option). The pathname shall be contained in  
 107881 the data or specified by the **-o** option. The file access permission bits and contents for the file to  
 107882 be produced shall be contained in that data. The mode bits of the created file (other than  
 107883 standard output) shall be set from the file access permission bits contained in the data; that is,  
 107884 other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect  
 107885 the file being produced. If either of the *op* characters '+' and '-' (see *chmod*) are specified in  
 107886 symbolic mode, the initial mode on which those operations are based is unspecified.

107887 If the pathname of the file to be produced exists, and the user does not have write permission on  
 107888 that file, *uudecode* shall terminate with an error. If the pathname of the file to be produced exists,  
 107889 and the user has write permission on that file, the existing file shall be overwritten.

107890 If the input data was produced by *uuencode* on a system with a different number of bits per byte  
 107891 than on the target system, the results of *uudecode* are unspecified.

107892 **OPTIONS**  
 107893 The *uudecode* utility shall conform to XBD [Section 12.2](#) (on page 201).

107894 The following option shall be supported by the implementation:

107895 **-o *outfile*** A pathname of a file that shall be used instead of any pathname contained in the  
 107896 input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate  
 107897 standard output.

107898 **OPERANDS**  
 107899 The following operand shall be supported:  
 107900 *file* The pathname of a file containing the output of *uuencode*.

107901 **STDIN**  
 107902 See the INPUT FILES section.

107903 **INPUT FILES**  
 107904 The input files shall be files containing the output of *uuencode*.

107905 **ENVIRONMENT VARIABLES**  
 107906 The following environment variables shall affect the execution of *uudecode*:

107907 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 107908 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 107909 variables used to determine the values of locale categories.)

107910 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 107911 internationalization variables.

107912 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 107913 characters (for example, single-byte as opposed to multi-byte characters in  
 107914 arguments and input files).

- 107915 *LC\_MESSAGES*
- 107916 Determine the locale that should be used to affect the format and contents of
- 107917 diagnostic messages written to standard error.
- 107918 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.
- 107919 **ASYNCHRONOUS EVENTS**
- 107920 Default.
- 107921 **STDOUT**
- 107922 If the file data header encoded by *uuencode* is `-` or `/dev/stdout`, or the `-o /dev/stdout` option
- 107923 overrides the file data, the standard output shall be in the same format as the file originally
- 107924 encoded by *uuencode*. Otherwise, the standard output shall not be used.
- 107925 **STDERR**
- 107926 The standard error shall be used only for diagnostic messages.
- 107927 **OUTPUT FILES**
- 107928 The output file shall be in the same format as the file originally encoded by *uuencode*.
- 107929 **EXTENDED DESCRIPTION**
- 107930 None.
- 107931 **EXIT STATUS**
- 107932 The following exit values shall be returned:
- 107933 0 Successful completion.
- 107934 >0 An error occurred.
- 107935 **CONSEQUENCES OF ERRORS**
- 107936 Default.
- 107937 **APPLICATION USAGE**
- 107938 The user who is invoking *uudecode* must have write permission on any file being created.
- 107939 The output of *uuencode* is essentially an encoded bit stream that is not cognizant of byte
- 107940 boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source,
- 107941 if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only
- 107942 data that is meaningful for such a transfer between architectures is generally character data.
- 107943 **EXAMPLES**
- 107944 None.
- 107945 **RATIONALE**
- 107946 Input files are not necessarily text files, as stated by an early proposal. Although the *uuencode*
- 107947 output is a text file, that output could have been wrapped within another file or mail message
- 107948 that is not a text file.
- 107949 The `-o` option is not historical practice, but was added at the request of WG15 so that the user
- 107950 could override the target pathname without having to edit the input data itself.
- 107951 In early drafts, the `[-o outfile]` option-argument allowed the use of `-` to mean standard output.
- 107952 The symbol `-` has only been used previously in POSIX.1-200x as a standard input indicator. The
- 107953 standard developers did not wish to overload the meaning of `-` in this manner. The `/dev/stdout`
- 107954 concept exists on most modern systems. The `/dev/stdout` syntax does not refer to a new special
- 107955 file. It is just a magic cookie to specify standard output.
- 107956 **FUTURE DIRECTIONS**
- 107957 None.

107958  
107959  
107960  
107961  
107962  
107963  
107964  
107965  
107966  
107967  
107968  
107969  
107970  
107971  
107972  
107973

**SEE ALSO**

*chmod*, *umask*, *uuencode*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

**CHANGE HISTORY**

First released in Issue 4.

**Issue 6**

This utility is marked as part of the User Portability Utilities option.

The `-o outfile` option is added, as specified in the IEEE P1003.2b draft standard.

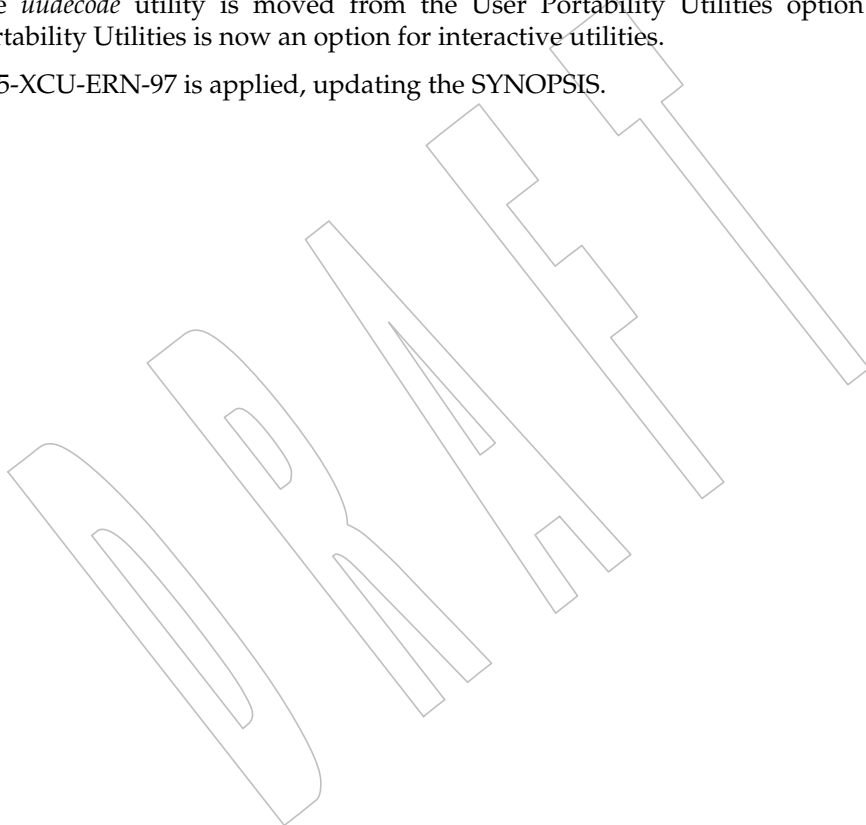
The normative text is reworded to avoid use of the term “must” for application requirements.

IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/35 is applied, clarifying in the DESCRIPTION that the initial mode used if either of the *op* characters is '+' or '-' is unspecified.

**Issue 7**

The *uudecode* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



107974 **NAME**

107975 uuencode — encode a binary file

107976 **SYNOPSIS**107977 uuencode [-m] [*file*] *decode\_pathname*107978 **DESCRIPTION**

107979 The *uuencode* utility shall write an encoded version of the named input file, or standard input if  
 107980 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms  
 107981 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal  
 107982 or symbolic notation) of the input file and the *decode\_pathname*, for re-creation of the file on  
 107983 another system that conforms to this volume of POSIX.1-200x.

107984 **OPTIONS**107985 The *uuencode* utility shall conform to XBD [Section 12.2](#) (on page 201).

107986 The following option shall be supported by the implementation:

107987 **-m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **-m**  
 107988 is not specified, the historical algorithm described in STDOUT shall be used.

107989 **OPERANDS**

107990 The following operands shall be supported:

107991 *decode\_pathname*

107992 The pathname of the file into which the *uudecode* utility shall place the decoded  
 107993 file. Specifying a *decode\_pathname* operand of */dev/stdout* shall indicate that  
 107994 *uudecode* is to use standard output. If there are characters in *decode\_pathname* that  
 107995 are not in the portable filename character set the results are unspecified.

107996 *file* A pathname of the file to be encoded.107997 **STDIN**

107998 See the INPUT FILES section.

107999 **INPUT FILES**

108000 Input files can be files of any type.

108001 **ENVIRONMENT VARIABLES**108002 The following environment variables shall affect the execution of *uuencode*:

108003 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 108004 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 108005 variables used to determine the values of locale categories.)

108006 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 108007 internationalization variables.

108008 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 108009 characters (for example, single-byte as opposed to multi-byte characters in  
 108010 arguments and input files).

108011 **LC\_MESSAGES**

108012 Determine the locale that should be used to affect the format and contents of  
 108013 diagnostic messages written to standard error.

108014 **NSI** **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.



108015 **ASYNCHRONOUS EVENTS**

108016 Default.

108017 **STDOUT**108018 **uuencode Base64 Algorithm**

108019 The standard output shall be a text file (encoded in the character set of the current locale) that  
 108020 begins with the line:

108021 "begin-base64 $\Delta$ %s $\Delta$ %s\n", <mode>, <decode\_pathname>

108022 and ends with the line:

108023 "====\n"

108024 In both cases, the lines shall have no preceding or trailing <blank>s.

108025 The encoding process represents 24-bit groups of input bits as output strings of four encoded  
 108026 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating  
 108027 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit  
 108028 groups, each of which shall be translated into a single digit in the Base64 alphabet. When  
 108029 encoding a bit stream via the Base64 encoding, the bit stream shall be presumed to be ordered  
 108030 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in  
 108031 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit  
 108032 group is used as an index into an array of 64 printable characters, as shown in Table 4-21.

108033 **Table 4-21 uuencode Base64 Values**

| Value | Encoding | Value | Encoding | Value | Encoding | Value | Encoding |
|-------|----------|-------|----------|-------|----------|-------|----------|
| 0     | A        | 17    | R        | 34    | i        | 51    | z        |
| 1     | B        | 18    | S        | 35    | j        | 52    | 0        |
| 2     | C        | 19    | T        | 36    | k        | 53    | 1        |
| 3     | D        | 20    | U        | 37    | l        | 54    | 2        |
| 4     | E        | 21    | V        | 38    | m        | 55    | 3        |
| 5     | F        | 22    | W        | 39    | n        | 56    | 4        |
| 6     | G        | 23    | X        | 40    | o        | 57    | 5        |
| 7     | H        | 24    | Y        | 41    | p        | 58    | 6        |
| 8     | I        | 25    | Z        | 42    | q        | 59    | 7        |
| 9     | J        | 26    | a        | 43    | r        | 60    | 8        |
| 10    | K        | 27    | b        | 44    | s        | 61    | 9        |
| 11    | L        | 28    | c        | 45    | t        | 62    | +        |
| 12    | M        | 29    | d        | 46    | u        | 63    | /        |
| 13    | N        | 30    | e        | 47    | v        |       |          |
| 14    | O        | 31    | f        | 48    | w        | (pad) | =        |
| 15    | P        | 32    | g        | 49    | x        |       |          |
| 16    | Q        | 33    | h        | 50    | y        |       |          |

108052 The character referenced by the index shall be placed in the output string.

108053 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters  
 108054 each. All line breaks or other characters not found in the table shall be ignored by decoding  
 108055 software (see *uudecode*).

108056 Special processing shall be performed if fewer than 24 bits are available at the end of a message  
 108057 or encapsulated part of a message. A full encoding quantum shall always be completed at the  
 108058 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall  
 108059 be added (on the right) to form an integral number of 6-bit groups. Output character positions

108060 that are not required to represent actual input data shall be set to the character '='. Since all  
108061 Base64 input is an integral number of octets, only the following cases can arise:

- 108062 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit  
108063 of encoded output shall be an integral multiple of 4 characters with no '=' padding.
- 108064 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded  
108065 output shall be three characters followed by one '=' padding character.
- 108066 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded  
108067 output shall be two characters followed by two '=' padding characters.

108068 A terminating "====" evaluates to nothing and denotes the end of the encoded data.

#### 108069 uuencode Historical Algorithm

108070 The standard output shall be a text file (encoded in the character set of the current locale) that  
108071 begins with the line:

108072 "beginΔ%sΔ%s\n" <mode>, <decode\_pathname>

108073 and ends with the line:

108074 "end\n"

108075 In both cases, the lines shall have no preceding or trailing <blank>s.

108076 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input  
108077 and writes four characters of output by splitting the input at six-bit intervals into four octets,  
108078 containing data in the lower six bits only. These octets shall be converted to characters by adding  
108079 a value of 0x20 to each octet, so that each octet is in the range [0x20,0x5f], and then it shall be  
108080 assumed to represent a printable character in the ISO/IEC 646:1991 standard encoded character  
108081 set. It then shall be translated into the corresponding character codes for the codeset in use in the  
108082 current locale. (For example, the octet 0x41, representing 'A', would be translated to 'A' in the  
108083 current codeset, such as 0xc1 if it were EBCDIC.)

108084 Where the bits of two octets are combined, the least significant bits of the first octet shall be  
108085 shifted left and combined with the most significant bits of the second octet shifted right. Thus  
108086 the three octets *A*, *B*, *C* shall be converted into the four octets:

108087  $0x20 + (((A \gg 2) \ll 6) \mid ((B \gg 4) \& 0xF)) \& 0x3F$

108088  $0x20 + (((A \ll 4) \mid ((B \gg 2) \& 0x3)) \& 0x3F) \mid ((C \gg 6) \& 0x3) \& 0x3F$

108089  $0x20 + (((B \ll 2) \mid ((C \gg 4) \& 0xF)) \& 0x3F) \mid ((C \gg 2) \& 0x3) \& 0x3F$

108090  $0x20 + (((C \ll 0) \mid ((C \gg 0) \& 0x3)) \& 0x3F)$

108091 These octets then shall be translated into the local character set.

108092 Each encoded line contains a length character, equal to the number of characters to be decoded  
108093 plus 0x20 translated to the local character set as described above, followed by the encoded  
108094 characters. The maximum number of octets to be encoded on each line shall be 45.

#### 108095 STDERR

108096 The standard error shall be used only for diagnostic messages.

#### 108097 OUTPUT FILES

108098 None.

#### 108099 EXTENDED DESCRIPTION

108100 None.

108101 **EXIT STATUS**

108102 The following exit values shall be returned:

108103 0 Successful completion.

108104 &gt;0 An error occurred.

108105 **CONSEQUENCES OF ERRORS**

108106 Default.

108107 **APPLICATION USAGE**108108 The file is expanded by 35 percent (each three octets become four, plus control information)  
108109 causing it to take longer to transmit.

108110 Since this utility is intended to create files to be used for data interchange between systems with  
108111 possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991  
108112 standard was chosen for a midpoint in the algorithm as a known reference point. The output  
108113 from *uuencode* is a text file on the local system. If the output were in the ISO/IEC 646:1991  
108114 standard codeset, it might not be a text file (at least because the <newline>s might not match),  
108115 and the goal of creating a text file would be defeated. If this text file was then carried to another  
108116 machine with the same codeset, it would be perfectly compatible with that system's *uudecode*. If  
108117 it was transmitted over a mail system or sent to a machine with a different codeset, it is assumed  
108118 that, as for every other text file, some translation mechanism would convert it (by the time it  
108119 reached a user on the other system) into an appropriate codeset. This translation only makes  
108120 sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard  
108121 representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives,  
108122 intermixed with other text files in the same codeset.

108123 **EXAMPLES**

108124 None.

108125 **RATIONALE**

108126 A new algorithm was added at the request of the international community to parallel work in  
108127 RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding  
108128 is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A  
108129 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented  
108130 per printable character. (The extra 65th character, '=', is used to signify a special processing  
108131 function.)

108132 This subset has the important property that it is represented identically in all versions of the  
108133 ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also  
108134 represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not  
108135 share this property, which is the reason that a second algorithm was added to the ISO POSIX-2  
108136 standard.

108137 The string "====" was used for the termination instead of the end used in the original format  
108138 because the latter is a string that could be valid encoded input.

108139 In an early draft, the `-m` option was named `-b` (for Base64), but it was renamed to reflect its  
108140 relationship to the RFC 2045. A `-u` was also present to invoke the default algorithm, but since  
108141 this was not historical practice, it was omitted as being unnecessary.

108142 See the RATIONALE section in *uudecode* for the derivation of the `/dev/stdout` symbol.

108143 **FUTURE DIRECTIONS**

108144 None.

108145 **SEE ALSO**108146 *chmod, mailx, uuencode*108147 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +108148 **CHANGE HISTORY**

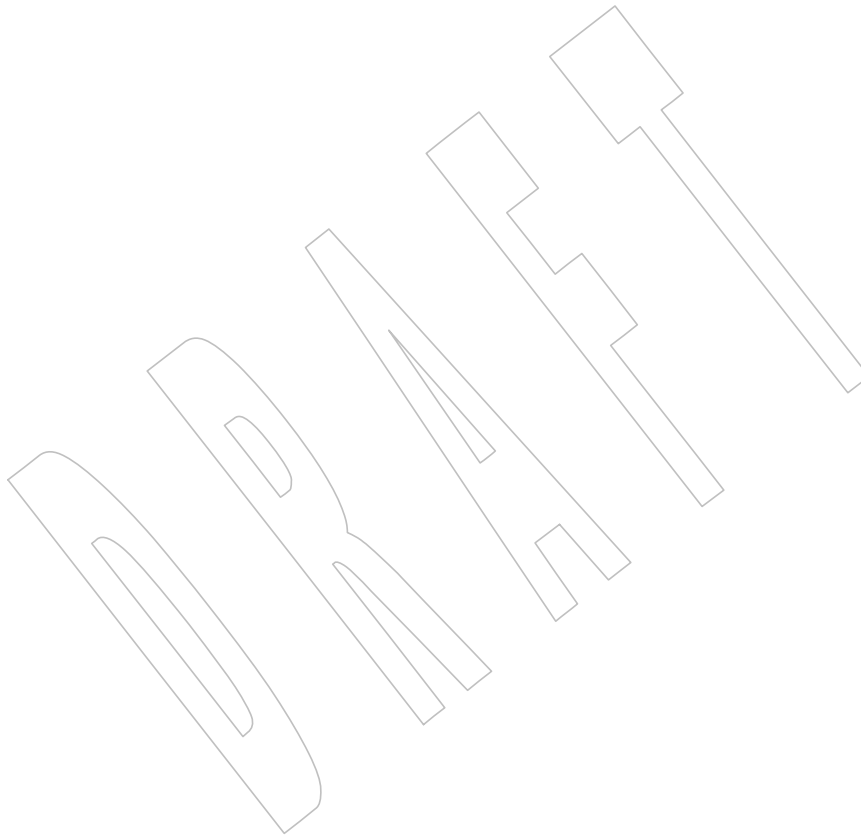
108149 First released in Issue 4.

108150 **Issue 6**

108151 This utility is marked as part of the User Portability Utilities option.

108152 The Base64 algorithm and the ability to output to `/dev/stdout` are added as specified in the  
108153 IEEE P1003.2b draft standard.108154 **Issue 7**108155 The *uuencode* utility is moved from the User Portability Utilities option to the Base. User  
108156 Portability Utilities is now an option for interactive utilities.

108157 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



108158 **NAME**  
 108159 uustat — uucp status inquiry and job control

108160 **SYNOPSIS**  
 108161 UU uustat [-q|-k *jobid*|-r *jobid*]  
 108162 uustat [-s *system*] [-u *user*]

108163 **DESCRIPTION**  
 108164 The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or  
 108165 provide general status on *uucp* connections to other systems.

108166 When no options are given, *uustat* shall write to standard output the status of all *uucp* requests  
 108167 issued by the current user.

108168 Typical implementations of this utility require a communications line configured to use XBD  
 108169 Chapter 11 (on page 185), but other communications means may be used. On systems where  
 108170 there are no available communications means (either temporarily or permanently), this utility  
 108171 shall write an error message describing the problem and exit with a non-zero exit status.

108172 **OPTIONS**  
 108173 The *uustat* utility shall conform to XBD Section 12.2 (on page 201).

108174 The following options shall be supported:

108175 **-q** Write the jobs queued for each machine.  
 108176 **-k *jobid*** Kill the *uucp* request whose job identification is *jobid*. The application shall ensure  
 108177 that the killed *uucp* request belongs to the person invoking *uustat* unless that user  
 108178 has appropriate privileges.  
 108179 **-r *jobid*** Rejuvenate *jobid*. The files associated with *jobid* are touched so that their  
 108180 modification time is set to the current time. This prevents the cleanup program  
 108181 from deleting the job until the jobs modification time reaches the limit imposed by  
 108182 the program.  
 108183 **-s *system*** Write the status of all *uucp* requests for remote system *system*.  
 108184 **-u *user*** Write the status of all *uucp* requests issued by *user*.

108185 **OPERANDS**  
 108186 None.

108187 **STDIN**  
 108188 Not used.

108189 **INPUT FILES**  
 108190 None.

108191 **ENVIRONMENT VARIABLES**  
 108192 The following environment variables shall affect the execution of *uustat*:

108193 **LANG** Provide a default value for the internationalization variables that are unset or null.  
 108194 (See XBD Section 8.2 (on page 160) for the precedence of internationalization  
 108195 variables used to determine the values of locale categories.)  
 108196 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 108197 internationalization variables.

108198 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 108199 characters (for example, single-byte as opposed to multi-byte characters in  
 108200 arguments).

108201 **LC\_MESSAGES**  
 108202 Determine the locale that should be used to affect the format and contents of  
 108203 diagnostic messages written to standard error, and informative messages written  
 108204 to standard output.

108205 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 108206 **ASYNCHRONOUS EVENTS**

108207 Default.

## 108208 **STDOUT**

108209 The standard output shall consist of information about each job selected, in an unspecified  
 108210 format. The information shall include at least the job ID, the user ID or name, and the remote  
 108211 system name.

## 108212 **STDERR**

108213 The standard error shall be used only for diagnostic messages.

## 108214 **OUTPUT FILES**

108215 None.

## 108216 **EXTENDED DESCRIPTION**

108217 None.

## 108218 **EXIT STATUS**

108219 The following exit values shall be returned:

108220 0 Successful completion.

108221 >0 An error occurred.

## 108222 **CONSEQUENCES OF ERRORS**

108223 Default.

## 108224 **APPLICATION USAGE**

108225 This utility is part of the UUCP Utilities option and need not be supported by all  
 108226 implementations.

## 108227 **EXAMPLES**

108228 None.

## 108229 **RATIONALE**

108230 None.

## 108231 **FUTURE DIRECTIONS**

108232 None.

## 108233 **SEE ALSO**

108234 *uucp*

108235 XBD [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185), [Section 12.2](#) (on page 201)

+

## 108236 **CHANGE HISTORY**

108237 First released in Issue 2.

### 108238 **Issue 6**

108239 The normative text is reworded to avoid use of the term “must” for application requirements.

108240 The *LC\_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

108241 The UN margin code and associated shading are removed from the *-q* option in response to The

- 108242 Open Group Base Resolution bwg2001-003.
- 108243 **Issue 7**
- 108244 SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.
- 108245 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.



108246 **NAME**

108247 uux — remote command execution

108248 **SYNOPSIS**108249 UU uux [-jnp] *command-string*108250 **DESCRIPTION**

108251 The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see  
 108252 [Section 2.9](#), on page 2263) on a specified system, and then send the standard output of the  
 108253 command to a file on a specified system. Only the first command of a pipeline can have a *system-*  
 108254 *name!* prefix. All other commands in the pipeline shall be executed on the system of the first  
 108255 command.

108256 The following restrictions are applicable to the shell pipeline processed by *uux*:

- 108257 • In gathering files from different systems, pathname expansion shall not be performed by  
 108258 *uux*. Thus, a request such as:

108259 uux "c99 remsys!~/\*.c"

108260 would attempt to copy the file named literally \*.c to the local system.

- 108261 • The redirection operators ">>", "<<", ">|", and ">&" shall not be accepted. Any use of  
 108262 these redirection operators shall cause this utility to write an error message describing the  
 108263 problem and exit with a non-zero exit status.

- 108264 • The reserved word ! cannot be used at the head of the pipeline to modify the exit status.  
 108265 (See the *command-string* operand description below.)

- 108266 • Alias substitution shall not be performed.

108267 A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by  
 108268 *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest*  
 108269 (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is  
 108270 implementation-defined), or a simple filename (which is prefixed by *uux* with the current  
 108271 directory). See *uucp* for the details.

108272 The execution of commands on remote systems shall take place in an execution directory known  
 108273 to the *uucp* system. All files required for the execution shall be put into this directory unless they  
 108274 already reside on that machine. Therefore, the application shall ensure that non-local filenames  
 108275 (without path or machine reference) are unique within the *uux* request.

108276 The *uux* utility shall attempt to get all files to the execution system. For files that are output files,  
 108277 the application shall ensure that the filename is escaped using parentheses.

108278 The remote system shall notify the user by mail if the requested command on the remote system  
 108279 was disallowed or the files were not accessible. This notification can be turned off by the *-n*  
 108280 option.

108281 Typical implementations of this utility require a communications line configured to use XBD |  
 108282 [Chapter 11](#) (on page 185), but other communications means may be used. On systems where |  
 108283 there are no available communications means (either temporarily or permanently), this utility |  
 108284 shall write an error message describing the problem and exit with a non-zero exit status.

108285 The *uux* utility cannot guarantee support for all character encodings in all circumstances. For  
 108286 example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and  
 108287 filenames need not be portable to non-internationalized systems, and so on. Under these  
 108288 circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991



108289 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used  
 108290 and that only characters defined in the portable filename character set be used for naming files.

**OPTIONS**

108291 The *uux* utility shall conform to XBD Section 12.2 (on page 201). |

108293 The following options shall be supported:

- 108294 **-j** Write the job identification string to standard output. This job identification can be  
 108295 used by *uustat* to obtain the status or terminate a job. -
- 108296 **-n** Do not notify the user if the command fails.
- 108297 **-p** Make the standard input to *uux* the standard input to the *command-string*. +

**OPERANDS**

108298 The following operand shall be supported:

108300 *command-string*

108301 A string made up of one or more arguments that are similar to normal command  
 108302 arguments, except that the command and any filenames can be prefixed by *system-*  
 108303 *name!*. A null *system-name* shall be interpreted as the local system.

**STDIN**

108304 The standard input shall not be used unless the *'-'* or **-p** option is specified; in those cases, the  
 108305 standard input shall be made the standard input of the *command-string*.  
 108306

**INPUT FILES**

108307 Input files shall be selected according to the contents of *command-string*.  
 108308

**ENVIRONMENT VARIABLES**

108309 The following environment variables shall affect the execution of *uux*:

108311 **LANG** Provide a default value for the internationalization variables that are unset or null. |  
 108312 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 108313 variables used to determine the values of locale categories.)

108314 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 108315 internationalization variables.

108316 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 108317 characters (for example, single-byte as opposed to multi-byte characters in  
 108318 arguments).

108319 **LC\_MESSAGES**  
 108320 Determine the locale that should be used to affect the format and contents of  
 108321 diagnostic messages written to standard error.

108322 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

**ASYNCHRONOUS EVENTS**

108323 Default.  
 108324

**STDOUT**

108325 The standard output shall not be used unless the **-j** option is specified; in that case, the job  
 108326 identification string shall be written to standard output in the following format:  
 108327

108328 "%s\n", <*jobid*>

**STDERR**

108329 The standard error shall be used only for diagnostic messages.  
 108330

**OUTPUT FILES**

108331 Output files shall be created or written, or both, according to the contents of *command-string*.  
 108332

108333 If **-n** is not used, mail files shall be modified following any command or file-access failures on  
 108334 the remote system.

**EXTENDED DESCRIPTION**

108335 None.  
 108336

**EXIT STATUS**

108337 The following exit values shall be returned:  
 108338

108339 0 Successful completion.

108340 >0 An error occurred.

**CONSEQUENCES OF ERRORS**

108341 Default.  
 108342

**APPLICATION USAGE**

108343 This utility is part of the UUCP Utilities option and need not be supported by all  
 108344 implementations.  
 108345

108346 Note that, for security reasons, many installations limit the list of commands executable on  
 108347 behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail  
 108348 via *uux*.

108349 Any characters special to the command interpreter should be quoted either by quoting the entire  
 108350 *command-string* or quoting the special characters as individual arguments.

108351 As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are  
 108352 expanded on the appropriate local system. This is done under the control of local settings of  
 108353 *LC\_COLLATE* and *LC\_CTYPE*. Thus, care should be taken when using bracketed filename  
 108354 patterns, as collation and typing rules may vary from one system to another. Also be aware that  
 108355 certain types of expression (that is, equivalence classes, character classes, and collating symbols)  
 108356 need not be supported on non-internationalized systems.

**EXAMPLES**

108357  
 108358 1. The following command gets **file1** from system **a** and **file2** from system **b**, executes *diff* on  
 108359 the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR*  
 108360 is the *uucp* public directory on the local system.)

108361 `uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"`

108362 2. The following command fails because *uux* places all files copied to a system in the same  
 108363 working directory. Although the files **xyz** are from two different systems, their filenames  
 108364 are the same and conflict.

108365 `uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"`

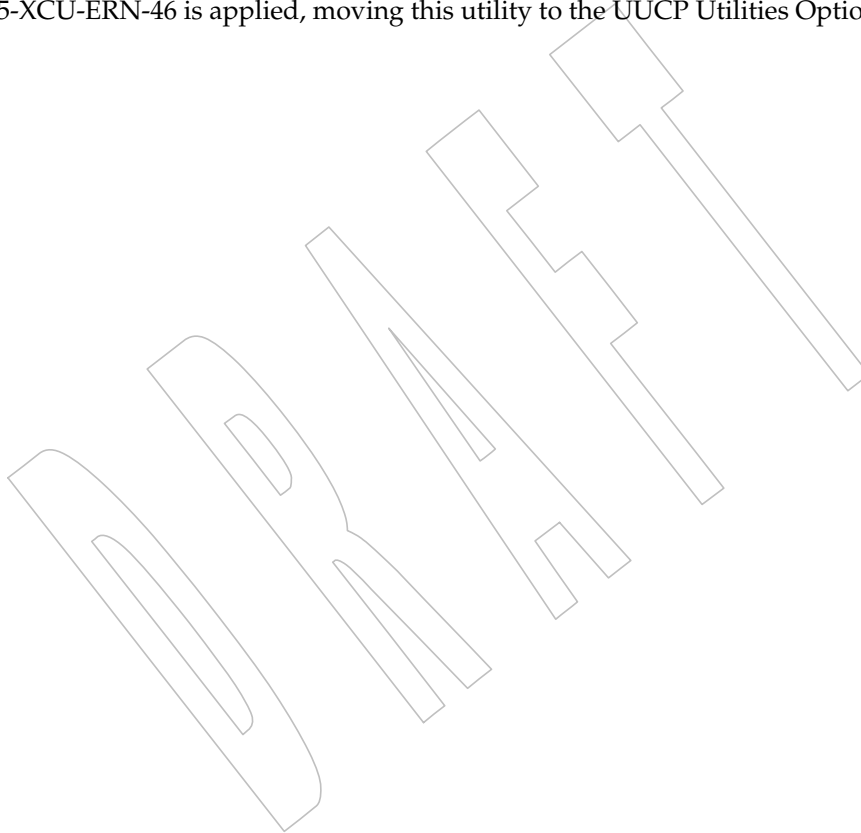
108366 3. The following command succeeds (assuming *diff* is permitted on system **a**) because the  
 108367 file local to system **a** is not copied to the working directory, and hence does not conflict  
 108368 with the file from system **c**.

108369 `uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"`

**RATIONALE**

108370 None.  
 108371

- 108372 **FUTURE DIRECTIONS**
- 108373       None.
- 108374 **SEE ALSO**
- 108375       Chapter 2 (on page 2245), *uucp*, *uuencode*, *uustat*
- 108376       XBD Chapter 8 (on page 159), Chapter 11 (on page 185), Section 12.2 (on page 201) +
- 108377 **CHANGE HISTORY**
- 108378       First released in Issue 2.
- 108379 **Issue 6**
- 108380       The obsolescent SYNOPSIS is removed.
- 108381       The normative text is reworded to avoid use of the term “must” for application requirements.
- 108382       The UN margin code and associated shading are removed from the `-j` option in response to The
- 108383       Open Group Base Resolution bwg2001-003.
- 108384 **Issue 7**
- 108385       SD5-XCU-ERN-46 is applied, moving this utility to the UUCP Utilities Option Group.



108386 **NAME**108387 val — validate SCCS files (**DEVELOPMENT**)108388 **SYNOPSIS**108389 XSI val -  
108390 val [-s] [-m *name*] [-r *SID*] [-y *type*] *file*...108391 **DESCRIPTION**108392 The *val* utility shall determine whether the specified *file* is an SCCS file meeting the  
108393 characteristics specified by the options.108394 **OPTIONS**108395 The *val* utility shall conform to XBD [Section 12.2](#) (on page 201), except that the usage of the ‘-’  
108396 operand is not strictly as intended by the guidelines (that is, reading options and operands from  
108397 standard input).

108398 The following options shall be supported:

108399 **-m** *name* Specify a *name*, which is compared with the SCCS %M% keyword in *file*; see [get](#).108400 **-r** *SID* Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be  
108401 made to determine whether the *SID* is ambiguous (for example, **-r 1** is ambiguous  
108402 because it physically does not exist but implies 1.1, 1.2, and so on, which may exist)  
108403 or invalid (for example, **-r 1.0** or **-r 1.1.0** are invalid because neither case can exist  
108404 as a valid delta number). If the *SID* is valid and not ambiguous, a check shall be  
108405 made to determine whether it actually exists.108406 **-s** Silence the diagnostic message normally written to standard output for any error  
108407 that is detected while processing each named file on a given command line.108408 **-y** *type* Specify a *type*, which shall be compared with the SCCS %Y% keyword in *file*; see  
108409 [get](#).108410 **OPERANDS**

108411 The following operands shall be supported:

108412 *file* A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is  
108413 ‘-’, the standard input shall be read: each line shall be independently processed  
108414 as if it were a command line argument list. (However, the line is not subjected to  
108415 any of the shell word expansions, such as parameter expansion or quote removal.)108416 **STDIN**108417 The standard input shall be a text file used only when the *file* operand is specified as ‘-’.108418 **INPUT FILES**

108419 Any SCCS files processed shall be files of an unspecified format.

108420 **ENVIRONMENT VARIABLES**108421 The following environment variables shall affect the execution of *val*:108422 **LANG** Provide a default value for the internationalization variables that are unset or null.  
108423 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
108424 variables used to determine the values of locale categories.)108425 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
108426 internationalization variables.

108427 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 108428 characters (for example, single-byte as opposed to multi-byte characters in  
 108429 arguments and input files).

108430 **LC\_MESSAGES**  
 108431 Determine the locale that should be used to affect the format and contents of  
 108432 diagnostic messages written to standard error, and informative messages written  
 108433 to standard output.

108434 **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

## 108435 ASYNCHRONOUS EVENTS

108436 Default.

## 108437 STDOUT

108438 The standard output shall consist of informative messages about either:

- 108439 1. Each file processed
- 108440 2. Each command line read from standard input

108441 If the standard input is not used, for each *file* operand yielding a discrepancy, the output line  
 108442 shall have the following format:

108443 "%s: %s\n", <pathname>, <unspecified string>

108444 If standard input is used, a line of input shall be written before each of the preceding lines for  
 108445 files containing discrepancies:

108446 "%s:\n", <input line>

## 108447 STDERR

108448 Not used.

## 108449 OUTPUT FILES

108450 None.

## 108451 EXTENDED DESCRIPTION

108452 None.

## 108453 EXIT STATUS

108454 The 8-bit code returned by *val* shall be a disjunction of the possible errors; that is, it can be  
 108455 interpreted as a bit string where set bits are interpreted as follows:

108456 0x80 = Missing file argument.  
 108457 0x40 = Unknown or duplicate option.  
 108458 0x20 = Corrupted SCCS file.  
 108459 0x10 = Cannot open file or file not SCCS.  
 108460 0x08 = *SID* is invalid or ambiguous.  
 108461 0x04 = *SID* does not exist.  
 108462 0x02 = %Y%, -y mismatch.  
 108463 0x01 = %M%, -m mismatch.

108464 Note that *val* can process two or more files on a given command line and can process multiple  
 108465 command lines (when reading the standard input). In these cases an aggregate code shall be  
 108466 returned: a logical OR of the codes generated for each command line and file processed.

## 108467 CONSEQUENCES OF ERRORS

108468 Default.

**APPLICATION USAGE**

Since the *val* exit status sets the 0x80 bit, shell applications checking "\$?" cannot tell if it terminated due to a missing file argument or receipt of a signal.

**EXAMPLES**

In a directory with three SCCS files—*s.x* (of *t* type "text"), *s.y*, and *s.z* (a corrupted file)—the following command could produce the output shown:

```
val - <<EOF
-y source s.x
-m y s.y
s.z
EOF

-y source s.x

s.x: %Y%, -y mismatch
s.z

s.z: corrupted SCCS file
```

**RATIONALE**

None.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*admin*, *delta*, *get*, *prs*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

**CHANGE HISTORY**

First released in Issue 2.

**Issue 6**

The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT STATUS.

**Issue 7**

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

108498 **NAME**  
 108499 `vi` — screen-oriented (visual) display editor

108500 **SYNOPSIS**  
 108501 UP `vi [-rR] [-c command] [-t tagstring] [-w size] [file...]`

## 108502 DESCRIPTION

108503 This utility shall be provided on systems that both support the User Portability Utilities option  
 108504 and define the POSIX2\_CHAR\_TERM symbol. On other systems it is optional.

108505 The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the  
 108506 editor are described in POSIX.1-200x; see the line editor *ex* for additional editing capabilities  
 108507 used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex* commands from  
 108508 within *vi*.

108509 This reference page uses the term *edit buffer* to describe the current working text. No specific  
 108510 implementation is implied by this term. All editing changes are performed on the edit buffer,  
 108511 and no changes to it shall affect any file until an editor command writes the file.

108512 When using *vi*, the terminal screen acts as a window into the editing buffer. Changes made to  
 108513 the editing buffer shall be reflected in the screen display; the position of the cursor on the screen  
 108514 shall indicate the position within the editing buffer.

108515 Certain terminals do not have all the capabilities necessary to support the complete *vi* definition.  
 108516 When these commands cannot be supported on such terminals, this condition shall not produce  
 108517 an error message such as “not an editor command” or report a syntax error. The implementation  
 108518 may either accept the commands and produce results on the screen that are the result of an  
 108519 unsuccessful attempt to meet the requirements of this volume of POSIX.1-200x or report an error  
 108520 describing the terminal-related deficiency.

## 108521 OPTIONS

108522 The *vi* utility shall conform to XBD [Section 12.2](#) (on page 201), except that ‘+’ may be  
 108523 recognized as an option delimiter as well as ‘-’.

108524 The following options shall be supported:

108525 `-c command` See the *ex* command description of the `-c` option.

108526 `-r` See the *ex* command description of the `-r` option.

108527 `-R` See the *ex* command description of the `-R` option.

108528 `-t tagstring` See the *ex* command description of the `-t` option.

108529 `-w size` See the *ex* command description of the `-w` option.

## 108530 OPERANDS

108531 See the OPERANDS section of the *ex* command for a description of the operands supported by  
 108532 the *vi* command.

## 108533 STDIN

108534 If standard input is not a terminal device, the results are undefined. The standard input consists  
 108535 of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.

108536 If a read from the standard input returns an error, or if the editor detects an end-of-file condition  
 108537 from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

108538 **INPUT FILES**

108539 See the INPUT FILES section of the *ex* command for a description of the input files supported by  
108540 the *vi* command.

108541 **ENVIRONMENT VARIABLES**

108542 See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables  
108543 that affect the execution of the *vi* command.

108544 **ASYNCHRONOUS EVENTS**

108545 See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the  
108546 execution of the *vi* command.

108547 **STDOUT**

108548 If standard output is not a terminal device, undefined results occur.

108549 Standard output may be used for writing prompts to the user, for informational messages, and  
108550 for writing lines from the file.

108551 **STDERR**

108552 If standard output is not a terminal device, undefined results occur.

108553 The standard error shall be used only for diagnostic messages.

108554 **OUTPUT FILES**

108555 See the OUTPUT FILES section of the *ex* command for a description of the output files  
108556 supported by the *vi* command.

108557 **EXTENDED DESCRIPTION**

108558 If the terminal does not have the capabilities necessary to support an unspecified portion of the  
108559 *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after  
108560 initialization, *vi* shall be in command mode; text input mode can be entered by one of several  
108561 commands used to insert or change text. In text input mode, <ESC> can be used to return to  
108562 command mode; other uses of <ESC> are described later in this section; see [Terminate](#)  
108563 [Command or Input Mode](#) (on page 3224).

108564 **Initialization in *ex* and *vi***

108565 See [Initialization in \*ex\* and \*vi\*](#) (on page 2576) for a description of *ex* and *vi* initialization for the *vi*  
108566 utility.

108567 **Command Descriptions in *vi***

108568 The following symbols are used in this reference page to represent arguments to commands.

108569 *buffer* See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility;  
108570 see [Command Descriptions in \*ex\*](#) (on page 2586).

108571 In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*]  
108572 preceding the command name, they can be specified in either order.

108573 *count* A positive integer used as an optional argument to most commands, either to give a  
108574 repeat count or as a size. This argument is optional and shall default to 1 unless  
108575 otherwise specified.

108576 The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R,  
108577 <control>-], %, &, ^, D, m, M, Q, u, U, and ZZ do not have *count* as an optional  
108578 argument. Regardless, it shall not be an error to specify a *count* to these commands, and  
108579 any specified *count* shall be ignored.

108580 *motion* An optional trailing argument used by the !, <, >, c, d, and y commands, which is used  
108581 to indicate the region of text that shall be affected by the command. The motion can be  
108582 either one of the command characters repeated or one of several other *vi* commands



108583 (listed in the following table). Each of the applicable commands specifies the region of  
 108584 text matched by repeating the command; each command that can be used as a motion  
 108585 command specifies the region of text it affects.

108586 Commands that take *motion* arguments operate on either lines or characters, depending  
 108587 on the circumstances. When operating on lines, all lines that fall partially or wholly  
 108588 within the text region specified for the command shall be affected. When operating on  
 108589 characters, only the exact characters in the specified text region shall be affected. Each  
 108590 motion command specifies this individually.

108591 When commands that may be motion commands are not used as motion commands,  
 108592 they shall set the current position to the current line and column as specified.

108593 The following commands shall be valid cursor motion commands:

|        |                   |   |    |   |   |   |
|--------|-------------------|---|----|---|---|---|
| 108594 | <apostrophe>      | ( | -  | j | H |   |
| 108595 | <carriage-return> | ) | \$ | k | L |   |
| 108596 | <comma>           | [ | [  | % | l | M |
| 108597 | <control>-H       | ] | ]  | _ | n | N |
| 108598 | <control>-N       | { | ;  | t | T |   |
| 108599 | <control>-P       | } | ?  | w | W |   |
| 108600 | <grave accent>    | ^ | b  | B |   |   |
| 108601 | <newline>         | + | e  | E |   |   |
| 108602 | <space>           |   | f  | F |   |   |
| 108603 | <zero>            | / | h  | G |   |   |

108604 Any *count* that is specified to a command that has an associated motion command shall  
 108605 be applied to the motion command. If a *count* is applied to both the command and its  
 108606 associated motion command, the effect shall be multiplicative.

108607 The following symbols are used in this section to specify locations in the edit buffer:

108608 *current character*

108609 The character that is currently indicated by the cursor.

108610 *end of a line*

108611 The point located between the last non-<newline> (if any) and the terminating  
 108612 <newline> of a line. For an empty line, this location coincides with the beginning of the  
 108613 line.

108614 *end of the edit buffer*

108615 The location corresponding to the end of the last line in the edit buffer.

108616 The following symbols are used in this section to specify command actions:

108617 *bigword* In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

- 108618 1. A maximal sequence of non-<blank>s preceded and followed by <blank>s or the  
 108619 beginning or end of a line or the edit buffer
- 108620 2. One or more sequential blank lines
- 108621 3. The first character in the edit buffer
- 108622 4. The last non-<newline> in the edit buffer

108623 *word* In the POSIX locale, *vi* shall recognize five kinds of words:

- 108624 1. A maximal sequence of letters, digits, and underscores, delimited at both ends  
 108625 by:

- 108626 — Characters other than letters, digits, or underscores
- 108627 — The beginning or end of a line
- 108628 — The beginning or end of the edit buffer
- 108629 2. A maximal sequence of characters other than letters, digits, underscores, or
- 108630 <blank>s, delimited at both ends by:
- 108631 — A letter, digit, underscore
- 108632 — <blank>s
- 108633 — The beginning or end of a line
- 108634 — The beginning or end of the edit buffer
- 108635 3. One or more sequential blank lines
- 108636 4. The first character in the edit buffer
- 108637 5. The last non-<newline> in the edit buffer

*section boundary*

A *section boundary* is one of the following:

- 108640 1. A line whose first character is a <form-feed>
- 108641 2. A line whose first character is an open curly brace ( ' { ' )
- 108642 3. A line whose first character is a period and whose second and third characters
- 108643 match a two-character pair in the **sections** edit option (see *ed*)
- 108644 4. A line whose first character is a period and whose only other character matches
- 108645 the first character of a two-character pair in the **sections** edit option, where the
- 108646 second character of the two-character pair is a <space>
- 108647 5. The first line of the edit buffer
- 108648 6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a
- 108649 ]] or } command; otherwise, the last non-<newline> of the last line of the edit
- 108650 buffer

*paragraph boundary*

A *paragraph boundary* is one of the following:

- 108653 1. A section boundary
- 108654 2. A line whose first character is a period and whose second and third characters
- 108655 match a two-character pair in the **paragraphs** edit option (see *ed*)
- 108656 3. A line whose first character is a period and whose only other character matches
- 108657 the first character of a two-character pair in the *paragraphs* edit option, where the
- 108658 second character of the two-character pair is a <space>
- 108659 4. One or more sequential blank lines

*remembered search direction*

See the description of *remembered search direction* in *ed*.

*sentence boundary*

A *sentence boundary* is one of the following:

- 108664 1. A paragraph boundary

- 108665                   2. The first non-<blank> that occurs after a paragraph boundary
- 108666                   3. The first non-<blank> that occurs after a period ( ' . ' ), exclamation mark ( ' ! ' ),
- 108667                   or question mark ( ' ? ' ), followed by two <space>s or the end of a line; any
- 108668                   number of closing parenthesis ( ' ) ' ), closing brackets ( ' ] ' ), double quote ( ' " ' ),
- 108669                   or single quote ( ' ' ' ) characters can appear between the punctuation mark and
- 108670                   the two <space>s or end-of-line

108671                   In the remainder of the description of the *vi* utility, the term “buffer line” refers to a line in the

108672                   edit buffer and the term “display line” refers to the line or lines on the display screen used to

108673                   display one buffer line. The term “current line” refers to a specific “buffer line”.

108674                   If there are display lines on the screen for which there are no corresponding buffer lines because

108675                   they correspond to lines that would be after the end of the file, they shall be displayed as a single

108676                   tilde ( ' ~ ' ) character, plus the terminating <newline>.

108677                   The last line of the screen shall be used to report errors or display informational messages. It

108678                   shall also be used to display the input for “line-oriented commands” ( / , ? , : , and ! ). When a line-

108679                   oriented command is executed, the editor shall enter text input mode on the last line on the

108680                   screen, using the respective command characters as prompt characters. (In the case of the !

108681                   command, the associated motion shall be entered by the user before the editor enters text input

108682                   mode.) The line entered by the user shall be terminated by a <newline>, a

108683                   non-<control>-V-escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more

108684                   characters than require a display width minus one column number of screen columns can be

108685                   entered.

108686                   If any command is executed that overwrites a portion of the screen other than the last line of the

108687                   screen (for example, the *ex suspend* or ! commands), other than the *ex shell* command, the user

108688                   shall be prompted for a character before the screen is refreshed and the edit session continued.

108689                   <tab>s shall take up the number of columns on the screen set by the **tabstop** edit option (see *ed*),

108690                   unless there are less than that number of columns before the display margin that will cause the

108691                   displayed line to be folded; in this case, they shall only take up the number of columns up to

108692                   that boundary.

108693                   The cursor shall be placed on the current line and relative to the current column as specified by

108694                   each command described in the following sections.

108695                   In open mode, if the current line is not already displayed, then it shall be displayed.

108696                   In visual mode, if the current line is not displayed, then the lines that are displayed shall be

108697                   expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be

108698                   displayed. If the screen is redrawn, no more than the number of display lines specified by the

108699                   value of the **window** edit option shall be displayed (unless the current line cannot be completely

108700                   displayed in the number of display lines specified by the **window** edit option) and the current

108701                   line shall be positioned as close to the center of the displayed lines as possible (within the

108702                   constraints imposed by the distance of the line from the beginning or end of the edit buffer). If

108703                   the current line is before the first line in the display and the screen is scrolled, an unspecified

108704                   portion of the current line shall be placed on the first line of the display. If the current line is after

108705                   the last line in the display and the screen is scrolled, an unspecified portion of the current line

108706                   shall be placed on the last line of the display.

108707                   In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into

108708                   the lines at the bottom of the display that are available for its presentation, the editor may choose

108709                   not to display any portion of the line. The lines of the display that do not contain text from the

108710                   edit buffer for this reason shall each consist of a single '@' character.

108711                   In visual mode, the editor may choose for unspecified reasons to not update lines in the display

108712                   to correspond to the underlying edit buffer text. The lines of the display that do not correctly

108713                   correspond to text from the edit buffer for this reason shall consist of a single '@' character (plus

108714 the terminating <newline>), and the <control>-R command shall cause the editor to update the  
108715 screen to correctly represent the edit buffer.

108716 Open and visual mode commands that set the current column set it to a column position in the  
108717 display, and not a character position in the line. In this case, however, the column position in the  
108718 display shall be calculated for an infinite width display; for example, the column related to a  
108719 character that is part of a line that has been folded onto additional screen lines will be offset from  
108720 the display line column where the buffer line begins, not from the beginning of a particular  
108721 display line.

108722 The display cursor column in the display is based on the value of the current column, as follows,  
108723 with each rule applied in turn:

- 108724 1. If the current column is after the last display line column used by the displayed line, the  
108725 display cursor column shall be set to the last display line column occupied by the last  
108726 non-<newline> in the current line; otherwise, the display cursor column shall be set to the  
108727 current column.
- 108728 2. If the character of which some portion is displayed in the display line column specified by  
108729 the display cursor column requires more than a single display line column:
  - 108730 a. If in text input mode, the display cursor column shall be adjusted to the first  
108731 display line column in which any portion of that character is displayed.
  - 108732 b. Otherwise, the display cursor column shall be adjusted to the last display line  
108733 column in which any portion of that character is displayed.

108734 The current column shall not be changed by these adjustments to the display cursor column.

108735 If an error occurs during the parsing or execution of a *vi* command:

- 108736 • The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for  
108737 example, the current line and column) shall not be further modified.
- 108738 • Unless otherwise specified by the following command sections, it is unspecified whether  
108739 an informational message shall be displayed.
- 108740 • Any partially entered *vi* command shall be discarded.
- 108741 • If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion  
108742 shall be discarded, except as otherwise specified by the **map** command (see *ed*).
- 108743 • If the *vi* command resulted from the execution of a buffer, no further commands caused by  
108744 the execution of the buffer shall be executed.

## 108745 Page Backwards

108746 *Synopsis:* [count] <control>-B

108747 If in open mode, the <control>-B command shall behave identically to the **z** command.  
108748 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

108749 If the **window** edit option is less than 3, display a screen where the last line of the display shall  
108750 be some portion of:

108751 *(current first line) -1*

108752 otherwise, display a screen where the first line of the display shall be some portion of:

108753 *(current first line) - count x ((window edit option) -2)*

108754 If this calculation would result in a line that is before the first line of the edit buffer, the first line  
108755 of the display shall display some portion of the first line of the edit buffer.

108756 *Current line:* If no lines from the previous display remain on the screen, set to the last line of the

108757 display; otherwise, set to (*line* – the number of new lines displayed on this screen).

108758 *Current column*: Set to non-`<blank>`.

### 108759 **Scroll Forward**

108760 *Synopsis*: [*count*] `<control>-D`

108761 If the current line is the last line of the edit buffer, it shall be an error.

108762 If no *count* is specified, *count* shall default to the *count* associated with the previous `<control>-D`  
108763 or `<control>-U` command. If there was no previous `<control>-D` or `<control>-U` command, *count*  
108764 shall default to the value of the **scroll** edit option.

108765 If in open mode, write lines starting with the line after the current line, until *count* lines or the  
108766 last line of the file have been written.

108767 *Current line*: If the current line + *count* is past the last line of the edit buffer, set to the last line of  
108768 the edit buffer; otherwise, set to the current line + *count*.

108769 *Current column*: Set to non-`<blank>`.

### 108770 **Scroll Forward by Line**

108771 *Synopsis*: [*count*] `<control>-E`

108772 Display the line count lines after the last line currently displayed.

108773 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines  
108774 after the last line currently displayed, the last line of the display shall display some portion of  
108775 the last line of the edit buffer.

108776 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first  
108777 line displayed.

108778 *Current column*: Unchanged.

### 108779 **Page Forward**

108780 *Synopsis*: [*count*] `<control>-F`

108781 If in open mode, the `<control>-F` command shall behave identically to the **z** command.  
108782 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

108783 If the **window** edit option is less than 3, display a screen where the first line of the display shall  
108784 be some portion of:

108785 (*current last line*) +1

108786 otherwise, display a screen where the first line of the display shall be some portion of:

108787 (*current first line*) + *count* x ((*window edit option*) - 2)

108788 If this calculation would result in a line that is after the last line of the edit buffer, the last line of  
108789 the display shall display some portion of the last line of the edit buffer.

108790 *Current line*: If no lines from the previous display remain on the screen, set to the first line of the  
108791 display; otherwise, set to (*line* + the number of new lines displayed on this screen).

108792 *Current column*: Set to non-`<blank>`.

108793 **Display Information**108794 *Synopsis:* <control>-G108795 This command shall be equivalent to the *ex file* command.108796 **Move Cursor Backwards**108797 *Synopsis:* [*count*] <control>-H108798 [*count*] h108799 the current *erase* character (see *stty*)

108800 If there are no characters before the current character on the current line, it shall be an error. If  
 108801 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
 108802 number of previous characters on the line.

108803 If used as a motion command:

- 108804 1. The text region shall be from the character before the starting cursor up to and including  
 108805 the *count*th character before the starting cursor.
- 108806 2. Any text copied to a buffer shall be in character mode.

108807 If not used as a motion command:

108808 *Current line:* Unchanged.

108809 *Current column:* Set to (*column* – the number of columns occupied by *count* characters ending  
 108810 with the previous current column).

108811 **Move Down**108812 *Synopsis:* [*count*] <newline>108813 [*count*] <control>-J108814 [*count*] <control>-M108815 [*count*] <control>-N108816 [*count*] j108817 [*count*] <carriage-return>108818 [*count*] +108819 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

108820 If used as a motion command:

- 108821 1. The text region shall include the starting line and the next *count* – 1 lines.
- 108822 2. Any text copied to a buffer shall be in line mode.

108823 If not used as a motion command:

108824 *Current line:* Set to *current line*+ *count*.

108825 *Current column:* Set to non-<blank> for the <carriage-return>, <control>-M, and + commands;  
 108826 otherwise, unchanged.

108827 **Clear and Redisplay**108828 *Synopsis:* <control>-L108829 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay  
108830 the screen.108831 *Current line:* Unchanged.108832 *Current column:* Unchanged.108833 **Move Up**108834 *Synopsis:* [count] <control>-P

108835 [count] k

108836 [count] -

108837 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

108838 If used as a motion command:

- 108839 1. The text region shall include the starting line and the previous *count* lines.
- 108840 2. Any text copied to a buffer shall be in line mode.

108841 If not used as a motion command:

108842 *Current line:* Set to *current line* - *count*.108843 *Current column:* Set to non-<blank> for the - command; otherwise, unchanged.108844 **Redraw Screen**108845 *Synopsis:* <control>-R108846 If any lines have been deleted from the display screen and flagged as deleted on the terminal  
108847 using the @ convention (see the beginning of the EXTENDED DESCRIPTION section), they shall  
108848 be redisplayed to match the contents of the edit buffer.108849 It is unspecified whether lines flagged with @ because they do not fit on the terminal display  
108850 shall be affected.108851 *Current line:* Unchanged.108852 *Current column:* Unchanged.108853 **Scroll Backward**108854 *Synopsis:* [count] <control>-U

108855 If the current line is the first line of the edit buffer, it shall be an error.

108856 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D  
108857 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*  
108858 shall default to the value of the **scroll** edit option.108859 *Current line:* If *count* is greater than the current line, set to 1; otherwise, set to the current line -  
108860 *count*.108861 *Current column:* Set to non-<blank>.

108862

**Scroll Backward by Line**

108863

*Synopsis:* [count] <control>-Y

108864

Display the line *count* lines before the first line currently displayed.

108865

If the current line is the first line of the edit buffer, it shall be an error. If this calculation would result in a line that is before the first line of the edit buffer, the first line of the display shall display some portion of the first line of the edit buffer.

108866

108867

*Current line:* Unchanged if the previous current character is displayed; otherwise, set to the first line displayed.

108868

108869

*Current column:* Unchanged.

108870

108871

**Edit the Alternate File**

108872

*Synopsis:* <control>-^

108873

This command shall be equivalent to the *ex* **edit** command, with the alternate pathname as its argument.

108874

108875

**Terminate Command or Input Mode**

108876

*Synopsis:* <ESC>

108877

If a partial *vi* command (as defined by at least one, non-*count* character) has been entered, discard the *count* and the command character(s).

108878

108879

Otherwise, if no command characters have been entered, and the <ESC> was the result of a map expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall not be an error.

108880

108881

108882

Otherwise, it shall be an error.

108883

*Current line:* Unchanged.

108884

*Current column:* Unchanged.

108885

**Search for tagstring**

108886

*Synopsis:* <control>-]

108887

If the current character is not a word or <blank>, it shall be an error.

108888

This command shall be equivalent to the *ex* **tag** command, with the argument to that command defined as follows.

108889

108890

If the current character is a <blank>:

108891

1. Skip all <blank>s after the cursor up to the end of the line.

108892

2. If the end of the line is reached, it shall be an error.

108893

Then, the argument to the *ex* **tag** command shall be the current character and all subsequent characters, up to the first non-word character or the end of the line.

108894



108895  
108896  
108897  
108898  
108899  
108900  
108901  
108902  
108903  
108904  
108905  
108906  
108907  
108908  
108909  
108910  
108911  
108912  
108913  
108914  
108915  
108916  
108917  
108918  
108919  
108920  
108921  
108922  
108923  
108924  
108925  
108926  
108927  
108928  
108929  
108930  
108931  
108932  
108933  
108934

### Move Cursor Forward

*Synopsis:*    [*count*] <space>  
                  [*count*] 1 (ell)

If there are less than *count* non-`<newline>`s after the cursor on the current line, *count* shall be adjusted to the number of non-`<newline>`s after the cursor on the line.

If used as a motion command:

1. If the current or *count*th character after the cursor is the last non-`<newline>` in the line, the text region shall be comprised of the current character up to and including the last non-`<newline>` in the line. Otherwise, the text region shall be from the current character up to, but not including, the *count*th character after the cursor.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

If there are no non-`<newline>`s after the current character on the current line, it shall be an error.

*Current line:* Unchanged.

*Current column:* Set to the last column that displays any portion of the *count*th character after the current character.

### Replace Text with Results from Shell Command

*Synopsis:*    [*count*] ! *motion shell-commands* <newline>

If the motion command is the ! command repeated:

1. If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent of the `ex :read !` command, with the text input, and no text shall be copied to any buffer.
2. Otherwise:
  - a. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an error.
  - b. The text region shall be from the current line up to and including the next *count* -1 lines.

Otherwise, the text region shall be the lines in which any character of the text region specified by the motion command appear.

Any text copied to a buffer shall be in line mode.

This command shall be equivalent to the `ex !` command for the specified lines.

### Move Cursor to End-of-Line

*Synopsis:*    [*count*] \$

It shall be an error if there are less than (*count* -1) lines after the current line in the edit buffer.

If used as a motion command:

1. If *count* is 1:
  - a. It shall be an error if the line is empty.
  - b. Otherwise, the text region shall consist of all characters from the starting cursor to the last non-`<newline>` in the line, inclusive, and any text copied to a buffer shall be in character mode.

- 108935 2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line,  
108936 the text region shall consist of the current and the next *count* -1 lines, and any text saved  
108937 to a buffer shall be in line mode.
- 108938 3. Otherwise, the text region shall consist of all characters from the starting cursor to the last  
108939 non-<newline> in the line that is *count* -1 lines forward from the current line, and any text  
108940 copied to a buffer shall be in character mode.

108941 If not used as a motion command:

108942 *Current line*: Set to the *current line* + *count*-1.

108943 *Current column*: The current column is set to the last display line column of the last  
108944 non-<newline> in the line, or column position 1 if the line is empty.

108945 The current column shall be adjusted to be on the last display line column of the last  
108946 non-<newline> of the current line as subsequent commands change the current line, until a  
108947 command changes the current column.

### 108948 **Move to Matching Character**

108949 *Synopsis*: %

108950 If the character at the current position is not a parenthesis, bracket, or curly brace, search  
108951 forward in the line to the first one of those characters. If no such character is found, it shall be an  
108952 error.

108953 The matching character shall be the parenthesis, bracket, or curly brace matching the  
108954 parenthesis, bracket, or curly brace, respectively, that was at the current position or that was  
108955 found on the current line.

108956 Matching shall be determined as follows, for an open parenthesis:

- 108957 1. Set a counter to 1.
- 108958 2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.
- 108959 3. If the end of the edit buffer is reached, it shall be an error.
- 108960 4. If an open parenthesis is found, increment the counter by 1.
- 108961 5. If a close parenthesis is found, decrement the counter by 1.
- 108962 6. If the counter is zero, the current character is the matching character.

108963 Matching for a close parenthesis shall be equivalent, except that the search shall be backwards,  
108964 from the starting character to the beginning of the buffer, a close parenthesis shall increment the  
108965 counter by 1, and an open parenthesis shall decrement the counter by 1.

108966 Matching for brackets and curly braces shall be equivalent, except that searching shall be done  
108967 for open and close brackets or open and close curly braces. It is implementation-defined whether  
108968 other characters are searched for and matched as well.

108969 If used as a motion command:

- 108970 1. If the matching cursor was after the starting cursor in the edit buffer, and the starting  
108971 cursor position was at or before the first non-<blank> non-<newline> in the starting line,  
108972 and the matching cursor position was at or after the last non-<blank> non-<newline> in  
108973 the matching line, the text region shall consist of the current line to the matching line,  
108974 inclusive, and any text copied to a buffer shall be in line mode.
- 108975 2. If the matching cursor was before the starting cursor in the edit buffer, and the starting  
108976 cursor position was at or after the last non-<blank> non-<newline> in the starting line,  
108977 and the matching cursor position was at or before the first non-<blank> non-<newline> in

108978 the matching line, the text region shall consist of the current line to the matching line,  
108979 inclusive, and any text copied to a buffer shall be in line mode.

108980 3. Otherwise, the text region shall consist of the starting character to the matching character,  
108981 inclusive, and any text copied to a buffer shall be in character mode.

108982 If not used as a motion command:

108983 *Current line*: Set to the line where the matching character is located.

108984 *Current column*: Set to the last column where any portion of the matching character is displayed.

### 108985 Repeat Substitution

108986 *Synopsis*: &

108987 Repeat the previous substitution command. This command shall be equivalent to the *ex &*  
108988 command with the current line as its addresses, and without *options*, *count*, or *flags*.

### 108989 Return to Previous Context at Beginning of Line

108990 *Synopsis*: ' *character*

108991 It shall be an error if there is no line in the edit buffer marked by *character*.

108992 If used as a motion command:

- 108993 1. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
108994 and the marked cursor in the edit buffer shall be logically swapped.
- 108995 2. The text region shall consist of the starting line up to and including the marked line, and  
108996 any text copied to a buffer shall be in line mode.

108997 If not used as a motion command:

108998 *Current line*: Set to the line referenced by the mark.

108999 *Current column*: Set to non-<blank>.

### 109000 Return to Previous Context

109001 *Synopsis*: ' *character*

109002 It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer  
109003 contains a character in the saved numbered character position, it shall be as if the marked  
109004 position is the first non-<blank>.

109005 If used as a motion command:

- 109006 1. It shall be an error if the marked cursor references the same character in the edit buffer as  
109007 the starting cursor.
- 109008 2. If the starting cursor is after the marked cursor, then the locations of the starting cursor  
109009 and the marked cursor in the edit buffer shall be logically swapped.
- 109010 3. If the starting line is empty or the starting cursor is at or before the first non-<blank>  
109011 non-<newline> of the starting line, and the marked cursor line is empty or the marked  
109012 cursor references the first character of the marked cursor line, the text region shall consist  
109013 of all lines containing characters from the starting cursor to the line before the marked  
109014 cursor line, inclusive, and any text copied to a buffer shall be in line mode.
- 109015 4. Otherwise, if the marked cursor line is empty or the marked cursor references a character  
109016 at or before the first non-<blank> non-<newline> of the marked cursor line, the region of  
109017 text shall be from the starting cursor to the last non-<newline> of the line before the  
109018 marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.

109019 5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked  
109020 cursor (exclusive), and any text copied to a buffer shall be in character mode.

109021 If not used as a motion command:

109022 *Current line*: Set to the line referenced by the mark.

109023 *Current column*: Set to the last column in which any portion of the character referenced by the  
109024 mark is displayed.

### 109025 **Return to Previous Section**

109026 *Synopsis*: [ *count* ] [ [

109027 Move the cursor backward through the edit buffer to the first character of the previous section  
109028 boundary, *count* times.

109029 If used as a motion command:

109030 1. If the starting cursor was at the first character of the starting line or the starting line was  
109031 empty, and the first character of the boundary was the first character of the boundary line,  
109032 the text region shall consist of the current line up to and including the line where the  
109033 *count*th next boundary starts, and any text copied to a buffer shall be in line mode.

109034 2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last  
109035 line of the edit buffer, the text region shall consist of the last character in the edit buffer up  
109036 to and including the starting character, and any text saved to a buffer shall be in character  
109037 mode.

109038 3. Otherwise, the text region shall consist of the starting character up to but not including  
109039 the first character in the *count*th next boundary, and any text copied to a buffer shall be in  
109040 character mode.

109041 If not used as a motion command:

109042 *Current line*: Set to the line where the *count*th next boundary in the edit buffer starts.

109043 *Current column*: Set to the last column in which any portion of the first character of the *count*th  
109044 next boundary is displayed, or column position 1 if the line is empty.

### 109045 **Move to Next Section**

109046 *Synopsis*: [ *count* ] ] ]

109047 Move the cursor forward through the edit buffer to the first character of the next section  
109048 boundary, *count* times.

109049 If used as a motion command:

109050 1. If the starting cursor was at the first character of the starting line or the starting line was  
109051 empty, and the first character of the boundary was the first character of the boundary line,  
109052 the text region shall consist of the current line up to and including the line where the  
109053 *count*th previous boundary starts, and any text copied to a buffer shall be in line mode.

109054 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first  
109055 character in the edit buffer up to but not including the starting character, and any text  
109056 copied to a buffer shall be in character mode.

109057 3. Otherwise, the text region shall consist of the first character in the *count*th previous  
109058 section boundary up to but not including the starting character, and any text copied to a  
109059 buffer shall be in character mode.

109060 If not used as a motion command:

109061 *Current line*: Set to the line where the *count*th previous boundary in the edit buffer starts.

109062 *Current column*: Set to the last column in which any portion of the first character of the *count*th  
109063 previous boundary is displayed, or column position 1 if the line is empty.

#### 109064 **Move to First Non-<blank> Position on Current Line**

109065 *Synopsis*:     `^`

109066 If used as a motion command:

- 109067 1. If the line has no non-<blank> non-<newline>s, or if the cursor is at the first non-<blank>  
109068 non-<newline> of the line, it shall be an error.
- 109069 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region  
109070 shall be comprised of the current character, up to, but not including, the first non-<blank>  
109071 non-<newline> of the line.
- 109072 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall  
109073 be from the character before the starting cursor up to and including the first non-<blank>  
109074 non-<newline> of the line.
- 109075 4. Any text copied to a buffer shall be in character mode.

109076 If not used as a motion command:

109077 *Current line*: Unchanged.

109078 *Current column*: Set to non-<blank>.

#### 109079 **Current and Line Above**

109080 *Synopsis*:     `[count] _`

109081 If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an error.

109082 If used as a motion command:

- 109083 1. If *count* is less than 2, the text region shall be the current line.
- 109084 2. Otherwise, the text region shall include the starting line and the next *count* - 1 lines.
- 109085 3. Any text copied to a buffer shall be in line mode.

109086 If not used as a motion command:

109087 *Current line*: Set to current line + *count* - 1.

109088 *Current column*: Set to non-<blank>.

#### 109089 **Move Back to Beginning of Sentence**

109090 *Synopsis*:     `[count] (`

109091 Move backward to the beginning of a sentence. This command shall be equivalent to the `[]`  
109092 command, with the exception that sentence boundaries shall be used instead of section  
109093 boundaries.

109094 **Move Forward to Beginning of Sentence**109095 *Synopsis:* [ *count* ] )

109096 Move forward to the beginning of a sentence. This command shall be equivalent to the ]] command, with the exception that sentence boundaries shall be used instead of section boundaries.

109097

109098

109099 **Move Back to Preceding Paragraph**109100 *Synopsis:* [ *count* ] {

109101 Move back to the beginning of the preceding paragraph. This command shall be equivalent to the [[ command, with the exception that paragraph boundaries shall be used instead of section boundaries.

109102

109103

109104 **Move Forward to Next Paragraph**109105 *Synopsis:* [ *count* ] }

109106 Move forward to the beginning of the next paragraph. This command shall be equivalent to the ]] command, with the exception that paragraph boundaries shall be used instead of section boundaries.

109107

109108

109109 **Move to Specific Column Position**109110 *Synopsis:* [ *count* ] |

109111 For the purposes of this command, lines that are too long for the current display and that have been folded shall be treated as having a single, 1-based, number of columns.

109112

109113 If there are less than *count* columns in which characters from the current line are displayed on the screen, *count* shall be adjusted to be the last column in which any portion of the line is displayed on the screen.

109114

109115

109116 If used as a motion command:

- 109117 1. If the line is empty, or the cursor character is the same as the character on the *count*th column of the line, it shall be an error.
- 109118
- 109119 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of the current character, up to but not including the character on the *count*th column of the line.
- 109120
- 109121
- 109122 3. If the cursor is after the *count*th column of the line, the text region shall be from the character before the starting cursor up to and including the character on the *count*th column of the line.
- 109123
- 109124
- 109125 4. Any text copied to a buffer shall be in character mode.

109126 If not used as a motion command:

109127 *Current line:* Unchanged.

109128 *Current column:* Set to the last column in which any portion of the character that is displayed in the *count* column of the line is displayed.

109129

109130 **Reverse Find Character**109131 *Synopsis:* [ *count* ] ,109132 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or  
109133 **T** command, respectively, with the specified *count* and the same search character.109134 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.109135 **Repeat**109136 *Synopsis:* [ *count* ] .109137 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall  
109138 be an error if none of these commands have been executed. Commands (other than commands  
109139 that enter text input mode) executed as a result of map expansions, shall not change the value of  
109140 the last repeatable command.109141 Repeated commands with associated motion commands shall repeat the motion command as  
109142 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the  
109143 repeated command or its associated motion command.109144 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall  
109145 not set the remembered search character for the **;** and **,** commands.109146 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric  
109147 buffer named with a number less than 9, the buffer associated with the repeated command shall  
109148 be set to be the buffer named by the name of the previous buffer logically incremented by 1.109149 If the repeated character is a text input command, the input text associated with that command  
109150 is repeated literally:

- 109151
- Input characters are neither macro or abbreviation-expanded.
  - Input characters are not interpreted in any special way with the exception that **<newline>**,  
109152 **<carriage-return>**, and **<control>-T** behave as described in [Input Mode Commands in vi](#)  
109153 (on page 3248).

109154 *Current line:* Set as described for the repeated command.109155 *Current column:* Set as described for the repeated command.109156 **Find Regular Expression**109157 *Synopsis:* /109158 If the input line contains no non-**<newline>**s, it shall be equivalent to a line containing only the  
109159 last regular expression encountered. The enhanced regular expressions supported by *vi* are  
109160 described in [Regular Expressions in ex](#) (on page 2608).  
109161109162 Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed  
109163 by an address offset or a *vi* **z** command.109164 If the regular expression is not the last regular expression on the line, or if a line offset or **z**  
109165 command is specified, the regular expression shall be terminated by an unescaped **'/'**  
109166 character, which shall not be used as part of the regular expression. If the regular expression is  
109167 not the first regular expression on the line, it shall be preceded by zero or more **<blank>**s, a  
109168 semicolon, zero or more **<blank>**s, and a leading **'/'** character, which shall not be interpreted as  
109169 part of the regular expression. It shall be an error to precede any regular expression with any  
109170 characters other than these.109171 Each search shall begin from the character after the first character of the last match (or, if it is the  
109172 first search, after the cursor). If the **wrapsan** edit option is set, the search shall continue to the  
109173 character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be

109174 an error if any search fails to find a match, and an informational message to this effect shall be  
109175 displayed.

109176 An optional address offset (see [Addressing in ex](#), on page 2579) can be specified after the last  
109177 regular expression by including a trailing '/' character after the regular expression and  
109178 specifying the address offset. This offset will be from the line containing the match for the last  
109179 regular expression specified. It shall be an error if the line offset would indicate a line address  
109180 less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be  
109181 supported. It shall be an error to follow the address offset with any other characters than  
109182 <blank>s.

109183 If not used as a motion command, an optional **z** command (see [Redraw Window](#), on page 3247)  
109184 can be specified after the last regular expression by including a trailing '/' character after the  
109185 regular expression, zero or more <blank>s, a 'z', zero or more <blank>s, an optional new  
109186 **window** edit option value, zero or more <blank>s, and a location character. The effect shall be as  
109187 if the **z** command was executed after the / command. It shall be an error to follow the **z**  
109188 command with any other characters than <blank>s.

109189 The remembered search direction shall be set to forward.

109190 If used as a motion command:

- 109191 1. It shall be an error if the last match references the same character in the edit buffer as the  
109192 starting cursor.
- 109193 2. If any address offset is specified, the last match shall be adjusted by the specified offset as  
109194 described previously.
- 109195 3. If the starting cursor is after the last match, then the locations of the starting cursor and  
109196 the last match in the edit buffer shall be logically swapped.
- 109197 4. If any address offset is specified, the text region shall consist of all lines containing  
109198 characters from the starting cursor to the last match line, inclusive, and any text copied to  
109199 a buffer shall be in line mode.
- 109200 5. Otherwise, if the starting line is empty or the starting cursor is at or before the first  
109201 non-<blank> non-<newline> of the starting line, and the last match line is empty or the  
109202 last match starts at the first character of the last match line, the text region shall consist of  
109203 all lines containing characters from the starting cursor to the line before the last match  
109204 line, inclusive, and any text copied to a buffer shall be in line mode.
- 109205 6. Otherwise, if the last match line is empty or the last match begins at a character at or  
109206 before the first non-<blank> non-<newline> of the last match line, the region of text shall  
109207 be from the current cursor to the last non-<newline> of the line before the last match line,  
109208 inclusive, and any text copied to a buffer shall be in character mode.
- 109209 7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first  
109210 character of the last match (exclusive), and any text copied to a buffer shall be in character  
109211 mode.

109212 If not used as a motion command:

109213 *Current line*: If a match is found, set to the last matched line plus the address offset, if any;  
109214 otherwise, unchanged.

109215 *Current column*: Set to the last column on which any portion of the first character in the last  
109216 matched string is displayed, if a match is found; otherwise, unchanged.



109217 **Move to First Character in Line**109218 *Synopsis:* 0 (zero)109219 Move to the first character on the current line. The character '0' shall not be interpreted as a  
109220 command if it is immediately preceded by a digit.

109221 If used as a motion command:

- 109222 1. If the cursor character is the first character in the line, it shall be an error.
- 109223 2. The text region shall be from the character before the cursor character up to and including  
109224 the first character in the line.
- 109225 3. Any text copied to a buffer shall be in character mode.

109226 If not used as a motion command:

109227 *Current line:* Unchanged.109228 *Current column:* The last column in which any portion of the first character in the line is  
109229 displayed, or if the line is empty, unchanged.109230 **Execute an ex Command**109231 *Synopsis:* :109232 Execute one or more *ex* commands.

109233 If any portion of the screen other than the last line of the screen was overwritten by any *ex*  
109234 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from  
109235 the user, and shall then read a character. This action may also be taken for other, unspecified  
109236 reasons.

109237 If the next character entered is a ':', another *ex* command shall be accepted and executed. Any  
109238 other character shall cause the screen to be refreshed and *vi* shall return to command mode.

109239 *Current line:* As specified for the *ex* command.109240 *Current column:* As specified for the *ex* command.109241 **Repeat Find**109242 *Synopsis:* [count] ;

109243 This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and  
109244 with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,  
109245 **f**, **T**, or **t** command, it shall be an error.

109246 **Shift Left**109247 *Synopsis:* [count] < *motion*

109248 If the motion command is the &lt; command repeated:

- 109249 1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
109250 error.
- 109251 2. The text region shall be from the current line, up to and including the next *count* -1 lines.

109252 Shift any line in the text region specified by the *count* and motion command one shiftwidth (see  
109253 the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The  
109254 unshifted lines shall be copied to the unnamed buffer in line mode.

109255 *Current line:* If the motion was from the current cursor position toward the end of the edit buffer,  
109256 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region

109257 specified by the motion command.

109258 *Current column*: Set to non-<blank>.

### 109259 **Shift Right**

109260 *Synopsis*: [count] > motion

109261 If the motion command is the > command repeated:

- 109262 1. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an  
109263 error.
- 109264 2. The text region shall be from the current line, up to and including the next *count* -1 lines.

109265 Shift any line with characters in the text region specified by the *count* and motion command one  
109266 shiftwidth (see the *ex shiftwidth* option) away from the start of the line, as described by the *ex* >  
109267 command. The unshifted lines shall be copied into the unnamed buffer in line mode.

109268 *Current line*: If the motion was from the current cursor position toward the end of the edit buffer,  
109269 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
109270 specified by the motion command.

109271 *Current column*: Set to non-<blank>.

### 109272 **Scan Backwards for Regular Expression**

109273 *Synopsis*: ?

109274 Scan backwards; the ? command shall be equivalent to the / command (see [Find Regular](#)  
109275 [Expression](#), on page 3231) with the following exceptions:

- 109276 1. The input prompt shall be a '? '.
- 109277 2. Each search shall begin from the character before the first character of the last match (or, if  
109278 it is the first search, the character before the cursor character).
- 109279 3. The search direction shall be from the cursor toward the beginning of the edit buffer, and  
109280 the **wrapsan** edit option shall affect whether the search wraps to the end of the edit  
109281 buffer and continues.
- 109282 4. The remembered search direction shall be set to backward.

### 109283 **Execute**

109284 *Synopsis*: @buffer

109285 If the *buffer* is specified as @, the last buffer executed shall be used. If no previous buffer has been  
109286 executed, it shall be an error.

109287 Behave as if the contents of the named buffer were entered as standard input. After each line of a  
109288 line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline>  
109289 were entered as standard input.

109290 If an error occurs during this process, an error message shall be written, and no more characters  
109291 resulting from the execution of this command shall be processed.

109292 If a *count* is specified, behave as if that count were entered as user input before the characters  
109293 from the @ buffer were entered.

109294 *Current line*: As specified for the individual commands.

109295 *Current column*: As specified for the individual commands.

109296 **Reverse Case**109297 *Synopsis:* [count] ~

109298 Reverse the case of the current character and the next *count* -1 characters, such that lowercase  
 109299 characters that have uppercase counterparts shall be changed to uppercase characters, and  
 109300 uppercase characters that have lowercase counterparts shall be changed to lowercase characters,  
 109301 as prescribed by the current locale. No other characters shall be affected by this command.

109302 If there are less than *count* -1 characters after the cursor in the edit buffer, *count* shall be adjusted  
 109303 to the number of characters after the cursor in the edit buffer minus 1.

109304 For the purposes of this command, the next character after the last non-<newline> on the line  
 109305 shall be the next character in the edit buffer.

109306 *Current line:* Set to the line including the (*count*-1)th character after the cursor.

109307 *Current column:* Set to the last column in which any portion of the (*count*-1)th character after the  
 109308 cursor is displayed.

109309 **Append**109310 *Synopsis:* [count] a

109311 Enter text input mode after the current cursor position. No characters already in the edit buffer  
 109312 shall be affected by this command. A *count* shall cause the input text to be appended *count* -1  
 109313 more times to the end of the input.

109314 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
 109315 on page 3248).

109316 **Append at End-of-Line**109317 *Synopsis:* [count] A

109318 This command shall be equivalent to the *vi* command:

109319 \$ [count] a

109320 (see [Append](#)).109321 **Move Backward to Preceding Word**109322 *Synopsis:* [count] b

109323 With the exception that words are used as the delimiter instead of bigwords, this command shall  
 109324 be equivalent to the **B** command.

109325 **Move Backward to Preceding Bigword**109326 *Synopsis:* [count] B

109327 If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an  
 109328 error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count*  
 109329 shall be adjusted to the number of bigword beginnings between the cursor and the start of the  
 109330 edit buffer.

109331 If used as a motion command:

- 109332 1. The text region shall be from the first character of the *count*th previous bigword beginning  
 109333 up to but not including the cursor character.

109334 2. Any text copied to a buffer shall be in character mode.

109335 If not used as a motion command:

109336 *Current line*: Set to the line containing the *current column*.

109337 *Current column*: Set to the last column upon which any part of the first character of the *count*th  
109338 previous bigword is displayed.

### 109339 Change

109340 *Synopsis*: `[buffer][count] c motion`

109341 If the motion command is the `c` command repeated:

- 109342 1. The buffer text shall be in line mode.
- 109343 2. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an  
109344 error.
- 109345 3. The text region shall be from the current line up to and including the next *count* - 1 lines.

109346 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

109347 The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text  
109348 to be replaced contains characters from more than a single line, or the buffer text is in line mode,  
109349 the replaced text shall be copied into the numeric buffers as well.

109350 If the buffer text is in line mode:

- 109351 1. Any lines that contain characters in the region shall be deleted, and the editor shall enter  
109352 text input mode at the beginning of a new line which shall replace the first line deleted.
- 109353 2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**  
109354 characters on the first line deleted shall be inserted as if entered by the user.

109355 Otherwise, if characters from more than one line are in the region of text:

- 109356 1. The text shall be deleted.
- 109357 2. Any text remaining in the last line in the text region shall be appended to the first line in  
109358 the region, and the last line in the region shall be deleted.
- 109359 3. The editor shall enter text input mode after the last character not deleted from the first  
109360 line in the text region, if any; otherwise, on the first column of the first line in the region.

109361 Otherwise:

- 109362 1. If the glyph for '\$' is smaller than the region, the end of the region shall be marked with  
109363 a '\$'.
- 109364 2. The editor shall enter text input mode, overwriting the region of text.

109365 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),  
109366 on page 3248).

### 109367 Change to End-of-Line

109368 *Synopsis*: `[buffer][count] C`

109369 This command shall be equivalent to the *vi* command:

109370 `[buffer][count] c$`

109371 See the `c` command.

109372

**Delete**

109373

*Synopsis:* `[buffer][count] d motion`

109374

If the motion command is the **d** command repeated:

109375

1. The buffer text shall be in line mode.

109376

2. If there are less than *count* - 1 lines after the current line in the edit buffer, it shall be an error.

109377

109378

3. The text region shall be from the current line up to and including the next *count* - 1 lines.

109379

Otherwise, the buffer text mode and text region shall be as specified by the motion command.

109380

If in open mode, and the current line is deleted, and the line remains on the display, an '@' character shall be displayed as the first glyph of that line.

109381

109382

Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be deleted contains characters from more than a single line, or the buffer text is in line mode, the deleted text shall be copied into the numeric buffers, as well.

109383

109384

109385

*Current line:* Set to the first text region line that appears in the edit buffer, unless that line has been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit buffer is empty.

109386

109387

109388

*Current column:*

109389

1. If the line is empty, set to column position 1.

109390

2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the end of the edit buffer:

109391

109392

a. If a character from the current line is displayed in the current column, set to the last column that displays any portion of that character.

109393

109394

b. Otherwise, set to the last column in which any portion of any character in the line is displayed.

109395

109396

3. Otherwise, if a character is displayed in the column that began the text region, set to the last column that displays any portion of that character.

109397

109398

4. Otherwise, set to the last column in which any portion of any character in the line is displayed.

109399

109400

**Delete to End-of-Line**

109401

*Synopsis:* `[buffer] D`

109402

Delete the text from the current position to the end of the current line; equivalent to the *vi* command:

109403

109404

`[buffer] d$`

109405

**Move to End-of-Word**

109406

*Synopsis:* `[count] e`

109407

With the exception that words are used instead of bigwords as the delimiter, this command shall be equivalent to the **E** command.

109408

109409

**Move to End-of-Bigword**

109410

*Synopsis:* [count] E

109411

If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between the cursor and the end of the edit buffer.

109412

109413

109414

If used as a motion command:

109415

1. The text region shall be from the last character of the *count*th next bigword up to and including the cursor character.

109416

109417

2. Any text copied to a buffer shall be in character mode.

109418

If not used as a motion command:

109419

*Current line:* Set to the line containing the current column.

109420

*Current column:* Set to the last column upon which any part of the last character of the *count*th next bigword is displayed.

109421

109422

**Find Character in Current Line (Forward)**

109423

*Synopsis:* [count] f character

109424

It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

109425

If used as a motion command:

109426

1. The text range shall be from the cursor character up to and including the *count*th occurrence of the specified character after the cursor.

109427

109428

2. Any text copied to a buffer shall be in character mode.

109429

If not used as a motion command:

109430

*Current line:* Unchanged.

109431

*Current column:* Set to the last column in which any portion of the *count*th occurrence of the specified character after the cursor appears in the line.

109432

109433

**Find Character in Current Line (Reverse)**

109434

*Synopsis:* [count] F character

109435

It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

109436

If used as a motion command:

109437

1. The text region shall be from the *count*th occurrence of the specified character before the cursor, up to, but not including the cursor character.

109438

109439

2. Any text copied to a buffer shall be in character mode.

109440

If not used as a motion command:

109441

*Current line:* Unchanged.

109442

*Current column:* Set to the last column in which any portion of the *count*th occurrence of the specified character before the cursor appears in the line.

109443

109444

**Move to Line**

109445

*Synopsis:*    [*count*] G

109446

If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than the last line of the edit buffer, it shall be an error.

109447

109448

If used as a motion command:

109449

1. The text region shall be from the cursor line up to and including the specified line.

109450

2. Any text copied to a buffer shall be in line mode.

109451

If not used as a motion command:

109452

*Current line:* Set to *count* if *count* is specified; otherwise, the last line.

109453

*Current column:* Set to non-<blank>.

109454

**Move to Top of Screen**

109455

*Synopsis:*    [*count*] H

109456

If the beginning of the line *count* greater than the first line of which any portion appears on the display does not exist, it shall be an error.

109457

109458

If used as a motion command:

109459

1. If in open mode, the text region shall be the current line.

109460

2. Otherwise, the text region shall be from the starting line up to and including (the first line of the display + *count* -1).

109461

109462

3. Any text copied to a buffer shall be in line mode.

109463

If not used as a motion command:

109464

If in open mode, this command shall set the current column to non-<blank> and do nothing else.

109465

Otherwise, it shall set the current line and current column as follows.

109466

*Current line:* Set to (the first line of the display + *count* -1).

109467

*Current column:* Set to non-<blank>.

109468

**Insert Before Cursor**

109469

*Synopsis:*    [*count*] i

109470

Enter text input mode before the current cursor position. No characters already in the edit buffer shall be affected by this command. A *count* shall cause the input text to be appended *count* -1 more times to the end of the input.

109471

109472

109473

*Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#), on page 3248).

109474

109475

**Insert at Beginning of Line**

109476

*Synopsis:*    [*count*] I

109477

This command shall be equivalent to the *vi* command  $\wedge[*count*]i$ .

109478

**Join**

109479

*Synopsis:*    [*count*] J

109480

If the current line is the last line in the edit buffer, it shall be an error.

109481

109482

109483

109484

This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex* command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex* command *count* value of *count* -1 for any other value of *count*, except that the current line and column shall be set as follows.

109485

*Current line:* Unchanged.

109486

109487

109488

*Current column:* The last column in which any portion of the character following the last character in the initial line is displayed, or the last non-<newline> in the line if no characters were appended.

109489

**Move to Bottom of Screen**

109490

*Synopsis:*    [*count*] L

109491

109492

If the beginning of the line *count* less than the last line of which any portion appears on the display does not exist, it shall be an error.

109493

If used as a motion command:

109494

109495

109496

109497

109498

1. If in open mode, the text region shall be the current line.
2. Otherwise, the text region shall include all lines from the starting cursor line to (the last line of the display  $-(count - 1)$ ).
3. Any text copied to a buffer shall be in line mode.

109498

If not used as a motion command:

109499

109500

109501

109502

109503

1. If in open mode, this command shall set the current column to non-<blank> and do nothing else.
2. Otherwise, it shall set the current line and current column as follows.

*Current line:* Set to (the last line of the display  $-(count - 1)$ ).*Current column:* Set to non-<blank>.

109504

**Mark Position**

109505

*Synopsis:*    m *letter*

109506

109507

This command shall be equivalent to the *ex* **mark** command with the specified character as an argument.

109508

**Move to Middle of Screen**

109509

*Synopsis:*    M

109510

The middle line of the display shall be calculated as follows:

109511

(the top line of the display) + (((number of lines displayed) + 1) / 2) - 1

109512

If used as a motion command:

109513

109514

109515

1. If in open mode, the text region shall be the current line.
2. Otherwise, the text region shall include all lines from the starting cursor line up to and including the middle line of the display.



109516 3. Any text copied to a buffer shall be in line mode.

109517 If not used as a motion command:

109518 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

109519 Otherwise, it shall set the current line and current column as follows.

109520 *Current line*: Set to the middle line of the display.

109521 *Current column*: Set to non-<blank>.

### 109522 **Repeat Regular Expression Find (Forward)**

109523 *Synopsis*: n

109524 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi /*  
109525 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi ?*  
109526 command with no characters entered by the user.

109527 If the **n** command is used as a motion command for the **!** command, the editor shall not enter  
109528 text input mode on the last line on the screen, and shall behave as if the user entered a single  
109529 '!' character as the text input.

### 109530 **Repeat Regular Expression Find (Reverse)**

109531 *Synopsis*: N

109532 Scan for the next match of the last pattern given to */* or *?*, but in the reverse direction; this is the  
109533 reverse of **n**.

109534 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi ?*  
109535 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi /*  
109536 command with no characters entered by the user. If the **N** command is used as a motion  
109537 command for the **!** command, the editor shall not enter text input mode on the last line on the  
109538 screen, and shall behave as if the user entered a single **!** character as the text input.

### 109539 **Insert Empty Line Below**

109540 *Synopsis*: o

109541 Enter text input mode in a new line appended after the current line. A *count* shall cause the input  
109542 text to be appended *count* -1 more times to the end of the already added text, each time starting  
109543 on a new, appended line.

109544 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),  
109545 on page 3248).

### 109546 **Insert Empty Line Above**

109547 *Synopsis*: O

109548 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input  
109549 text to be appended *count* -1 more times to the end of the already added text, each time starting  
109550 on a new, appended line.

109551 *Current line/column*: As specified for the text input commands (see [Input Mode Commands in vi](#),  
109552 on page 3248).

109553 **Put from Buffer Following**109554 *Synopsis:* `[buffer] p`109555 If no *buffer* is specified, the unnamed buffer shall be used.109556 If the buffer text is in line mode, the text shall be appended below the current line, and each line  
109557 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
109558 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
109559 appended line.109560 If the buffer text is in character mode, the text shall be appended into the current line after the  
109561 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
109562 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the  
109563 already added text, each time starting after the last added character.109564 *Current line:* If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.109565 *Current column:* If the buffer text is in line mode:

- 109566
1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any  
109567 portion of the first non-<blank> in the line is displayed.
  2. If there is no non-<blank> in the first line of the buffer, set to the last column on which  
109568 any portion of the last non-<newline> in the first line of the buffer is displayed.

109570 If the buffer text is in character mode:

- 109571
1. If the text in the buffer is from more than a single line, then set to the last column on  
109572 which any portion of the first character from the buffer is displayed.
  2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any  
109573 portion of the last character from the buffer is displayed.
  3. Otherwise, set to the first column on which any portion of the first character from the  
109574 buffer is displayed.

109577 **Put from Buffer Before**109578 *Synopsis:* `[buffer] P`109579 If no *buffer* is specified, the unnamed buffer shall be used.109580 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of  
109581 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be  
109582 appended *count* -1 more times to the end of the already added text, each time starting on a new,  
109583 appended line.109584 If the buffer text is in character mode, the text shall be inserted into the current line before the  
109585 cursor, and each line of the buffer other than the first and last shall become a new line in the edit  
109586 buffer. A *count* shall cause the buffer text to be appended *count* -1 more times to the end of the  
109587 already added text, each time starting after the last added character.109588 *Current line:* Unchanged.109589 *Current column:* If the buffer text is in line mode:

- 109590
1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any  
109591 portion of that character is displayed.
  2. If there is no non-<blank> in the first line of the buffer, set to the last column on which  
109592 any portion of the last non-<newline> in the first line of the buffer is displayed.

109594 If the buffer text is in character mode:

- 109595 1. If the text in the buffer is from more than a single line, then set to the last column on  
109596 which any portion of the first character from the buffer is displayed.
- 109597 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any  
109598 portion of the last character from the buffer is displayed.
- 109599 3. Otherwise, set to the first column on which any portion of the first character from the  
109600 buffer is displayed.

### 109601 **Enter ex Mode**

109602 *Synopsis:* Q

109603 Leave visual or open mode and enter *ex* command mode.

109604 *Current line:* Unchanged.

109605 *Current column:* Unchanged.

### 109606 **Replace Character**

109607 *Synopsis:* [count] r character

109608 Replace the *count* characters at and after the cursor with the specified character. If there are less  
109609 than *count* non-*<newline>*s at and after the cursor on the line, it shall be an error.

109610 If character is *<control>-V*, any next character other than the *<newline>* shall be stripped of any  
109611 special meaning and used as a literal character.

109612 If character is *<ESC>*, no replacement shall be made and the current line and current column  
109613 shall be unchanged.

109614 If character is *<carriage-return>* or *<newline>*, *count* new lines shall be appended to the current  
109615 line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be  
109616 discarded, and any remaining characters after the cursor in the current line shall be moved to the  
109617 last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same  
109618 number of **autoindent** characters found on the line from which the command was executed.

109619 *Current line:* Unchanged unless the replacement character is a *<carriage-return>* or *<newline>*, in  
109620 which case it shall be set to line + *count*.

109621 *Current column:* Set to the last column position on which a portion of the last replaced character  
109622 is displayed, or if the replacement character caused new lines to be created, set to non-*<blank>*.

### 109623 **Replace Characters**

109624 *Synopsis:* R

109625 Enter text input mode at the current cursor position possibly replacing text on the current line. A  
109626 *count* shall cause the input text to be appended *count* - 1 more times to the end of the input.

109627 *Current line/column:* As specified for the text input commands (see [Input Mode Commands in vi](#),  
109628 on page 3248).

109629 **Substitute Character**109630 *Synopsis:* `[buffer][count] s`109631 This command shall be equivalent to the *vi* command:109632 `[buffer][count] c<space>`109633 **Substitute Lines**109634 *Synopsis:* `[buffer][count] S`109635 This command shall be equivalent to the *vi* command:109636 `[buffer][count] c_`109637 **Move Cursor to Before Character (Forward)**109638 *Synopsis:* `[count] t character`109639 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

109640 If used as a motion command:

- 109641 1. The text region shall be from the cursor up to but not including the *count*th occurrence of
- 109642 the specified character after the cursor.
- 109643 2. Any text copied to a buffer shall be in character mode.

109644 If not used as a motion command:

109645 *Current line:* Unchanged.109646 *Current column:* Set to the last column in which any portion of the character before the *count*th

109647 occurrence of the specified character after the cursor appears in the line.

109648 **Move Cursor to After Character (Reverse)**109649 *Synopsis:* `[count] T character`109650 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

109651 If used as a motion command:

- 109652 1. If the character before the cursor is the specified character, it shall be an error.
- 109653 2. The text region shall be from the character before the cursor up to but not including the
- 109654 *count*th occurrence of the specified character before the cursor.
- 109655 3. Any text copied to a buffer shall be in character mode.

109656 If not used as a motion command:

109657 *Current line:* Unchanged.109658 *Current column:* Set to the last column in which any portion of the character after the *count*th

109659 occurrence of the specified character before the cursor appears in the line.

109660  
109661  
109662  
109663  
109664  
109665  
109666  
109667  
109668  
109669  
109670  
109671  
109672  
109673  
109674  
109675  
109676  
109677  
109678  
109679  
109680  
109681  
109682  
109683  
109684  
109685  
109686  
109687  
109688  
109689  
109690  
109691  
109692  
109693  
109694  
109695  
109696  
109697  
109698

## Undo

*Synopsis:*     u

This command shall be equivalent to the *ex* **undo** command except that the current line and current column shall be set as follows:

*Current line:* Set to the first line added or changed if any; otherwise, move to the line preceding any deleted text if one exists; otherwise, move to line 1.

*Current column:* If undoing an *ex* command, set to the first non-<blank>.

Otherwise, if undoing a text input command:

1. If the command was a **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to the value it held when the text input command was entered.
2. Otherwise, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline>s follow the text deleted from this line, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

1. If text was added or changed, set to the last column in which any portion of the first character added or changed is displayed.
2. If text was deleted, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline>s follow the deleted text, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, set to non-<blank>.

## Undo Current Line

*Synopsis:*     U

Restore the current line to its state immediately before the most recent time that it became the current line.

*Current line:* Unchanged.

*Current column:* Set to the first column in the line in which any portion of the first character in the line is displayed.

## Move to Beginning of Word

*Synopsis:*     [*count*] w

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the **W** command.

## Move to Beginning of Bigword

*Synopsis:*     [*count*] W

If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last bigword in the edit buffer.

If used as a motion command:

- 109699 1. If the associated command is *c*, *count* is 1, and the cursor is on a <blank>, the region of  
109700 text shall be the current character and no further action shall be taken.
- 109701 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer,  
109702 then the command shall succeed, and the region of text shall include the last character of  
109703 the edit buffer.
- 109704 3. If there are <blank>s or an end-of-line that precede the *count*th bigword, and the  
109705 associated command is *c*, the region of text shall be up to and including the last character  
109706 before the preceding <blank>s or end-of-line.
- 109707 4. If there are <blank>s or an end-of-line that precede the bigword, and the associated  
109708 command is *d* or *y*, the region of text shall be up to and including the last <blank> before  
109709 the start of the bigword or end-of-line.
- 109710 5. Any text copied to a buffer shall be in character mode.

109711 If not used as a motion command:

- 109712 1. If the cursor is on the last character of the edit buffer, it shall be an error.

109713 *Current line*: Set to the line containing the current column.

109714 *Current column*: Set to the last column in which any part of the first character of the *count*th next  
109715 bigword is displayed.

#### 109716 Delete Character at Cursor

109717 *Synopsis*: [*buffer*][*count*] x

109718 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the  
109719 unnamed buffer.

109720 If the line is empty, it shall be an error. If there are less than *count* non-<newline>s at and after  
109721 the cursor on the current line, *count* shall be adjusted to the number of non-<newline>s at and  
109722 after the cursor.

109723 *Current line*: Unchanged.

109724 *Current column*: If the line is empty, set to column position 1. Otherwise, if there were *count* or  
109725 less non-<newline>s at and after the cursor on the current line, set to the last column that  
109726 displays any part of the last non-<newline> of the line. Otherwise, unchanged.

#### 109727 Delete Character Before Cursor

109728 *Synopsis*: [*buffer*][*count*] X

109729 Delete the *count* characters before the current character into *buffer*, if specified, and into the  
109730 unnamed buffer.

109731 If there are no characters before the current character on the current line, it shall be an error. If  
109732 there are less than *count* previous characters on the current line, *count* shall be adjusted to the  
109733 number of previous characters on the line.

109734 *Current line*: Unchanged.

109735 *Current column*: Set to (current column – the width of the deleted characters).

109736 **Yank**109737 *Synopsis:* `[buffer][count] y motion`109738 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.109739 If the motion command is the *y* command repeated:

- 109740 1. The buffer shall be in line mode.
- 109741 2. If there are less than *count* -1 lines after the current line in the edit buffer, it shall be an
- 109742 error.
- 109743 3. The text region shall be from the current line up to and including the next *count* -1 lines.

109744 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

109745 *Current line:* If the motion was from the current cursor position toward the end of the edit buffer,  
 109746 unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region  
 109747 specified by the motion command.

109748 *Current column:*

- 109749 1. If the motion was from the current cursor position toward the end of the edit buffer,  
 109750 unchanged.
- 109751 2. Otherwise, if the current line is empty, set to column position 1.
- 109752 3. Otherwise, set to the last column that displays any part of the first character in the file  
 109753 that is part of the text region specified by the motion command.

109754 **Yank Current Line**109755 *Synopsis:* `[buffer][count] Y`109756 This command shall be equivalent to the *vi* command:109757 `[buffer][count] y_`109758 **Redraw Window**109759 If in open mode, the *z* command shall have the Synopsis:109760 *Synopsis:* `[count] z`

109761 If *count* is not specified, it shall default to the **window** edit option -1. The *z* command shall be  
 109762 equivalent to the *ex z* command, with a type character of = and a *count* of *count* -2, except that  
 109763 the current line and current column shall be set as follows, and the **window** edit option shall not  
 109764 be affected. If the calculation for the *count* argument would result in a negative number, the  
 109765 *count* argument to the *ex z* command shall be zero. A blank line shall be written after the last line  
 109766 is written.

109767 *Current line:* Unchanged.109768 *Current column:* Unchanged.109769 If not in open mode, the *z* command shall have the following Synopsis:109770 *Synopsis:* `[line] z [count] character`

109771 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the  
 109772 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

109773 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the  
 109774 *ex window* command), and the screen shall be redrawn.

109775 *line* shall be placed as specified by the following characters:

109776 <newline>, <carriage-return>

109777 Place the beginning of the line on the first line of the display.

109778 . Place the beginning of the line in the center of the display. The middle line of the display  
109779 shall be calculated as described for the **M** command.

109780 – Place an unspecified portion of the line on the last line of the display.

109781 + If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a  
109782 screen where the first line of the display shall be (current last line) +1. If there are no lines  
109783 after the last line in the display, it shall be an error.

109784 ^ If *line* was specified, display a screen where the last line of the display shall contain an  
109785 unspecified portion of the first line of a display that had an unspecified portion of the  
109786 specified line on the last line of the display. If this calculation results in a line before the  
109787 beginning of the edit buffer, display the first screen of the edit buffer.

109788 Otherwise, display a screen where the last line of the display shall contain an unspecified  
109789 portion of (current first line –1). If this calculation results in a line before the beginning of  
109790 the edit buffer, it shall be an error.

109791 *Current line*: If *line* and the '^' character were specified:

109792 1. If the first screen was displayed as a result of the command attempting to display lines  
109793 before the beginning of the edit buffer: if the first screen was already displayed,  
109794 unchanged; otherwise, set to (current first line –1).

109795 2. Otherwise, set to the last line of the display.

109796 If *line* and the '+' character were specified, set to the first line of the display.

109797 Otherwise, if *line* was specified, set to *line*.

109798 Otherwise, unchanged.

109799 *Current column*: Set to non-<blank>.

## 109800 **Exit**

109801 *Synopsis*: `ZZ`

109802 This command shall be equivalent to the `ex xit` command with no addresses, trailing `!`, or  
109803 filename (see the `ex xit` command).

## 109804 **Input Mode Commands in vi**

109805 In text input mode, the current line shall consist of zero or more of the following categories, plus  
109806 the terminating <newline>:

109807 1. Characters preceding the text input entry point

109808 Characters in this category shall not be modified during text input mode.

109809 2. **autoindent** characters

109810 **autoindent** characters shall be automatically inserted into each line that is created in text  
109811 input mode, either as a result of entering a <newline> or <carriage-return> while in text  
109812 input mode, or as an effect of the command itself; for example, **O** or **o** (see the `ex`  
109813 **autoindent** command), as if entered by the user.

109814 It shall be possible to erase **autoindent** characters with the <control>-D command; it is  
109815 unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W  
109816 characters. Erasing any **autoindent** character turns the glyph into erase-columns and



109817 deletes the character from the edit buffer, but does not change its representation on the  
109818 screen.

### 109819 3. Text input characters

109820 Text input characters are the characters entered by the user. Erasing any text input  
109821 character turns the glyph into erase-columns and deletes the character from the edit  
109822 buffer, but does not change its representation on the screen.

109823 Each text input character entered by the user (that does not have a special meaning) shall  
109824 be treated as follows:

- 109825 a. The text input character shall be appended to the last character in the edit buffer  
109826 from the first, second, or third categories.
- 109827 b. If there are no erase-columns on the screen, the text input command was the **R**  
109828 command, and characters in the fifth category from the original line follow the  
109829 cursor, the next such character shall be deleted from the edit buffer. If the  
109830 **slowopen** edit option is not set, the corresponding glyph on the screen shall  
109831 become erase-columns.
- 109832 c. If there are erase-columns on the screen, as many columns as they occupy, or as are  
109833 necessary, shall be overwritten to display the text input character. (If only part of a  
109834 multi-column glyph is overwritten, the remainder shall be left on the screen, and  
109835 continue to be treated as erase-columns; it is unspecified whether the remainder of  
109836 the glyph is modified in any way.)
- 109837 d. If additional display line columns are needed to display the text input character:
  - 109838 1. If the **slowopen** edit option is set, the text input characters shall be  
109839 displayed on subsequent display line columns, overwriting any characters  
109840 displayed in those columns.
  - 109841 2. Otherwise, any characters currently displayed on or after the column on the  
109842 display line where the text input character is to be displayed shall be  
109843 pushed ahead the number of display line columns necessary to display the  
109844 rest of the text input character.

### 109845 4. Erase-columns

109846 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and  
109847 may be overwritten on the screen by subsequent text input characters. When text input  
109848 mode ends, all erase-columns shall no longer appear on the screen.

109849 Erase-columns are initially the region of text specified by the **c** command (see [Change](#), on  
109850 page 3236); however, erasing **autoindent** or text input characters causes the glyphs of the  
109851 erased characters to be treated as erase-columns.

### 109852 5. Characters following the text region for the **c** command, or the text input entry point for 109853 all other commands

109854 Characters in this category shall not be modified during text input mode, except as  
109855 specified in category 3.b. for the **R** text input command, or as <blank>s deleted when a  
109856 <newline> or <carriage-return> is entered.

109857 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was  
109858 created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an  
109859 error, the editor shall behave as if the erasing character was entered immediately after the last  
109860 text input character entered on the previous line, and all of the non-<newline>s on the current  
109861 line shall be treated as erase-columns.

109862 When text input mode is entered, or after a text input mode character is entered (except as

109863 specified for the special characters below), the cursor shall be positioned as follows:

- 109864 1. On the first column that displays any part of the first erase-column, if one exists
- 109865 2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last
- 109866 character in the first, second, or third categories, if one exists
- 109867 3. Otherwise, the first column that displays any part of the first character in the fifth
- 109868 category, if one exists
- 109869 4. Otherwise, the display line column after the last character in the first, second, or third
- 109870 categories, if one exists
- 109871 5. Otherwise, on column position 1

109872 The characters that are updated on the screen during text input mode are unspecified, other than

109873 that the last text input character shall always be updated, and, if the **slowopen** edit option is not

109874 set, the current cursor character shall always be updated.

109875 The following specifications are for command characters entered during text input mode.

## 109876 NUL

109877 *Synopsis:* NUL

109878 If the first character of the text input is a NUL, the most recently input text shall be input as if

109879 entered by the user, and then text input mode shall be exited. The text shall be input literally;

109880 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted

109881 in any special manner. It is unspecified whether implementations shall support more than 256

109882 bytes of remembered input text.

## 109883 <control>-D

109884 *Synopsis:* <control>-D

109885 The <control>-D character shall have no special meaning when in text input mode for a line-

109886 oriented command (see [Command Descriptions in vi](#), on page 3216).

109887 This command need not be supported on block-mode terminals.

109888 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a '0' or

109889 '^' character:

- 109890 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no
- 109891 further action taken.
- 109892 2. Otherwise, the <control>-D character shall have no special meaning.

109893 If the last input character was a '0', the cursor shall be moved to column position 1.

109894 Otherwise, if the last input character was a '^', the cursor shall be moved to column position 1.

109895 In addition, the **autoindent** level for the next input line shall be derived from the same line from

109896 which the **autoindent** level for the current input line was derived.

109897 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the

109898 *ex* **shiftwidth** command) boundary.

109899 All of the glyphs on columns between the starting cursor position and (inclusively) the ending

109900 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on

109901 page 3248).

109902 *Current line:* Unchanged.

109903 *Current column:* Set to 1 if the <control>-D was preceded by a '^' or '0'; otherwise, set to

109904 (column -1) - ((column -2) % **shiftwidth**).

109905           **<control>-H**

109906           *Synopsis:*     <control>-H

109907           If in text input mode for a line-oriented command, and there are no characters to erase, text  
109908           input mode shall be terminated, no further action shall be done for this command, and the  
109909           current line and column shall be unchanged.

109910           If there are characters other than **autoindent** characters that have been input on the current line  
109911           before the cursor, the cursor shall move back one character.

109912           Otherwise, if there are **autoindent** characters on the current line before the cursor, it is  
109913           implementation-defined whether the <control>-H command is an error or if the cursor moves  
109914           back one **autoindent** character.

109915           Otherwise, if the cursor is in column position 1 and there are previous lines that have been  
109916           input, it is implementation-defined whether the <control>-H command is an error or if it is  
109917           equivalent to entering <control>-H after the last input character on the previous input line.

109918           Otherwise, it shall be an error.

109919           All of the glyphs on columns between the starting cursor position and (inclusively) the ending  
109920           cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on  
109921           page 3248).

109922           The current erase character (see *stty*) shall cause an equivalent action to the <control>-H  
109923           command, unless the previously inserted character was a backslash, in which case it shall be as  
109924           if the literal current erase character had been inserted instead of the backslash.

109925           *Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to  
109926           line -1.

109927           *Current column:* Set to the first column that displays any portion of the character backed up over.

109928           **<newline>**

109929           *Synopsis:*     <newline>  
109930                     <carriage-return>  
109931                     <control>-J  
109932                     <control>-M

109933           If input was part of a line-oriented command, text input mode shall be terminated and the  
109934           command shall continue execution with the input provided.

109935           Otherwise, terminate the current line. If there are no characters other than **autoindent** characters  
109936           on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the  
109937           **autoindent** characters in the line are modified by entering these characters.

109938           Continue text input mode on a new line appended after the current line. If the **slowopen** edit  
109939           option is set, the lines on the screen below the current line shall not be pushed down, but the  
109940           first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the  
109941           screen below the current line shall be pushed down.

109942           If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be  
109943           added as a prefix to the line as described by the *ex* **autoindent** edit option.

109944           All columns after the cursor that are erase-columns (as described in [Input Mode Commands in vi](#),  
109945           on page 3248) shall be discarded.

109946           If the **autoindent** edit option is set, all <blank>s immediately following the cursor shall be  
109947           discarded.

109948           All remaining characters after the cursor shall be transferred to the new line, positioned after

109949 any **autoindent** characters.

109950 *Current line*: Set to current line +1.

109951 *Current column*: Set to the first column that displays any portion of the first character after the

109952 **autoindent** characters on the new line, if any, or the first column position after the last

109953 **autoindent** character, if any, or column position 1.

109954 **<control>-T**

109955 *Synopsis*: `<control>-T`

109956 The `<control>-T` character shall have no special meaning when in text input mode for a line-

109957 oriented command (see [Command Descriptions in vi](#), on page 3216).

109958 This command need not be supported on block-mode terminals.

109959 Behave as if the user entered the minimum number of `<blank>`s necessary to move the cursor

109960 forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth** command)

109961 boundary.

109962 *Current line*: Unchanged.

109963 *Current column*: Set to  $column + \text{shiftwidth} - ((column - 1) \% \text{shiftwidth})$ .

109964 **<control>-U**

109965 *Synopsis*: `<control>-U`

109966 If there are characters other than **autoindent** characters that have been input on the current line

109967 before the cursor, the cursor shall move to the first character input after the **autoindent**

109968 characters.

109969 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is

109970 implementation-defined whether the `<control>-U` command is an error or if the cursor moves to

109971 the first column position on the line.

109972 Otherwise, if the cursor is in column position 1 and there are previous lines that have been

109973 input, it is implementation-defined whether the `<control>-U` command is an error or if it is

109974 equivalent to entering `<control>-U` after the last input character on the previous input line.

109975 Otherwise, it shall be an error.

109976 All of the glyphs on columns between the starting cursor position and (inclusively) the ending

109977 cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on

109978 page 3248).

109979 The current *kill* character (see *stty*) shall cause an equivalent action to the `<control>-U` command,

109980 unless the previously inserted character was a backslash, in which case it shall be as if the literal

109981 current *kill* character had been inserted instead of the backslash.

109982 *Current line*: Unchanged, unless previously input lines are erased, in which case it shall be set to

109983 line -1.

109984 *Current column*: Set to the first column that displays any portion of the last character backed up

109985 over.

109986

**<control>-V**

109987

*Synopsis:* <control>-V

109988

&lt;control&gt;-Q

109989

Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal character, removing any special meaning that it may have to the editor in text input mode. If a <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as described in the <newline> command character during input mode.

109990

109991

109992

109993

109994

For purposes of the display only, the editor shall behave as if a '^' character was entered, and the cursor shall be positioned as if overwriting the '^' character. When a subsequent character is entered, the editor shall behave as if that character was entered instead of the original <control>-V or <control>-Q character.

109995

109996

109997

109998

*Current line:* Unchanged.

109999

*Current column:* Unchanged.

110000

**<control>-W**

110001

*Synopsis:* <control>-W

110002

If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move back over the last word preceding the cursor (including any <blank>s between the end of the last word and the current cursor); the cursor shall not move to before the first character after the end of any **autoindent** characters.

110003

110004

110005

110006

Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-W command is an error or if the cursor moves to the first column position on the line.

110007

110008

110009

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-W command is an error or if it is equivalent to entering <control>-W after the last input character on the previous input line.

110010

110011

110012

Otherwise, it shall be an error.

110013

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in [Input Mode Commands in vi](#) (on page 3248).

110014

110015

110016

*Current line:* Unchanged, unless previously input lines are erased, in which case it shall be set to line -1.

110017

110018

*Current column:* Set to the first column that displays any portion of the last character backed up over.

110019

110020

**<ESC>**

110021

*Synopsis:* <ESC>

110022

If input was part of a line-oriented command:

110023

1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to command mode. The terminal shall be alerted.

110024

110025

2. If <ESC> was entered, text input mode shall be terminated and the command shall continue execution with the input provided.

110026

110027

Otherwise, terminate text input mode and return to command mode.

110028

Any **autoindent** characters entered on newly created lines that have no other non-<newline>s

- 110029 shall be deleted.
- 110030 Any leading **autoindent** and <blank>s on newly created lines shall be rewritten to be the  
110031 minimum number of <blank>s possible.
- 110032 The screen shall be redisplayed as necessary to match the contents of the edit buffer.
- 110033 *Current line:* Unchanged.
- 110034 *Current column:*
- 110035 1. If there are text input characters on the current line, the column shall be set to the last  
110036 column where any portion of the last text input character is displayed.
  - 110037 2. Otherwise, if a character is displayed in the current column, unchanged.
  - 110038 3. Otherwise, set to column position 1.
- 110039 **EXIT STATUS**
- 110040 The following exit values shall be returned:
- 110041 0 Successful completion.
- 110042 >0 An error occurred.
- 110043 **CONSEQUENCES OF ERRORS**
- 110044 When any error is encountered and the standard input is not a terminal device file, *vi* shall not  
110045 write the file or return to command or text input mode, and shall terminate with a non-zero exit  
110046 status.
- 110047 Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP  
110048 asynchronous event.
- 110049 Otherwise, when an error is encountered, the editor shall behave as specified in [Command](#)  
110050 [Descriptions in vi](#) (on page 3216).
- 110051 **APPLICATION USAGE**
- 110052 None.
- 110053 **EXAMPLES**
- 110054 None.
- 110055 **RATIONALE**
- 110056 See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility  
110057 specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been  
110058 implemented as a single utility, this is not required by POSIX.1-200x.
- 110059 It is recognized that portions of *vi* would be difficult, if not impossible, to implement  
110060 satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing,  
110061 thus it is not a mandatory requirement that such features should work on all terminals. It is the  
110062 intention, however, that a *vi* implementation should provide the full set of capabilities on all  
110063 terminals capable of supporting them.
- 110064 Historically, *vi* exited immediately if the standard input was not a terminal. POSIX.1-200x  
110065 permits, but does not require, this behavior. An end-of-file condition is not equivalent to an end-  
110066 of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.
- 110067 The text in the STDOUT section reflects the usage of the verb *display* in this section; some  
110068 implementations of *vi* use standard output to write to the terminal, but POSIX.1-200x does not  
110069 require that to be the case.
- 110070 Historically, implementations reverted to open mode if the terminal was incapable of supporting  
110071 full visual mode. POSIX.1-200x requires this behavior. Historically, the open mode of *vi* behaved  
110072 roughly equivalently to the visual mode, with the exception that only a single line from the edit

110073 buffer (one “buffer line”) was kept current at any time. This line was normally displayed on the  
 110074 next-to-last line of a terminal with cursor addressing (and the last line performed its normal  
 110075 visual functions for line-oriented commands and messages). In addition, some few commands  
 110076 behaved differently in open mode than in visual mode. POSIX.1-200x requires conformance to  
 110077 historical practice.

110078 Historically, *ex* and *vi* implementations have expected text to proceed in the usual  
 110079 European/Latin order of left to right, top to bottom. There is no requirement in POSIX.1-200x  
 110080 that this be the case. The specification was deliberately written using words like “before”,  
 110081 “after”, “first”, and “last” in order to permit implementations to support the natural text order  
 110082 of the language.

110083 Historically, lines past the end of the edit buffer were marked with single tilde (‘~’) characters;  
 110084 that is, if the one-based display was 20 lines in length, and the last line of the file was on line  
 110085 one, then lines 2-20 would contain only a single ‘~’ character.

110086 Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it  
 110087 did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the  
 110088 bottom of the screen, the screen lines where the line would have been displayed were displayed  
 110089 as single ‘@’ characters, instead of displaying part of the line. POSIX.1-200x permits, but does  
 110090 not require, this behavior. Implementations are encouraged to attempt always to display a  
 110091 complete line at the bottom of the screen when doing scrolling or screen positioning by buffer  
 110092 lines.

110093 Historically, lines marked with ‘@’ were also used to minimize output to dumb terminals over  
 110094 slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen  
 110095 that were not close to the cursor were simply marked with an ‘@’ sign instead of being updated  
 110096 to match the current text. POSIX.1-200x permits, but does not require this feature because it is  
 110097 used ever less frequently as terminals become smarter and connections are faster.

#### 110098 Initialization in *ex* and *vi*

110099 Historically, *vi* always had a line in the edit buffer, even if the edit buffer was “empty”. For  
 110100 example:

- 110101 1. The *ex* command = executed from visual mode wrote “1” when the buffer was empty.
- 110102 2. Writes from visual mode of an empty edit buffer wrote files of a single character (a  
 110103 <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
- 110104 3. Put and read commands into an empty edit buffer left an empty line at the top of the edit  
 110105 buffer.

110106 For consistency, POSIX.1-200x does not permit any of these behaviors.

110107 Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was  
 110108 modified if it was not originally set. POSIX.1-200x does not permit this behavior.

#### 110109 Command Descriptions in *vi*

110110 Motion commands are among the most complicated aspects of *vi* to describe. With some  
 110111 exceptions, the text region and buffer type effect of a motion command on a *vi* command are  
 110112 described on a case-by-case basis. The descriptions of text regions in POSIX.1-200x are not  
 110113 intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a  
 110114 region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can  
 110115 be in either direction, and, if the **wrapsan** option is set, so can movements to search points.  
 110116 Historically, lines are always stored into buffers in text order; that is, from the start of the edit  
 110117 buffer to the end. POSIX.1-200x requires conformance to historical practice.

110118 Historically, command counts were applied to any associated motion, and were multiplicative to  
 110119 any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**.

110120 POSIX.1-200x requires this behavior. Historically, *vi* commands that used bigwords, words,  
 110121 paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only  
 110122 <blank>s, inconsistently. Some commands treated them as a single entity, while others treated  
 110123 each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty lines as  
 110124 individual words; that is, the command would move the cursor to each new empty line. The **e**  
 110125 and **E** commands treated groups of empty lines as a single word; that is, the first use would  
 110126 move past the group of lines. The **b** command would just beep at the user, or if done from the  
 110127 start of the line as a motion command, fail in unexpected ways. If the lines contained only (or  
 110128 ended with) <blank>s, the **w** and **W** commands would just beep at the user, the **E** and **e**  
 110129 commands would treat the group as a single word, and the **B** and **b** commands would treat the  
 110130 lines as individual words. For consistency and simplicity of specification, POSIX.1-200x requires  
 110131 that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement  
 110132 through lines ending with <blank>s be consistent with other movements.

110133 Historically, *vi* documentation indicated that any number of double quotes were skipped after  
 110134 punctuation marks at sentence boundaries; however, implementations only skipped single  
 110135 quotes. POSIX.1-200x requires both to be skipped.

110136 Historically, the first and last characters in the edit buffer were word boundaries. This historical  
 110137 practice is required by POSIX.1-200x.

110138 Historically, *vi* attempted to update the minimum number of columns on the screen possible,  
 110139 which could lead to misleading information being displayed. POSIX.1-200x makes no  
 110140 requirements other than that the current character being entered is displayed correctly, leaving  
 110141 all other decisions in this area up to the implementation.

110142 Historically, lines were arbitrarily folded between columns of any characters that required  
 110143 multiple column positions on the screen, with the exception of tabs, which terminated at the  
 110144 right-hand margin. POSIX.1-200x permits the former and requires the latter. Implementations  
 110145 that do not arbitrarily break lines between columns of characters that occupy multiple column  
 110146 positions should not permit the cursor to rest on a column that does not contain any part of a  
 110147 character.

110148 The historical *vi* had a problem in that all movements were by buffer lines, not by display or  
 110149 screen lines. This is often the right thing to do; for example, single line movements, such as **j** or  
 110150 **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only  
 110151 make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling  
 110152 commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines  
 110153 in these cases can result in completely random motion; for example, **1<control>-D** can result in a  
 110154 completely changed screen, without any overlap. This is clearly not what the user wanted. The  
 110155 problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at  
 110156 the first non-<blank> of the line, they may all refer to the same location in large lines, and will  
 110157 result in no movement at all.

110158 In addition, if the line is larger than the screen, using buffer lines can make it impossible to  
 110159 display parts of the line—there are not any commands that do not display the beginning of the  
 110160 line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the  
 110161 same time, the user suffers. Finally, the page and half-page scrolling commands historically  
 110162 moved to the first non-<blank> in the new line. If the line is approximately the same size as the  
 110163 screen, this is inadequate because the cursor before and after a <control>-D command will refer  
 110164 to the same location on the screen.

110165 Implementations of *ex* and *vi* exist that do not have these problems because the relevant  
 110166 commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**,  
 110167 and **M**) operate on display (screen) lines, not (edit) buffer lines.

110168 POSIX.1-200x does not permit this behavior by default because the standard developers believed  
 110169 that users would find it too confusing. However, historical practice has been relaxed. For



110170 example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a  
 110171 line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the  
 110172 line was displayed, and the screen lines corresponding to the line contained single '@'  
 110173 characters. This behavior is permitted, but not required by POSIX.1-200x, so that it is possible for  
 110174 implementations to support long lines in small screens more reasonably without changing the  
 110175 commands to be oriented to the display (instead of oriented to the buffer). POSIX.1-200x also  
 110176 permits implementations to refuse to edit any edit buffer containing a line that will not fit on the  
 110177 screen in its entirety.

110178 The display area (for example, the value of the **window** edit option) has historically been  
 110179 "grown", or expanded, to display new text when local movements are done in displays where  
 110180 the number of lines displayed is less than the maximum possible. Expansion has historically  
 110181 been the first choice, when the target line is less than the maximum possible expansion value  
 110182 away. Scrolling has historically been the next choice, done when the target line is less than half a  
 110183 display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex*  
 110184 commands generally always caused the screen to be redrawn. POSIX.1-200x does not specify a  
 110185 standard behavior because there may be external issues, such as connection speed, the number  
 110186 of characters necessary to redraw as opposed to scroll, or terminal capabilities that  
 110187 implementations will have to accommodate.

110188 The current line in POSIX.1-200x maps one-to-one to a buffer line in the file. The current column  
 110189 does not. There are two different column values that are described by POSIX.1-200x. The first is  
 110190 the current column value as set by many of the *vi* commands. This value is remembered for the  
 110191 lifetime of the editor. The second column value is the actual position on the screen where the  
 110192 cursor rests. The two are not always the same. For example, when the cursor is backed by a  
 110193 multi-column character, the actual cursor position on the screen has historically been the last  
 110194 column of the character in command mode, and the first column of the character in input mode.

110195 Commands that set the current line, but that do not set the current cursor value (for example, **j**  
 110196 and **k**) attempt to get as close as possible to the remembered column position, so that the cursor  
 110197 tends to restrict itself to a vertical column as the user moves around in the edit buffer.  
 110198 POSIX.1-200x requires conformance to historical practice, requiring that the display location of  
 110199 the cursor on the display line be adjusted from the current column value as necessary to support  
 110200 this historical behavior.

110201 Historically, only a single line (and for some terminals, a single line minus 1 column) of  
 110202 characters could be entered by the user for the line-oriented commands; that is, **;**, **!**, **/**, or **?**.  
 110203 POSIX.1-200x permits, but does not require, this limitation.

110204 Historically, "soft" errors in *vi* caused the terminal to be alerted, but no error message was  
 110205 displayed. As a general rule, no error message was displayed for errors in command execution  
 110206 in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when  
 110207 a searched-for object was not found. Examples of soft errors included **h** at the left margin,  
 110208 <control>-**B** or **[** at the beginning of the file, **2G** at the end of the file, and so on. In addition,  
 110209 errors such as **%**, **]**, **]**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well.  
 110210 Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n**  
 110211 displayed an error message if no previous regular expression had been specified, and **;** did not  
 110212 display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in  
 110213 this area might reasonably be based on a runtime evaluation of the speed of a network  
 110214 connection. Finally, some implementations have provided error messages for soft errors in order  
 110215 to assist naive users, based on the value of a verbose edit option. POSIX.1-200x does not list  
 110216 specific errors for which an error message shall be displayed. Implementations should conform  
 110217 to historical practice in the absence of any strong reason to diverge.

### Page Backwards

The <control>-B and <control>-F commands historically considered it an error to attempt to page past the beginning or end of the file, whereas the <control>-D and <control>-U commands simply moved to the beginning or end of the file. For consistency, POSIX.1-200x requires the latter behavior for all four commands. All four commands still consider it an error if the current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of the file. Historically, the <control>-B and <control>-F commands skip two lines in order to include overlapping lines when a single command is entered. This makes less sense in the presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation used by historical implementations of the *vi* editor for <control>-B was:

```
((current first line) - count x (window edit option)) +2
```

and for <control>-F was:

```
((current first line) + count x (window edit option)) -2
```

This calculation does not work well when intermixing commands with and without counts; for example, **3<control>-F is not equivalent to entering the <control>-F command three times, and is not reversible by entering the <control>-B command three times. For consistency with other *vi* commands that take counts, POSIX.1-200x requires a different calculation.**

### Scroll Forward

The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll** command. 4BSD used:

```
((window edit option) +1) /2
```

while System V used the value of the **scroll** edit option. The System V version is specified by POSIX.1-200x because the standard developers believed that it was more intuitive and permitted the user a method of setting the scroll value initially without also setting the number of lines that are displayed.

### Scroll Forward by Line

Historically, the <control>-E and <control>-Y commands considered it an error if the last and first lines, respectively, were already on the screen. POSIX.1-200x requires conformance to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in open mode. For simplicity and consistency of specification, POSIX.1-200x requires that they behave as usual, albeit with a single line screen.

### Clear and Redisplay

The historical <control>-L command refreshed the screen exactly as it was supposed to be currently displayed, replacing any '@' characters for lines that had been deleted but not updated on the screen with refreshed '@' characters. The intent of the <control>-L command is to refresh when the screen has been accidentally overwritten; for example, by a **write** command from another user, or modem noise.

110255

**Redraw Screen**

110256

110257

110258

110259

110260

110261

The historical <control>-R command redisplayed only when necessary to update lines that had been deleted but not updated on the screen and that were flagged with '@' characters. There is no requirement that the screen be in any way refreshed if no lines of this form are currently displayed. POSIX.1-200x permits implementations to extend this command to refresh lines on the screen flagged with '@' characters because they are too long to be displayed in the current framework; however, the current line and column need not be modified.

110262

**Search for tagstring**

110263

110264

110265

110266

110267

Historically, the first non-<blank> at or after the cursor was the first character, and all subsequent characters that were word characters, up to the end of the line, were included. For example, with the cursor on the leading space or on the '#' character in the text "#bar@", the tag was "#bar". On the character 'b' it was "bar", and on the 'a' it was "ar". POSIX.1-200x requires this behavior.

110268

**Replace Text with Results from Shell Command**

110269

110270

110271

110272

110273

Historically, the <, >, and ! commands considered most cursor motions other than line-oriented motions an error; for example, the command >/foo<CR> succeeded, while the command >I failed, even though the text region described by the two commands might be identical. For consistency, all three commands only consider entire lines and not partial lines, and the region is defined as any line that contains a character that was specified by the motion.

110274

**Move to Matching Character**

110275

110276

110277

Other matching characters have been left implementation-defined in order to allow extensions such as matching '<' and '>' for searching HTML, or #ifdef, #else, and #endif for searching C source.

110278

**Repeat Substitution**

110279

110280

POSIX.1-200x requires that any c and g flags specified to the previous substitute command be ignored; however, the r flag may still apply, if supported by the implementation.

110281

**Return to Previous (Context or Section)**

110282

110283

110284

110285

110286

110287

110288

110289

110290

110291

110292

The [, ], (, ), {, and } commands are all affected by "section boundaries", but in some historical implementations not all of the commands recognize the same section boundaries. This is a bug, not a feature, and a unique section-boundary algorithm was not described for each command. One special case that is preserved is that the sentence command moves to the end of the last line of the edit buffer while the other commands go to the beginning, in order to preserve the traditional character cut semantics of the sentence command. Historically, vi section boundaries at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit buffer if one exists. To increase consistency with other section locations, this has been simplified by POSIX.1-200x to the first character of the first and last lines of the edit buffer, or the first and the last lines of the edit buffer if they are empty.

110293

110294

110295

110296

110297

110298

Sentence boundaries were problematic in the historical vi. They were not only the boundaries as defined for the section and paragraph commands, but they were the first non-<blank> that occurred after those boundaries, as well. Historically, the vi section commands were documented as taking an optional window size as a count preceding the command. This was not implemented in historical versions, so POSIX.1-200x requires that the count repeat the command, for consistency with other vi commands.

**Repeat**

Historically, mapped commands other than text input commands could not be repeated using the **period** command. POSIX.1-200x requires conformance to historical practice.

The restrictions on the interpretation of special characters (for example, <control>-H) in the repetition of text input mode commands is intended to match historical practice. For example, given the input sequence:

```
iab<control>-H<control>-H<control>-Hdef<escape>
```

the user should be informed of an error when the sequence is first entered, but not during a command repetition. The character <control>-T is specifically exempted from this restriction. Historical implementations of *vi* ignored <control>-T characters that were input in the original command during command repetition. POSIX.1-200x prohibits this behavior.

**Find Regular Expression**

Historically, commands did not affect the line searched to or from if the motion command was a search (*l*, *?*, *N*, *n*) and the final position was the start/end of the line. There were some special cases and *vi* was not consistent. POSIX.1-200x does not permit this behavior, for consistency. Historical implementations permitted but were unable to handle searches as motion commands that wrapped (that is, due to the edit option **wraps**) to the original location. POSIX.1-200x requires that this behavior be treated as an error.

Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode. POSIX.1-200x requires conformance to historical practice.

Historically, in open mode, a **z** specified to a search command redisplayed the current line instead of displaying the current screen with the current line highlighted. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

Historically, trailing **z** commands were permitted and ignored if entered as part of a search used as a motion command. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

**Execute an ex Command**

Historically, *vi* implementations restricted the commands that could be entered on the colon command line (for example, **append** and **change**), and some other commands were known to cause them to fail catastrophically. For consistency, POSIX.1-200x does not permit these restrictions. When executing an *ex* command by entering `:`, it is not possible to enter a <newline> as part of the command because it is considered the end of the command. A different approach is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for example, the following is valid:

```
Q
s/break here/break\
here/
vi
```

POSIX.1-200x requires that, if the *ex* command overwrites any part of the screen that would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character could be any character; for example, a character input by the user before the message appeared, or even a mapped character. This is probably a bug, but implementations that have tried to be more rigorous by requiring that the user enter a specific character, or that the user enter a character after the message was displayed, have been forced by user indignation back into historical behavior. POSIX.1-200x requires conformance to historical practice.

110345

**Shift Left (Right)**110346  
110347  
110348  
110349  
110350  
110351  
110352

Refer to the Rationale for the ! and / commands. Historically, the < and > commands sometimes moved the cursor to the first non-<blank> (for example if the command was repeated or with \_ as the motion command), and sometimes left it unchanged. POSIX.1-200x does not permit this inconsistency, requiring instead that the cursor always move to the first non-<blank>. Historically, the < and > commands did not support buffer arguments, although some implementations allow the specification of an optional buffer. This behavior is neither required nor disallowed by POSIX.1-200x.

110353

**Execute**110354  
110355  
110356  
110357

Historically, buffers could execute other buffers, and loops, infinite and otherwise, were possible. POSIX.1-200x requires conformance to historical practice. The *\*buffer* syntax of *ex* is not required in *vi*, because it is not historical practice and has been used in some *vi* implementations to support additional scripting languages.

110358

**Reverse Case**110359  
110360  
110361  
110362  
110363  
110364  
110365  
110366

Historically, the ~ command ignored any associated *count*, and acted only on the characters in the current line. For consistency with other *vi* commands, POSIX.1-200x requires that an associated *count* act on the next *count* characters, and that the command move to subsequent lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient manner. There exist *vi* implementations that optionally require an associated motion command for the ~ command. Implementations supporting this functionality are encouraged to base it on the **tildedop** edit option and handle the text regions and cursor positioning identically to the **yank** command.

110367

**Append**110368  
110369  
110370

Historically, *counts* specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line *count* times, and did not repeat the subsequent lines of the input text. POSIX.1-200x requires that the entire text input be repeated *count* times.

110371

**Move Backward to Preceding Word**110372  
110373  
110374  
110375  
110376  
110377

Historically, *vi* became confused if word commands were used as motion commands in empty files. POSIX.1-200x requires that this be an error. Historical implementations of *vi* had a large number of bugs in the word movement commands, and they varied greatly in behavior in the presence of empty lines, "words" made up of a single character, and lines containing only <blank>s. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

110378

**Change to End-of-Line**110379  
110380  
110381  
110382  
110383  
110384

Some historical implementations of the **C** command did not behave as described by POSIX.1-200x when the \$ key was remapped because they were implemented by pushing the \$ key onto the input queue and reprocessing it. POSIX.1-200x does not permit this behavior. Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric buffers. For consistency and simplicity of specification, POSIX.1-200x requires that they behave like their respective **c** commands in all respects.

## Delete

Historically, lines in open mode that were deleted were scrolled up, and an @ glyph written over the beginning of the line. In the case of terminals that are incapable of the necessary cursor motions, the editor erased the deleted line from the screen. POSIX.1-200x requires conformance to historical practice; that is, if the terminal cannot display the '@' character, the line cannot remain on the screen.

## Delete to End-of-Line

Some historical implementations of the **D** command did not behave as described by POSIX.1-200x when the \$ key was remapped because they were implemented by pushing the \$ key onto the input queue and reprocessing it. POSIX.1-200x does not permit this behavior.

## Join

An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent. POSIX.1-200x requires conformance to historical practice. The *vi* **J** command is specified in terms of the *ex* **join** command with an *ex* command *count* value. The address correction for a *count* that is past the end of the edit buffer is necessary for historical compatibility for both *ex* and *vi*.

## Mark Position

Historical practice is that only lowercase letters, plus '' and ''', could be used to mark a cursor position. POSIX.1-200x requires conformance to historical practice, but encourages implementations to support other characters as marks as well.

## Repeat Regular Expression Find (Forward and Reverse)

Historically, the **N** and **n** commands could not be used as motion components for the **c** command. With the exception of the **cN** command, which worked if the search crossed a line boundary, the text region would be discarded, and the user would not be in text input mode. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

## Insert Empty Line (Below and Above)

Historically, counts to the **O** and **o** commands were used as the number of physical lines to open, if the terminal was dumb and the **slowopen** option was not set. This was intended to minimize traffic over slow connections and repainting for dumb terminals. POSIX.1-200x does not permit this behavior, requiring that a *count* to the open command behave as for other text input commands. This change to historical practice was made for consistency, and because a superset of the functionality is provided by the **slowopen** edit option.

## Put from Buffer (Following and Before)

Historically, counts to the **p** and **P** commands were ignored if the buffer was a line mode buffer, but were (mostly) implemented as described in POSIX.1-200x if the buffer was a character mode buffer. Because implementations exist that do not have this limitation, and because pasting lines multiple times is generally useful, POSIX.1-200x requires that *count* be supported for all **p** and **P** commands.

Historical implementations of *vi* were widely known to have major problems in the **p** and **P** commands, particularly when unusual regions of text were copied into the edit buffer. The standard developers viewed these as bugs, and they are not permitted for consistency and simplicity of specification.

Historically, a **P** or **p** command (or an *ex* **put** command executed from open or visual mode) executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

110429

**Replace Character**

110430

110431

110432

110433

Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return> argument, for which it replaced *count* characters with a single <newline>. POSIX.1-200x does not permit these inconsistencies.

110434

110435

110436

110437

Historically, the **r** command permitted the <control>-V escaping of entered characters, such as <ESC> and the <carriage-return>; however, it required two leading <control>-V characters instead of one. POSIX.1-200x requires that this be changed for consistency with the other text input commands of *vi*.

110438

110439

110440

110441

110442

Historically, it is an error to enter the **r** command if there are less than *count* characters at or after the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r** command on empty lines, it would require that too large a *count* be adjusted to match the number of characters at or after the cursor for consistency, which is sufficiently different from historical practice to be avoided. POSIX.1-200x requires conformance to historical practice.

110443

**Replace Characters**

110444

110445

110446

110447

110448

Historically, if there were **autoindent** characters in the line on which the **R** command was run, and **autoindent** was set, the first <newline> would be properly indented and no characters would be replaced by the <newline>. Each additional <newline> would replace *n* characters, where *n* was the number of characters that were needed to indent the rest of the line to the proper indentation level. This behavior is a bug and is not permitted by POSIX.1-200x.

110449

**Undo**

110450

110451

110452

110453

110454

110455

110456

110457

110458

110459

110460

Historical practice for cursor positioning after undoing commands was mixed. In most cases, when undoing commands that affected a single line, the cursor was moved to the start of added or changed text, or immediately after deleted text. However, if the user had moved from the line being changed, the column was either set to the first non-<blank>, returned to the origin of the command, or remained unchanged. When undoing commands that affected multiple lines or entire lines, the cursor was moved to the first character in the first line restored. As an example of how inconsistent this was, a search, followed by an **o** text input command, followed by an **undo** would return the cursor to the location where the **o** command was entered, but a **cw** command followed by an **o** command followed by an **undo** would return the cursor to the first non-<blank> of the line. POSIX.1-200x requires the most useful of these behaviors, and discards the least useful, in the interest of consistency and simplicity of specification.

110461

**Yank**

110462

110463

110464

110465

110466

110467

110468

110469

110470

110471

Historically, the **yank** command did not move to the end of the motion if the motion was in the forward direction. It moved to the end of the motion if the motion was in the backward direction, except for the **\_** command, or for the **G** and **'** commands when the end of the motion was on the current line. This was further complicated by the fact that for a number of motion commands, the **yank** command moved the cursor but did not update the screen; for example, a subsequent command would move the cursor from the end of the motion, even though the cursor on the screen had not reflected the cursor movement for the **yank** command. POSIX.1-200x requires that all **yank** commands associated with backward motions move the cursor to the end of the motion for consistency, and specifically, to make **'** commands as motions consistent with search patterns as motions.

### Yank Current Line

Some historical implementations of the **Y** command did not behave as described by POSIX.1-200x when the `'_'` key was remapped because they were implemented by pushing the `'_'` key onto the input queue and reprocessing it. POSIX.1-200x does not permit this behavior.

### Redraw Window

Historically, the **z** command always redrew the screen. This is permitted but not required by POSIX.1-200x, because of the frequent use of the **z** command in macros such as `map n nz.` for screen positioning, instead of its use to change the screen size. The standard developers believed that expanding or scrolling the screen offered a better interface for users. The ability to redraw the screen is preserved if the optional new window size is specified, and in the `<control>-L` and `<control>-R` commands.

The semantics of `z^` are confusing at best. Historical practice is that the screen before the screen that ended with the specified line is displayed. POSIX.1-200x requires conformance to historical practice.

Historically, the **z** command would not display a partial line at the top or bottom of the screen. If the partial line would normally have been displayed at the bottom of the screen, the command worked, but the partial line was replaced with `'@'` characters. If the partial line would normally have been displayed at the top of the screen, the command would fail. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

Historically, the **z** command with a line specification of 1 ignored the command. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

Historically, the **z** command did not set the cursor column to the first non-`<blank>` for the character if the first screen was to be displayed, and was already displayed. For consistency and simplicity of specification, POSIX.1-200x does not permit this behavior.

### Input Mode Commands in vi

Historical implementations of *vi* did not permit the user to erase more than a single line of input, or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent** characters. As there exist implementations of *vi* that do not have these limitations, both behaviors are permitted, but only historical practice is required. In the case of these extensions, *vi* is required to pause at the **autoindent** and previous line boundaries.

Historical implementations of *vi* updated only the portion of the screen where the current cursor character was displayed. For example, consider the *vi* input keystrokes:

```
iabcd<escape>0C<tab>
```

Historically, the `<tab>` would overwrite the characters "abcd" when it was displayed. Other implementations replace only the `'a'` character with the `<tab>`, and then push the rest of the characters ahead of the cursor. Both implementations have problems. The historical implementation is probably visually nicer for the above example; however, for the keystrokes:

```
iabcd<ESC>0R<tab><ESC>
```

the historical implementation results in the string "bcd" disappearing and then magically reappearing when the `<ESC>` character is entered. POSIX.1-200x requires the former behavior when overwriting erase-columns—that is, overwriting characters that are no longer logically part of the edit buffer—and the latter behavior otherwise.

Historical implementations of *vi* discarded the `<control>-D` and `<control>-T` characters when they were entered at places where their command functionality was not appropriate. POSIX.1-200x requires that the `<control>-T` functionality always be available, and that `<control>-D` be treated as any other key when not operating on **autoindent** characters.



110518  
110519  
110520  
110521  
  
110522  
  
110523  
110524  
110525  
110526  
  
110527  
  
110528  
110529  
110530  
110531  
110532  
  
110533  
  
110534  
110535  
110536  
110537  
110538  
110539  
  
110540  
  
110541  
110542  
  
110543  
110544  
110545  
110546  
110547  
110548  
110549  
110550  
  
110551  
  
110552  
110553  
  
110554  
110555  
  
110556  
110557  
  
110558

**NUL**

Some historical implementations of *vi* limited the number of characters entered using the NUL input character to 256 bytes. POSIX.1-200x permits this limitation; however, implementations are encouraged to remove this limit.

**<control>-D**

See also Rationale for the input mode command <newline>. The hidden assumptions in the <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>s take up a single column on the screen and that <tab>s are comprised of an integral number of <space>s.

**<newline>**

Implementations are permitted to rewrite **autoindent** characters in the line when <newline>, <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are used, because historical implementations have both done so and found it necessary to do so. For example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and **shiftwidth** set to 3, will result in the <tab> being replaced by several <space>s.

**<control>-T**

See also the Rationale for the input mode command <newline>. Historically, <control>-T only worked if no non-<blank>s had yet been input in the current input line. In addition, the characters inserted by <control>-T were treated as **autoindent** characters, and could not be erased using normal user erase characters. Because implementations exist that do not have these limitations, and as moving to a column boundary is generally useful, POSIX.1-200x requires that both limitations be removed.

**<control>-V**

Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal. POSIX.1-200x requires conformance to historical practice.

The uses described for <control>-V can also be accomplished with <control>-Q, which is useful on terminals that use <control>-V for the down-arrow function. However, most historical implementations use <control>-Q for the *termios* START character, so the editor will generally not receive the <control>-Q unless **stty ixon** mode is set to off. (In addition, some historical implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to off.) Any of the command characters described in POSIX.1-200x can be made ineffective by their selection as *termios* control characters, using the *stty* utility or other methods described in the System Interfaces volume of POSIX.1-200x.

**<ESC>**

Historically, SIGINT alerted the terminal when used to end input mode. This behavior is permitted, but not required, by POSIX.1-200x.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*ed*, *ex*, *stty*

XBD Section 12.2 (on page 201)

+

**CHANGE HISTORY**

110559 First released in Issue 2.  
110560

**Issue 5**

110561 The FUTURE DIRECTIONS section is added.  
110562

**Issue 6**

110563 This utility is marked as part of the User Portability Utilities option.  
110564

110565 The APPLICATION USAGE section is added.  
110566

110566 The obsolescent SYNOPSIS is removed.  
110567

110567 The following new requirements on POSIX implementations derive from alignment with the  
110568 Single UNIX Specification:

- The **reindent** command description is added.  
110569

110570 The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft  
110571 standard.

110572 IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.

110573 IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in  
110574 a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.

110575 The **-l** option is removed.

110576 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/41 is applied, adding [*count*] to the  
110577 Synopsis for **[[**.

110578 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/42 is applied, adding [*count*] to the  
110579 Synopsis for **]]**.

**Issue 7**

110580 Austin Group Interpretation 1003.1-2001 #027 is applied, clarifying that '+' may be recognized  
110581 as an option delimiter in the OPTIONS section.  
110582

110583 Austin Group Interpretation 1003.1-2001 #087 is applied, updating the Put from Buffer Before (**P**)  
110584 command description to address multi-line requirements.

110585 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

110586 **NAME**  
 110587 wait — await process completion

110588 **SYNOPSIS**  
 110589 wait [*pid*...]

110590 **DESCRIPTION**  
 110591 When an asynchronous list (see [Section 2.9.3.1](#), on page 2266) is started by the shell, the process  
 110592 ID of the last command in each element of the asynchronous list shall become known in the  
 110593 current shell execution environment; see [Section 2.12](#) (on page 2277).

110594 If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the  
 110595 invoking shell have terminated and exit with a zero exit status.

110596 If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall  
 110597 wait until all of them have terminated. If one or more *pid* operands are specified that represent  
 110598 unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with  
 110599 exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process  
 110600 requested by the last *pid* operand.

110601 The known process IDs are applicable only for invocations of *wait* in the current shell execution  
 110602 environment.

110603 **OPTIONS**  
 110604 None.

110605 **OPERANDS**  
 110606 The following operand shall be supported:

110607 *pid* One of the following:

- 110608 1. The unsigned decimal integer process ID of a command, for which the  
 110609 utility is to wait for the termination.
- 110610 2. A job control job ID (see XBD [Section 3.202](#), on page 61) that identifies a  
 110611 background process group to be waited for. The job control job ID notation  
 110612 is applicable only for invocations of *wait* in the current shell execution  
 110613 environment; see [Section 2.12](#) (on page 2277). The exit status of *wait* shall be  
 110614 determined by the last command in the pipeline.

110615 **Note:** The job control job ID type of *pid* is only available on systems supporting  
 110616 the User Portability Utilities option.

110617 **STDIN**  
 110618 Not used.

110619 **INPUT FILES**  
 110620 None.

110621 **ENVIRONMENT VARIABLES**  
 110622 The following environment variables shall affect the execution of *wait*:

110623 *LANG* Provide a default value for the internationalization variables that are unset or null.  
 110624 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization  
 110625 variables used to determine the values of locale categories.)

110626 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 110627 internationalization variables.

110628 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 110629 characters (for example, single-byte as opposed to multi-byte characters in  
 110630 arguments).

110631 *LC\_MESSAGES*  
 110632 Determine the locale that should be used to affect the format and contents of  
 110633 diagnostic messages written to standard error.

110634 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

#### 110635 **ASYNCHRONOUS EVENTS**

110636 Default.

#### 110637 **STDOUT**

110638 Not used.

#### 110639 **STDERR**

110640 The standard error shall be used only for diagnostic messages.

#### 110641 **OUTPUT FILES**

110642 None.

#### 110643 **EXTENDED DESCRIPTION**

110644 None.

#### 110645 **EXIT STATUS**

110646 If one or more operands were specified, all of them have terminated or were not known by the  
 110647 invoking shell, and the status of the last operand specified is known, then the exit status of *wait*  
 110648 shall be the exit status information of the command indicated by the last operand specified. If  
 110649 the process terminated abnormally due to the receipt of a signal, the exit status shall be greater  
 110650 than 128 and shall be distinct from the exit status generated by other signals, but the exact value  
 110651 is unspecified. (See the *kill -l* option.) Otherwise, the *wait* utility shall exit with one of the  
 110652 following values:

110653 0 The *wait* utility was invoked with no operands and all process IDs known by the  
 110654 invoking shell have terminated.

110655 1-126 The *wait* utility detected an error.

110656 127 The command identified by the last *pid* operand specified is unknown.

#### 110657 **CONSEQUENCES OF ERRORS**

110658 Default.

#### 110659 **APPLICATION USAGE**

110660 On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility  
 110661 execution environment, such as one of the following:

```
110662 (wait)
110663 nohup wait ...
110664 find . -exec wait ... \;
```

110665 it returns immediately because there are no known process IDs to wait for in those  
 110666 environments.

110667 Historical implementations of interactive shells have discarded the exit status of terminated  
 110668 background processes before each shell prompt. Therefore, the status of background processes  
 110669 was usually lost unless it terminated while *wait* was waiting for it. This could be a serious  
 110670 problem when a job that was expected to run for a long time actually terminated quickly with a  
 110671 syntax or initialization error because the exit status returned was usually zero if the requested  
 110672 process ID was not found. This volume of POSIX.1-200x requires the implementation to keep the  
 110673 status of terminated jobs available until the status is requested, so that scripts like:

```

110674 j1&
110675 p1=$!
110676 j2&
110677 wait $p1
110678 echo Job 1 exited with status $?
110679 wait $!
110680 echo Job 2 exited with status $?

```

work without losing status on any of the jobs. The shell is allowed to discard the status of any process if it determines that the application cannot get the process ID for that process from the shell. It is also required to remember only {CHILD\_MAX} number of processes in this way. Since the only way to get the process ID from the shell is by using the '!' shell parameter, the shell is allowed to discard the status of an asynchronous list if "\$!" was not referenced before another asynchronous list was started. (This means that the shell only has to keep the status of the last asynchronous list started if the application did not reference "\$!". If the implementation of the shell is smart enough to determine that a reference to "\$!" was not saved anywhere that the application can retrieve it later, it can use this information to trim the list of saved information. Note also that a successful call to *wait* with no operands discards the exit status of all asynchronous lists.)

If the exit status of *wait* is greater than 128, there is no way for the application to know if the waited-for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

#### EXAMPLES

Although the exact value used when a process is terminated by a signal is unspecified, if it is known that a signal terminated a process, a script can still reliably determine which signal by using *kill* as shown by the following script:

```

110701 sleep 1000&
110702 pid=$!
110703 kill -kill $pid
110704 wait $pid
110705 echo $pid was terminated by a SIG$(kill -l $?) signal.

```

If the following sequence of commands is run in less than 31 seconds:

```

110707 sleep 257 | sleep 31 &
110708 jobs -l %%
110709 either of the following commands returns the exit status of the second sleep in the pipeline:

```

```

110710 wait <pid of sleep 31>
110711 wait %%

```

#### RATIONALE

The description of *wait* does not refer to the *waitpid()* function from the System Interfaces volume of POSIX.1-200x because that would needlessly overspecify this interface. However, the wording means that *wait* is required to wait for an explicit process when it is given an argument so that the status information of other processes is not consumed. Historical implementations use the *wait()* function defined in the System Interfaces volume of POSIX.1-200x until *wait()* returns the requested process ID or finds that the requested process does not exist. Because this means that a shell script could not reliably get the status of all background children if a second background job was ever started before the first job finished, it is recommended that the *wait* utility use a method such as the functionality provided by the *waitpid()* function.

The ability to wait for multiple *pid* operands was adopted from the KornShell.

110723 This new functionality was added because it is needed to determine the exit status of any  
110724 asynchronous list accurately. The only compatibility problem that this change creates is for a  
110725 script like

```
110726 while sleep 60 do
110727 job& echo Job started $(date) as $! done
```

110728 which causes the shell to monitor all of the jobs started until the script terminates or runs out of  
110729 memory. This would not be a problem if the loop did not reference "\$!" or if the script would  
110730 occasionally *wait* for jobs it started.

#### 110731 **FUTURE DIRECTIONS**

110732 None.

#### 110733 **SEE ALSO**

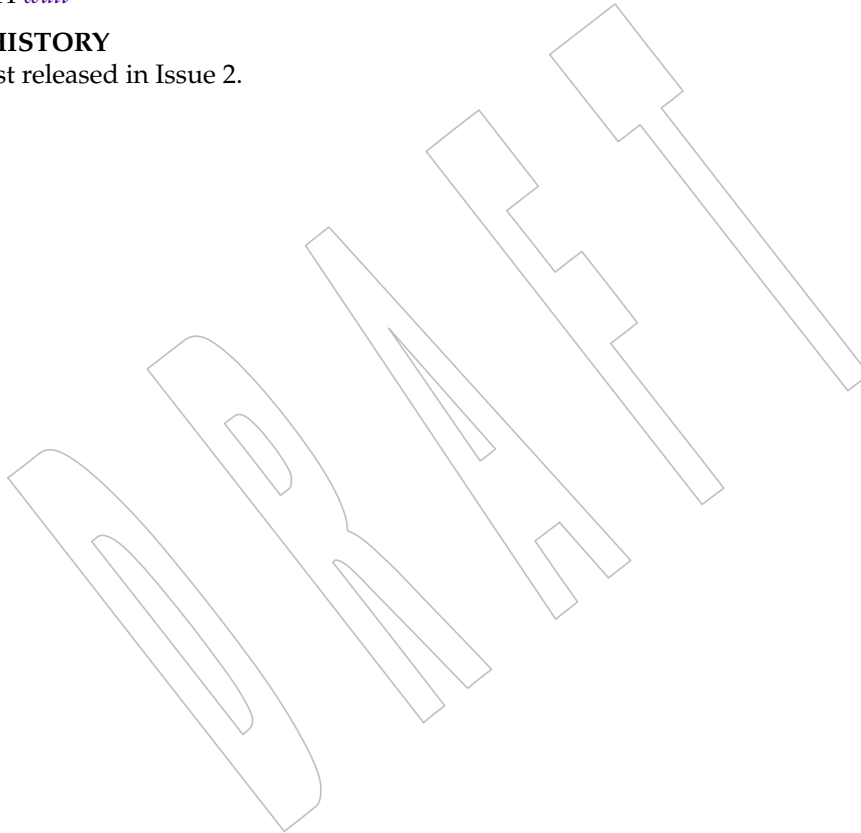
110734 [Chapter 2](#) (on page 2245), *kill*, *sh*

110735 XBD [Section 3.202](#) (on page 61), [Chapter 8](#) (on page 159)

110736 XSH *wait*

#### 110737 **CHANGE HISTORY**

110738 First released in Issue 2.



110739 **NAME**110740 `wc` — word, line, and byte or character count110741 **SYNOPSIS**110742 `wc [-c|-m] [-lw] [file...]`110743 **DESCRIPTION**110744 The `wc` utility shall read one or more input files and, by default, write the number of  
110745 <newline>s, words, and bytes contained in each input file to the standard output.110746 The utility also shall write a total count for all named files, if more than one input file is  
110747 specified.110748 The `wc` utility shall consider a *word* to be a non-zero-length string of characters delimited by  
110749 white space.110750 **OPTIONS**110751 The `wc` utility shall conform to XBD [Section 12.2](#) (on page 201).

110752 The following options shall be supported:

110753 `-c` Write to the standard output the number of bytes in each input file.110754 `-l` Write to the standard output the number of <newline>s in each input file.110755 `-m` Write to the standard output the number of characters in each input file.110756 `-w` Write to the standard output the number of words in each input file.110757 When any option is specified, `wc` shall report only the information requested by the specified  
110758 options.110759 **OPERANDS**

110760 The following operand shall be supported:

110761 *file* A pathname of an input file. If no *file* operands are specified, the standard input  
110762 shall be used.110763 **STDIN**110764 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES  
110765 section.110766 **INPUT FILES**

110767 The input files may be of any type.

110768 **ENVIRONMENT VARIABLES**110769 The following environment variables shall affect the execution of `wc`:110770 `LANG` Provide a default value for the internationalization variables that are unset or null. |  
110771 (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
110772 variables used to determine the values of locale categories.)110773 `LC_ALL` If set to a non-empty string value, override the values of all the other  
110774 internationalization variables.110775 `LC_CTYPE` Determine the locale for the interpretation of sequences of bytes of text data as  
110776 characters (for example, single-byte as opposed to multi-byte characters in  
110777 arguments and input files) and which characters are defined as white space  
110778 characters.

110779 *LC\_MESSAGES*  
 110780 Determine the locale that should be used to affect the format and contents of  
 110781 diagnostic messages written to standard error and informative messages written to  
 110782 standard output.

110783 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

### 110784 ASYNCHRONOUS EVENTS

110785 Default.

### 110786 STDOUT

110787 By default, the standard output shall contain an entry for each input file of the form:

110788 "%d %d %d %s\n", *<newlines>*, *<words>*, *<bytes>*, *<file>*

110789 If the *-m* option is specified, the number of characters shall replace the *<bytes>* field in this  
 110790 format.

110791 If any options are specified and the *-l* option is not specified, the number of *<newline>*s shall  
 110792 not be written.

110793 If any options are specified and the *-w* option is not specified, the number of words shall not be  
 110794 written.

110795 If any options are specified and neither *-c* nor *-m* is specified, the number of bytes or characters  
 110796 shall not be written.

110797 If no input *file* operands are specified, no name shall be written and no *<blank>*s preceding the  
 110798 pathname shall be written.

110799 If more than one input *file* operand is specified, an additional line shall be written, of the same  
 110800 format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead  
 110801 of a pathname and the total of each column shall be written as appropriate. Such an additional  
 110802 line, if any, is written at the end of the output.

### 110803 STDERR

110804 The standard error shall be used only for diagnostic messages.

### 110805 OUTPUT FILES

110806 None.

### 110807 EXTENDED DESCRIPTION

110808 None.

### 110809 EXIT STATUS

110810 The following exit values shall be returned:

110811 0 Successful completion.

110812 >0 An error occurred.

### 110813 CONSEQUENCES OF ERRORS

110814 Default.



**APPLICATION USAGE**

The `-m` option is not a switch, but an option at the same level as `-c`. Thus, to produce the full default output with character counts instead of bytes, the command required is:

```
wc -mlw
```

**EXAMPLES**

None.

**RATIONALE**

The output file format pseudo-*printf*( ) string differs from the System V version of *wc*:

```
"%7d%7d%7d %s\n"
```

which produces possibly ambiguous and unparseable results for very large files, as it assumes no number shall exceed six digits.

Some historical implementations use only `<space>`, `<tab>`, and `<newline>` as word separators. The equivalent of the ISO C standard *isspace*( ) function is more appropriate.

The `-c` option stands for “character” count, even though it counts bytes. This stems from the sometimes erroneous historical view that bytes and characters are the same size. Due to international requirements, the `-m` option (reminiscent of “multi-byte”) was added to obtain actual character counts.

Early proposals only specified the results when input files were text files. The current specification more closely matches historical practice. (Bytes, words, and `<newline>`s are counted separately and the results are written when an end-of-file is detected.)

Historical implementations of the *wc* utility only accepted one argument to specify the options `-c`, `-l`, and `-w`. Some of them also had multiple occurrences of an option cause the corresponding count to be written multiple times and had the order of specification of the options affect the order of the fields on output, but did not document either of these. Because common usage either specifies no options or only one option, and because none of this was documented, the changes required by this volume of POSIX.1-200x should not break many historical applications (and do not break any historical conforming applications).

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*cksum*

XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

**CHANGE HISTORY**

First released in Issue 2.

**Issue 7**

SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

110851 **NAME**110852           what — identify SCCS files (**DEVELOPMENT**)110853 **SYNOPSIS**110854 XSI        **what** [-s] *file...*110855 **DESCRIPTION**

110856       The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get*)  
 110857       substitutes for the %Z% keyword ("@( #) ") and shall write to standard output what follows  
 110858       until the first occurrence of one of the following:

110859           "   &gt;   newline   \   NUL

110860 **OPTIONS**110861       The *what* utility shall conform to XBD [Section 12.2](#) (on page 201).

110862       The following option shall be supported:

110863       -s           Quit after finding the first occurrence of the pattern in each file.

110864 **OPERANDS**

110865       The following operands shall be supported:

110866       *file*           A pathname of a file to search.110867 **STDIN**

110868       Not used.

110869 **INPUT FILES**

110870       The input files shall be of any file type.

110871 **ENVIRONMENT VARIABLES**110872       The following environment variables shall affect the execution of *what*:

110873       LANG           Provide a default value for the internationalization variables that are unset or null. |  
 110874                      (See XBD [Section 8.2](#) (on page 160) for the precedence of internationalization |  
 110875                      variables used to determine the values of locale categories.)

110876       LC\_ALL          If set to a non-empty string value, override the values of all the other  
 110877                      internationalization variables.

110878       LC\_CTYPE        Determine the locale for the interpretation of sequences of bytes of text data as  
 110879                      characters (for example, single-byte as opposed to multi-byte characters in  
 110880                      arguments and input files).

110881       LC\_MESSAGES     Determine the locale that should be used to affect the format and contents of  
 110882                      diagnostic messages written to standard error.  
 110883

110884       NLSPATH         Determine the location of message catalogs for the processing of LC\_MESSAGES.

110885 **ASYNCHRONOUS EVENTS**

110886       Default.

110887 **STDOUT**110888       The standard output shall consist of the following for each *file* operand:

110889       "%s:\n\t%s\n", &lt;pathname&gt;, &lt;identification string&gt;

110890 **STDERR**

110891 The standard error shall be used only for diagnostic messages.

110892 **OUTPUT FILES**

110893 None.

110894 **EXTENDED DESCRIPTION**

110895 None.

110896 **EXIT STATUS**

110897 The following exit values shall be returned:

110898 0 Any matches were found.

110899 1 Otherwise.

110900 **CONSEQUENCES OF ERRORS**

110901 Default.

110902 **APPLICATION USAGE**110903 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which  
110904 automatically inserts identifying information, but it can also be used where the information is  
110905 inserted by any other means.110906 When the string "@(#)" is included in a library routine in a shared library, it might not be found  
110907 in an **a.out** file using that library routine.110908 **EXAMPLES**110909 If the C-language program in file **f.c** contains:110910 `char ident[] = "@(#)identification information";`110911 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:110912 `what f.c f.o a.out`

110913 writes:

110914 `f.c:`  
110915 `identification information`110916 `...`110917 `f.o:`  
110918 `identification information`110919 `...`110920 `a.out:`  
110921 `identification information`110922 `...`110923 **RATIONALE**

110924 None.

110925 **FUTURE DIRECTIONS**

110926 None.

110927 **SEE ALSO**110928 *get*110929 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

+

110930 **CHANGE HISTORY**

110931 First released in Issue 2.

110932 **NAME**110933        **who** — display who is on the system110934 **SYNOPSIS**110935 XSI        **who** [-mTu] [-abdHlprt] [*file*]110936 XSI        **who** [-mu] -s [-bHlprt] [*file*]110937        **who** -q [*file*]110938        **who** am i110939        **who** am I110940 **DESCRIPTION**110941        The *who* utility shall list various pieces of information about accessible users. The domain of  
110942 accessibility is implementation-defined.110943 XSI        Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed  
110944 time since activity occurred on the line, and the process ID of the command interpreter for each  
110945 current system user.110946 **OPTIONS**110947        The *who* utility shall conform to XBD [Section 12.2](#) (on page 201).110948        The following options shall be supported. The metavariables, such as *<line>*, refer to fields  
110949 described in the STDOUT section.110950 XSI        **-a**        Process the implementation-defined database or named file with the **-b**, **-d**, **-l**, **-p**,  
110951 **-r**, **-t**, **-T** and **-u** options turned on.110952 XSI        **-b**        Write the time and date of the last system reboot. The system reboot time is the  
110953 time at which the implementation is able to commence running processes.110954 XSI        **-d**        Write a list of all processes that have expired and not been respawned by the *init*  
110955 system process. The *<exit>* field shall appear for dead processes and contain the  
110956 termination and exit values of the dead process. This can be useful in determining  
110957 why a process terminated.110958 XSI        **-H**        Write column headings above the regular output.110959 XSI        **-l**        (The letter ell.) List only those lines on which the system is waiting for someone to  
110960 login. The *<name>* field shall be **LOGIN** in such cases. Other fields shall be the  
110961 same as for user entries except that the *<state>* field does not exist.110962        **-m**        Output only information about the current terminal.110963 XSI        **-p**        List any other process that is currently active and has been previously spawned by  
110964 *init*.110965 XSI        **-q**        (Quick.) List only the names and the number of users currently logged on. When  
110966 this option is used, all other options shall be ignored.110967 XSI        **-r**        Write the current *run-level* of the *init* process.110968 XSI        **-s**        List only the *<name>*, *<line>*, and *<time>* fields. This is the default case.110969 XSI        **-t**        Indicate the last change to the system clock.

110970           **-T**           Show the state of each terminal, as described in the STDOUT section.

110971           **-u**           Write “idle time” for each displayed user in addition to any other information. The  
 110972 idle time is the time since any activity occurred on the user’s terminal. The method  
 110973 XSI of determining this is unspecified. This option shall list only those users who are  
 110974 currently logged in. The *<name>* is the user’s login name. The *<line>* is the name  
 110975 of the line as found in the directory */dev*. The *<time>* is the time that the user  
 110976 logged in. The *<activity>* is the number of hours and minutes since activity last  
 110977 occurred on that particular line. A dot indicates that the terminal has seen activity  
 110978 in the last minute and is therefore “current”. If more than twenty-four hours have  
 110979 elapsed or the line has not been used since boot time, the entry shall be marked  
 110980 *<old>*. This field is useful when trying to determine whether a person is working at  
 110981 the terminal or not. The *<pid>* is the process ID of the user’s login process.

**OPERANDS**

110982 XSI The following operands shall be supported:

110984 **am i, am I** In the POSIX locale, limit the output to describing the invoking user, equivalent to  
 110985 the **-m** option. The **am** and **i** or **I** must be separate arguments.

110986 *file* Specify a pathname of a file to substitute for the implementation-defined database  
 110987 of logged-on users that *who* uses by default.

**STDIN**

110988 Not used.

**INPUT FILES**

110990 None.

**ENVIRONMENT VARIABLES**

110993 The following environment variables shall affect the execution of *who*:

110994 **LANG** Provide a default value for the internationalization variables that are unset or null. |  
 110995 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 110996 variables used to determine the values of locale categories.)

110997 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
 110998 internationalization variables.

110999 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
 111000 characters (for example, single-byte as opposed to multi-byte characters in  
 111001 arguments).

111002 **LC\_MESSAGES**  
 111003 Determine the locale that should be used to affect the format and contents of  
 111004 diagnostic messages written to standard error.

111005 **LC\_TIME** Determine the locale used for the format and contents of the date and time strings.

111006 XSI **NLSPATH** Determine the location of message catalogs for the processing of **LC\_MESSAGES**.

111007 **TZ** Determine the timezone used when writing date and time information. If **TZ** is  
 111008 unset or null, an unspecified default timezone shall be used.

**ASYNCHRONOUS EVENTS**

111009 Default.

**STDOUT**

111012 The *who* utility shall write its default format to the standard output in an implementation-  
 111013 defined format, subject only to the requirement of containing the information described above.

111014 XSI OF XSI-conformant systems shall write the default information to the standard output in the  
 111015 following general format:

111016 <name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]

111017 For the **-b** option, <line> shall be "system boot". The <name> is unspecified.

111018 The following format shall be used for the **-T** option:

111019 "%s %c %s %s\n" <name>, <terminal state>, <terminal name>,  
111020 <time of login>

111021 where <terminal state> is one of the following characters:

111022 + The terminal allows write access to other users.

111023 - The terminal denies write access to other users.

111024 ? The terminal write-access state cannot be determined.

111025 <space> This entry is not associated with a terminal.

111026 In the POSIX locale, the <time of login> shall be equivalent in format to the output of:

111027 date +"%b %e %H:%M"

111028 If the **-u** option is used with **-T**, the idle time shall be added to the end of the previous format in  
111029 an unspecified format.

### 111030 **STDERR**

111031 The standard error shall be used only for diagnostic messages.

### 111032 **OUTPUT FILES**

111033 None.

### 111034 **EXTENDED DESCRIPTION**

111035 None.

### 111036 **EXIT STATUS**

111037 The following exit values shall be returned:

111038 0 Successful completion.

111039 >0 An error occurred.

### 111040 **CONSEQUENCES OF ERRORS**

111041 Default.

### 111042 **APPLICATION USAGE**

111043 The name *init* used for the system process is the most commonly used on historical systems, but  
111044 it may vary.

111045 The "domain of accessibility" referred to is a broad concept that permits interpretation either on  
111046 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

### 111047 **EXAMPLES**

111048 None.

### 111049 **RATIONALE**

111050 Due to differences between historical implementations, the base options provided were a  
111051 compromise to allow users to work with those functions. The standard developers also  
111052 considered removing all the options, but felt that these options offered users valuable  
111053 functionality. Additional options to match historical systems are available on XSI-conformant  
111054 systems.

111055 It is recognized that the *who* command may be of limited usefulness, especially in a multi-level  
111056 secure environment. The standard developers considered, however, that having some standard  
111057 method of determining the "accessibility" of other users would aid user portability.

111058 No format was specified for the default *who* output for systems not supporting the XSI option. In  
 111059 such a user-oriented command, designed only for human use, this was not considered to be a  
 111060 deficiency.

111061 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require  
 111062 that they use the same format.

111063 It is acceptable for an implementation to produce no output for an invocation of *who mil*.

#### 111064 **FUTURE DIRECTIONS**

111065 None.

#### 111066 **SEE ALSO**

111067 *mesg*

111068 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201) +

#### 111069 **CHANGE HISTORY**

111070 First released in Issue 2.

#### 111071 **Issue 6**

111072 This utility is marked as part of the User Portability Utilities option.

111073 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

#### 111074 **Issue 7**

111075 SD5-XCU-ERN-58 is applied, clarifying the **-b** option.

111076 The *who* utility is moved from the User Portability Utilities option to the Base. User Portability  
 111077 Utilities is now an option for interactive utilities.

111078 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

111079 **NAME**

111080 write — write to another user

111081 **SYNOPSIS**111082 write *user\_name* [*terminal*]111083 **DESCRIPTION**111084 The *write* utility shall read lines from the standard input and write them to the terminal of the  
111085 specified user. When first invoked, it shall write the message:111086 **Message from sender-login-id** (*sending-terminal*) [*date*]...111087 to *user\_name*. When it has successfully completed the connection, the sender's terminal shall be  
111088 alerted twice to indicate that what the sender is typing is being written to the recipient's  
111089 terminal.

111090 If the recipient wants to reply, this can be accomplished by typing:

111091 write *sender-login-id* [*sending-terminal*]111092 upon receipt of the initial message. Whenever a line of input as delimited by an NL, EOF, or  
111093 EOL special character (see XBD [Chapter 11](#), on page 185) is accumulated while in canonical  
111094 input mode, the accumulated data shall be written on the other user's terminal. Characters shall  
111095 be processed as follows:

- 111096 • Typing <alert> shall write the alert character to the recipient's terminal.
- 111097 • Typing the erase and kill characters shall affect the sender's terminal in the manner  
111098 described by the **termios** interface in XBD [Chapter 11](#) (on page 185).
- 111099 • Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate  
111100 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.
- 111101 • Typing characters from *LC\_CTYPE* classifications **print** or **space** shall cause those  
111102 characters to be sent to the recipient's terminal.
- 111103 • When and only when the *stty* **ixten** local mode is enabled, the existence and processing of  
111104 additional special control characters and multi-byte or single-byte functions is  
111105 implementation-defined.
- 111106 • Typing other non-printable characters shall cause implementation-defined sequences of  
111107 printable characters to be written to the recipient's terminal.

111108 To write to a user who is logged in more than once, the *terminal* argument can be used to  
111109 indicate which terminal to write to; otherwise, the recipient's terminal is selected in an  
111110 implementation-defined manner and an informational message is written to the sender's  
111111 standard output, indicating which terminal was chosen.

111112 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*  
111113 utility. However, a user's privilege may further constrain the domain of accessibility of other  
111114 users' terminals. The *write* utility shall fail when the user lacks the appropriate privileges to  
111115 perform the requested action.

111116 **OPTIONS**

111117 None.

111118 **OPERANDS**

111119 The following operands shall be supported:



|        |     |                               |                                                                                                                                                                                                                                                                                                            |
|--------|-----|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 111120 |     | <i>user_name</i>              | Login name of the person to whom the message shall be written. The application shall ensure that this operand is of the form returned by the <i>who</i> utility.                                                                                                                                           |
| 111121 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111122 |     | <i>terminal</i>               | Terminal identification in the same format provided by the <i>who</i> utility.                                                                                                                                                                                                                             |
| 111123 |     | <b>STDIN</b>                  |                                                                                                                                                                                                                                                                                                            |
| 111124 |     |                               | Lines to be copied to the recipient's terminal are read from standard input.                                                                                                                                                                                                                               |
| 111125 |     | <b>INPUT FILES</b>            |                                                                                                                                                                                                                                                                                                            |
| 111126 |     |                               | None.                                                                                                                                                                                                                                                                                                      |
| 111127 |     | <b>ENVIRONMENT VARIABLES</b>  |                                                                                                                                                                                                                                                                                                            |
| 111128 |     |                               | The following environment variables shall affect the execution of <i>write</i> :                                                                                                                                                                                                                           |
| 111129 |     | <i>LANG</i>                   | Provide a default value for the internationalization variables that are unset or null. (See XBD Section 8.2 (on page 160) for the precedence of internationalization variables used to determine the values of locale categories.)                                                                         |
| 111130 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111131 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111132 |     | <i>LC_ALL</i>                 | If set to a non-empty string value, override the values of all the other internationalization variables.                                                                                                                                                                                                   |
| 111133 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111134 |     | <i>LC_CTYPE</i>               | Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files). If the recipient's locale does not use an <i>LC_CTYPE</i> equivalent to the sender's, the results are undefined. |
| 111135 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111136 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111137 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111138 |     | <i>LC_MESSAGES</i>            |                                                                                                                                                                                                                                                                                                            |
| 111139 |     |                               | Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.                                                                                                                           |
| 111140 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111141 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111142 | XSI | <i>NLSPATH</i>                | Determine the location of message catalogs for the processing of <i>LC_MESSAGES</i> .                                                                                                                                                                                                                      |
| 111143 |     | <b>ASYNCHRONOUS EVENTS</b>    |                                                                                                                                                                                                                                                                                                            |
| 111144 |     |                               | If an interrupt signal is received, <i>write</i> shall write an appropriate message on the recipient's terminal and exit with a status of zero. It shall take the standard action for all other signals.                                                                                                   |
| 111145 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111146 |     | <b>STDOUT</b>                 |                                                                                                                                                                                                                                                                                                            |
| 111147 |     |                               | An informational message shall be written to standard output if a recipient is logged in more than once.                                                                                                                                                                                                   |
| 111148 |     |                               |                                                                                                                                                                                                                                                                                                            |
| 111149 |     | <b>STDERR</b>                 |                                                                                                                                                                                                                                                                                                            |
| 111150 |     |                               | The standard error shall be used only for diagnostic messages.                                                                                                                                                                                                                                             |
| 111151 |     | <b>OUTPUT FILES</b>           |                                                                                                                                                                                                                                                                                                            |
| 111152 |     |                               | The recipient's terminal is used for output.                                                                                                                                                                                                                                                               |
| 111153 |     | <b>EXTENDED DESCRIPTION</b>   |                                                                                                                                                                                                                                                                                                            |
| 111154 |     |                               | None.                                                                                                                                                                                                                                                                                                      |
| 111155 |     | <b>EXIT STATUS</b>            |                                                                                                                                                                                                                                                                                                            |
| 111156 |     |                               | The following exit values shall be returned:                                                                                                                                                                                                                                                               |
| 111157 |     | 0                             | Successful completion.                                                                                                                                                                                                                                                                                     |
| 111158 |     | >0                            | The addressed user is not logged on or the addressed user denies permission.                                                                                                                                                                                                                               |
| 111159 |     | <b>CONSEQUENCES OF ERRORS</b> |                                                                                                                                                                                                                                                                                                            |
| 111160 |     |                               | Default.                                                                                                                                                                                                                                                                                                   |

**APPLICATION USAGE**

The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.

**EXAMPLES**

None.

**RATIONALE**

The *write* utility was included in this volume of POSIX.1-200x since it can be implemented on all terminal types. The standard developers considered the *talk* utility, which cannot be implemented on certain terminals, to be a “better” communications interface. Both of these programs are in widespread use on historical implementations. Therefore, the standard developers decided that both utilities should be specified.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write* require that they all use or accept the same format.

**FUTURE DIRECTIONS**

None.

**SEE ALSO**

*mesg*, *talk*, *who*

XBD [Chapter 8](#) (on page 159), [Chapter 11](#) (on page 185)

**CHANGE HISTORY**

First released in Issue 2.

**Issue 5**

The FUTURE DIRECTIONS section is added.

**Issue 6**

This utility is marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term “must” for application requirements.

**Issue 7**

The *write* utility is moved from the User Portability Utilities option to the Base. User Portability Utilities is now an option for interactive utilities.

111188 **NAME**111189 `xargs` — construct argument lists and invoke utility111190 **SYNOPSIS**111191 XSI `xargs [-ptx] [-E eofstr] [-I replstr|-L number|-n number]`  
111192 `[-s size] [utility [argument...]]`111193 **DESCRIPTION**111194 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands  
111195 specified followed by as many arguments read in sequence from standard input as fit in length  
111196 and number constraints specified by the options. The *xargs* utility shall then invoke the  
111197 constructed command line and wait for its completion. This sequence shall be repeated until one  
111198 of the following occurs:

- 111199
- An end-of-file condition is detected on standard input.
  - An argument consisting of just the logical end-of-file string (see the `-E eofstr` option) is  
111200 found on standard input after double-quote processing, apostrophe processing, and  
111201 backslash escape processing (see next paragraph). All arguments up to but not including  
111202 the argument consisting of just the logical end-of-file string shall be used as arguments in  
111203 constructed command lines.
  - An invocation of a constructed command line returns an exit status of 255.

111206 The application shall ensure that arguments in the standard input are separated by unquoted  
111207 `<blank>s`, unescaped `<blank>s`, or `<newline>s`. A string of zero or more non-double-quote  
111208 (`'` `"` `'`) characters and non-`<newline>s` can be quoted by enclosing them in double-quotes. A  
111209 string of zero or more non-apostrophe (`'` `'` `'`) characters and non-`<newline>s` can be quoted by  
111210 enclosing them in apostrophes. Any unquoted character can be escaped by preceding it with a  
111211 backslash. The utility named by *utility* shall be executed one or more times until the end-of-file is  
111212 reached or the logical end-of file string is found. The results are unspecified if the utility named  
111213 by *utility* attempts to read from its standard input.111214 The generated command line length shall be the sum of the size in bytes of the utility name and  
111215 each argument treated as strings, including a null byte terminator for each of these strings. The  
111216 *xargs* utility shall limit the command line length such that when the command line is invoked,  
111217 the combined argument and environment lists (see the *exec* family of functions in the System  
111218 Interfaces volume of POSIX.1-200x) shall not exceed `{ARG_MAX}-2048` bytes. Within this  
111219 constraint, if neither the `-n` nor the `-s` option is specified, the default command line length shall  
111220 be at least `{LINE_MAX}`.111221 **OPTIONS**111222 The *xargs* utility shall conform to XBD [Section 12.2](#) (on page 201).

111223 The following options shall be supported:

111224 `-E eofstr` Use *eofstr* as the logical end-of-file string. If `-E` is not specified, it is unspecified  
111225 whether the logical end-of-file string is the underscore character (`'_'`) or the end-  
111226 of-file string capability is disabled. When *eofstr* is the null string, the logical end-of-  
111227 file string capability shall be disabled and underscore characters shall be taken  
111228 literally.111229 XSI `-I replstr` Insert mode: *utility* is executed for each line from standard input, taking the entire  
111230 line as a single argument, inserting it in *arguments* for each occurrence of *replstr*. A  
111231 maximum of five arguments in *arguments* can each contain one or more instances  
111232 of *replstr*. Any `<blank>s` at the beginning of each line shall be ignored.  
111233 Constructed arguments cannot grow larger than 255 bytes. Option `-x` shall be

|        |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 111234 |     | forced on.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 111235 | XSI | <b>-L <i>number</i></b> The <i>utility</i> shall be executed for each non-empty <i>number</i> lines of arguments from standard input. The last invocation of <i>utility</i> shall be with fewer lines of arguments if fewer than <i>number</i> remain. A line is considered to end with the first <newline> unless the last character of the line is a <blank>; a trailing <blank> signals continuation to the next non-empty line, inclusive. |
| 111236 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111237 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111238 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111239 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111240 |     | <b>-n <i>number</i></b> Invoke <i>utility</i> using as many standard input arguments as possible, up to <i>number</i> (a positive decimal integer) arguments maximum. Fewer arguments shall be used if:                                                                                                                                                                                                                                        |
| 111241 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111242 |     | • The command line length accumulated exceeds the size specified by the <b>-s</b> option (or {LINE_MAX} if there is no <b>-s</b> option).                                                                                                                                                                                                                                                                                                      |
| 111243 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111244 |     | • The last iteration has fewer than <i>number</i> , but not zero, operands remaining.                                                                                                                                                                                                                                                                                                                                                          |
| 111245 |     | <b>-p</b> Prompt mode: the user is asked whether to execute <i>utility</i> at each invocation. Trace mode ( <b>-t</b> ) is turned on to write the command instance to be executed, followed by a prompt to standard error. An affirmative response read from <b>/dev/tty</b> shall execute the command; otherwise, that particular invocation of <i>utility</i> shall be skipped.                                                              |
| 111246 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111247 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111248 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111249 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111250 |     | <b>-s <i>size</i></b> Invoke <i>utility</i> using as many standard input arguments as possible yielding a command line length less than <i>size</i> (a positive decimal integer) bytes. Fewer arguments shall be used if:                                                                                                                                                                                                                      |
| 111251 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111252 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111253 |     | • The total number of arguments exceeds that specified by the <b>-n</b> option.                                                                                                                                                                                                                                                                                                                                                                |
| 111254 | XSI | • The total number of lines exceeds that specified by the <b>-L</b> option.                                                                                                                                                                                                                                                                                                                                                                    |
| 111255 |     | • End-of-file is encountered on standard input before <i>size</i> bytes are accumulated.                                                                                                                                                                                                                                                                                                                                                       |
| 111256 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111257 |     | Values of <i>size</i> up to at least {LINE_MAX} bytes shall be supported, provided that the constraints specified in the DESCRIPTION are met. It shall not be considered an error if a value larger than that supported by the implementation or exceeding the constraints specified in the DESCRIPTION is given; <i>xargs</i> shall use the largest value it supports within the constraints.                                                 |
| 111258 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111259 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111260 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111261 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111262 |     | <b>-t</b> Enable trace mode. Each generated command line shall be written to standard error just prior to invocation.                                                                                                                                                                                                                                                                                                                          |
| 111263 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 111264 |     | <b>-x</b> Terminate if a constructed command line will not fit in the implied or specified size (see the <b>-s</b> option above).                                                                                                                                                                                                                                                                                                              |
| 111265 |     |                                                                                                                                                                                                                                                                                                                                                                                                                                                |

**OPERANDS**

The following operands shall be supported:

|        |                 |                                                                                                                                                                                                                                                                                                                                                                    |
|--------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 111266 |                 |                                                                                                                                                                                                                                                                                                                                                                    |
| 111267 |                 |                                                                                                                                                                                                                                                                                                                                                                    |
| 111268 | <i>utility</i>  | The name of the utility to be invoked, found by search path using the <i>PATH</i> environment variable, described in XBD Chapter 8 (on page 159). If <i>utility</i> is omitted, the default shall be the <i>echo</i> utility. If the <i>utility</i> operand names any of the special built-in utilities in Section 2.14 (on page 2280), the results are undefined. |
| 111269 |                 |                                                                                                                                                                                                                                                                                                                                                                    |
| 111270 |                 |                                                                                                                                                                                                                                                                                                                                                                    |
| 111271 |                 |                                                                                                                                                                                                                                                                                                                                                                    |
| 111272 | <i>argument</i> | An initial option or operand for the invocation of <i>utility</i> .                                                                                                                                                                                                                                                                                                |

**STDIN**

The standard input shall be a text file. The results are unspecified if an end-of-file condition is detected immediately following an escaped <newline>.

111276 **INPUT FILES**111277 The file `/dev/tty` shall be used to read responses required by the `-p` option.111278 **ENVIRONMENT VARIABLES**111279 The following environment variables shall affect the execution of *xargs*:111280 **LANG** Provide a default value for the internationalization variables that are unset or null. |  
111281 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
111282 variables used to determine the values of locale categories.)111283 **LC\_ALL** If set to a non-empty string value, override the values of all the other  
111284 internationalization variables.111285 **LC\_COLLATE**111286 Determine the locale for the behavior of ranges, equivalence classes, and multi-  
111287 character collating elements used in the extended regular expression defined for  
111288 the **yesexpr** locale keyword in the *LC\_MESSAGES* category.111289 **LC\_CTYPE** Determine the locale for the interpretation of sequences of bytes of text data as  
111290 characters (for example, single-byte as opposed to multi-byte characters in  
111291 arguments and input files) and the behavior of character classes used in the  
111292 extended regular expression defined for the **yesexpr** locale keyword in the  
111293 *LC\_MESSAGES* category.111294 **LC\_MESSAGES**111295 Determine the locale for the processing of affirmative responses and that should be  
111296 used to affect the format and contents of diagnostic messages written to standard  
111297 error.111298 XSI **NLSPATH** Determine the location of message catalogs for the processing of *LC\_MESSAGES*.111299 **PATH** Determine the location of *utility*, as described in XBD Chapter 8 (on page 159). |111300 **ASYNCHRONOUS EVENTS**

111301 Default.

111302 **STDOUT**

111303 Not used.

111304 **STDERR**111305 The standard error shall be used for diagnostic messages and the `-t` and `-p` options. If the `-t`  
111306 option is specified, the *utility* and its constructed argument list shall be written to standard error,  
111307 as it will be invoked, prior to invocation. If `-p` is specified, a prompt of the following format  
111308 shall be written (in the POSIX locale):

111309 " ? . . . "

111310 at the end of the line of the output from `-t`.111311 **OUTPUT FILES**

111312 None.

111313 **EXTENDED DESCRIPTION**

111314 None.

111315 **EXIT STATUS**

111316 The following exit values shall be returned:

111317 0 All invocations of *utility* returned exit status zero.111318 1-125 A command line meeting the specified requirements could not be assembled, one or  
111319 more of the invocations of *utility* returned a non-zero exit status, or some other error  
111320 occurred.

111321 126 The utility specified by *utility* was found but could not be invoked.

111322 127 The utility specified by *utility* could not be found.

### 111323 CONSEQUENCES OF ERRORS

111324 If a command line meeting the specified requirements cannot be assembled, the utility cannot be  
 111325 invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits  
 111326 with exit status 255, the *xargs* utility shall write a diagnostic message and exit without  
 111327 processing any remaining input.

### 111328 APPLICATION USAGE

111329 The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no  
 111330 further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit*  
 111331 with an appropriate value to avoid accidentally returning with 255.

111332 Note that since input is parsed as lines, <blank>s separate arguments, and backslash,  
 111333 apostrophe, and double-quote characters are used for quoting, if *xargs* is used to bundle the  
 111334 output of commands like *find dir -print* or *ls* into commands to be executed, unexpected results  
 111335 are likely if any filenames contain <blank>s, <newline>s, or quoting characters. This can be  
 111336 solved by using *find* to call a script that converts each file found into a quoted string that is then  
 111337 piped to *xargs*, but in most cases it is preferable just to have *find* do the argument aggregation  
 111338 itself by using *-exec* with a '+' terminator instead of ';' . Note that the quoting rules used by  
 111339 *xargs* are not the same as in the shell. They were not made consistent here because existing  
 111340 applications depend on the current rules. An easy (but inefficient) method that can be used to  
 111341 transform input consisting of one argument per line into a quoted form that *xargs* interprets  
 111342 correctly is to precede each non-<newline> character with a backslash. More efficient  
 111343 alternatives are shown in Example 2 and Example 5 below.

111344 On implementations with a large value for {ARG\_MAX}, *xargs* may produce command lines  
 111345 longer than {LINE\_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used  
 111346 to create a text file, users should explicitly set the maximum command line length with the *-s*  
 111347 option.

111348 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if  
 111349 an error occurs so that applications can distinguish "failure to find a utility" from "invoked  
 111350 utility exited with an error indication". The value 127 was chosen because it is not commonly  
 111351 used for other meanings; most utilities use small values for "normal error conditions" and the  
 111352 values above 128 can be confused with termination due to receipt of a signal. The value 126 was  
 111353 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some  
 111354 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction  
 111355 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to  
 111356 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for  
 111357 any other reason.

### 111358 EXAMPLES

111359 1. The following command combines the output of the parenthesized commands (minus the  
 111360 apostrophes) onto one line, which is then appended to the file *log*. It assumes that the  
 111361 expansion of "\$0 \$\*" does not include any apostrophes or <newline>s.

```
111362 (logname; date; printf "'%s'\n" "$0 $*") | xargs -E "" >>log
```

111363 2. The following command invokes *diff* with successive pairs of arguments originally typed  
 111364 as command line arguments. It assumes there are no embedded <newline>s in the  
 111365 elements of the original argument list.

```
111366 printf "%s\n" "$@" | sed 's/^[^:alnum:]]/\\&/g' |

 111367 xargs -E "" -n 2 -x diff
```

3. In the following commands, the user is asked which files in the current directory (excluding dotfiles) are to be archived. The files are archived into **arch**; *a*, one at a time or *b*, many at a time. The commands assume that no filenames contain <blank>s, <newline>s, backslashes, apostrophes, or double-quote characters.

```
a. ls | xargs -E "" -p -L 1 ar -r arch
```

```
b. ls | xargs -E "" -p -L 1 | xargs -E "" ar -r arch
```

4. The following command invokes *command1* one or more times with multiple arguments, stopping if an invocation of *command1* has a non-zero exit status.

```
xargs -E "" sh -c 'command1 "$@" || exit 255' sh < xargs_input
```

5. On XSI-conformant systems, the following command moves all files from directory **\$1** to directory **\$2**, and echoes each move command just before doing it. It assumes no filenames contain <newline>s and that neither **\$1** nor **\$2** contains the sequence "{ }".

```
ls -A "$1" | sed -e 's/"/\\"/g' -e 's/.*"/&/' |
xargs -E "" -I {} -t mv "$1"/{} "$2"/{}
```

## RATIONALE

The *xargs* utility was usually found only in System V-based systems; BSD systems included an *apply* utility that provided functionality similar to *xargs -n number*. The SVID lists *xargs* as a software development extension. This volume of POSIX.1-200x does not share the view that it is used only for development, and therefore it is not optional.

The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the number of processes launched by a simplistic use of the *find -exec* combination. The *xargs* utility is also used to enforce an upper limit on memory required to launch a process. With this basis in mind, this volume of POSIX.1-200x selected only the minimal features required.

Although the 255 exit status is mostly an accident of historical implementations, it allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125 range when *xargs* exits. There is no statement of how the various non-zero utility exit status codes are accumulated by *xargs*. The value could be the addition of all codes, their highest value, the last one received, or a single value such as 1. Since no algorithm is arguably better than the others, and since many of the standard utilities say little more (portably) than "pass/fail", no new algorithm was invented.

Several other *xargs* options were removed because simple alternatives already exist within this volume of POSIX.1-200x. For example, the *-i replstr* option can be just as efficiently performed using a shell **for** loop. Since *xargs* calls an *exec* function with each input line, the *-i* option does not usually exploit the grouping capabilities of *xargs*.

The requirement that *xargs* never produces command lines such that invocation of *utility* is within 2048 bytes of hitting the POSIX *exec* {ARG\_MAX} limitations is intended to guarantee that the invoked utility has room to modify its environment variables and command line arguments and still be able to invoke another utility. Note that the minimum {ARG\_MAX} allowed by the System Interfaces volume of POSIX.1-200x is 4096 bytes and the minimum value allowed by this volume of POSIX.1-200x is 2048 bytes; therefore, the 2048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a utility if the environment passed in to *xargs* comes close to using {ARG\_MAX} bytes.

The version of *xargs* required by this volume of POSIX.1-200x is required to wait for the completion of the invoked command before invoking another command. This was done because historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide parallel operation of the invoked utilities are encouraged to add an option enabling parallel invocation, but should still wait for termination of all of the children before *xargs* terminates

111416 normally.

111417 The `-e` option was omitted from the ISO POSIX-2:1993 standard in the belief that the `eofstr`  
 111418 option-argument was recognized only when it was on a line by itself and before quote and  
 111419 escape processing were performed, and that the logical end-of-file processing was only enabled  
 111420 if a `-e` option was specified. In that case, a simple `sed` script could be used to duplicate the `-e`  
 111421 functionality. Further investigation revealed that:

- 111422 • The logical end-of-file string was checked for after quote and escape processing, making a  
 111423 `sed` script that provided equivalent functionality much more difficult to write.
- 111424 • The default was to perform logical end-of-file processing with an underscore as the logical  
 111425 end-of-file string.

111426 To correct this misunderstanding, the `-E eofstr` option was adopted from the X/Open Portability  
 111427 Guide. Users should note that the description of the `-E` option matches historical documentation  
 111428 of the `-e` option (which was not adopted because it did not support the Utility Syntax  
 111429 Guidelines), by saying that if `eofstr` is the null string, logical end-of-file processing is disabled.  
 111430 Historical implementations of `xargs` actually did not disable logical end-of-file processing; they  
 111431 treated a null argument found in the input as a logical end-of-file string. (A null `string` argument  
 111432 could be generated using single or double quotes ( ' ' or " " ). Since this behavior was not  
 111433 documented historically, it is considered to be a bug.

111434 The `-I`, `-L`, and `-n` options are mutually-exclusive. Some implementations use the last one  
 111435 specified if more than one is given on a command line; other implementations treat  
 111436 combinations of the options in different ways.

#### 111437 FUTURE DIRECTIONS

111438 None.

#### 111439 SEE ALSO

111440 [Chapter 2](#) (on page 2245), [diff](#), [echo](#), [find](#)

111441 XBD [Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

111442 XSH [exec](#)

#### 111443 CHANGE HISTORY

111444 First released in Issue 2.

##### 111445 Issue 5

111446 A second FUTURE DIRECTION is added.

##### 111447 Issue 6

111448 The obsolescent `-e`, `-i`, and `-l` options are removed.

111449 The following new requirements on POSIX implementations derive from alignment with the  
 111450 Single UNIX Specification:

- 111451 • The `-p` option is added.
- 111452 • In the INPUT FILES section, the file `/dev/tty` is used to read responses required by the `-p`  
 111453 option.
- 111454 • The STDERR section is updated to describe the `-p` option.

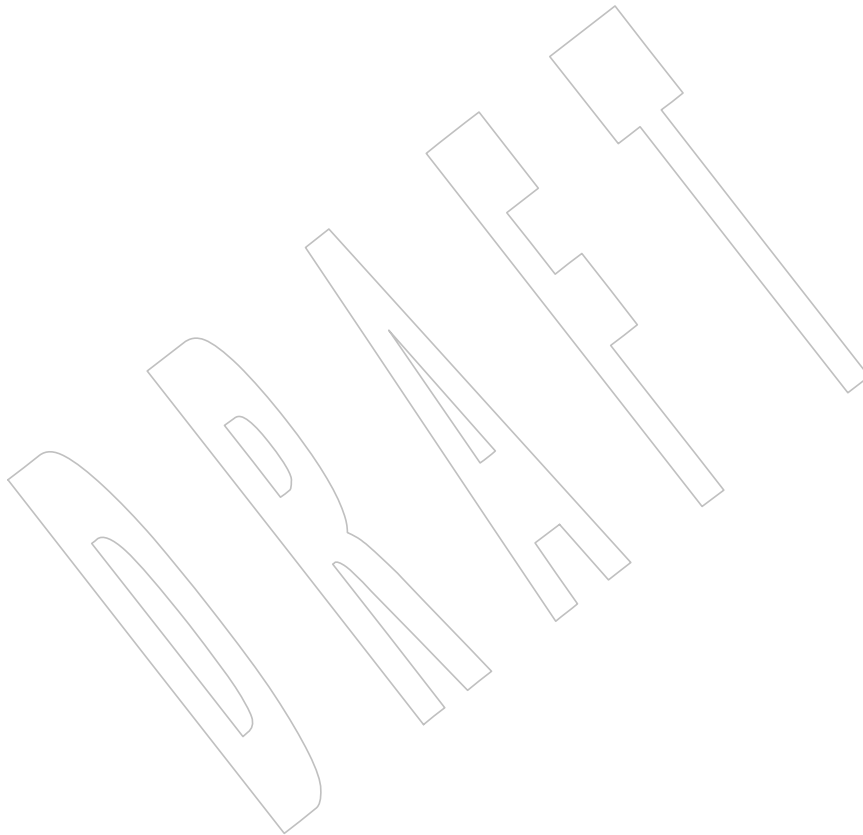
111455 The description of the `-E` option is aligned with the ISO POSIX-2:1993 standard.

111456 The normative text is reworded to avoid use of the term “must” for application requirements.



|        |                |                                                                                           |
|--------|----------------|-------------------------------------------------------------------------------------------|
| 111457 | <b>Issue 7</b> |                                                                                           |
| 111458 |                | SD5-XCU-ERN-68 is applied.                                                                |
| 111459 |                | SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.                                         |
| 111460 |                | SD5-XCU-ERN-128 is applied, clarifying the DESCRIPTION of the logical end-of-file string. |
| 111461 |                | SD5-XCU-ERN-132 is applied, updating the EXAMPLES section.                                |

+



111462 **NAME**111463 yacc — yet another compiler compiler (**DEVELOPMENT**)111464 **SYNOPSIS**111465 CD yacc [-dltv] [-b *file\_prefix*] [-p *sym\_prefix*] *grammar*111466 **DESCRIPTION**

111467 The *yacc* utility shall read a description of a context-free grammar in *grammar* and write C source  
 111468 code, conforming to the ISO C standard, to a code file, and optionally header information into a  
 111469 header file, in the current directory. The C code shall define a function and related routines and  
 111470 macros for an automaton that executes a parsing algorithm meeting the requirements in  
 111471 [Algorithms](#) (on page 3301).

111472 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

111473 The C source code and header file shall be produced in a form suitable as input for the C  
 111474 compiler (see [c99](#)).

111475 **OPTIONS**

111476 The *yacc* utility shall conform to XBD [Section 12.2](#) (on page 201), except for Guideline 9.

111477 The following options shall be supported:

111478 **-b** *file\_prefix* Use *file\_prefix* instead of *y* as the prefix for all output filenames. The code file  
 111479 **y.tab.c**, the header file **y.tab.h** (created when **-d** is specified), and the description  
 111480 file **y.output** (created when **-v** is specified), shall be changed to *file\_prefix.tab.c*,  
 111481 *file\_prefix.tab.h*, and *file\_prefix.output*, respectively.

111482 **-d** Write the header file; by default only the code file is written. The **#define**  
 111483 statements associate the token codes assigned by *yacc* with the user-declared token  
 111484 names. This allows source files other than **y.tab.c** to access the token codes.

111485 **-l** Produce a code file that does not contain any **#line** constructs. If this option is not  
 111486 present, it is unspecified whether the code file or header file contains **#line**  
 111487 directives. This should only be used after the grammar and the associated actions  
 111488 are fully debugged.

111489 **-p** *sym\_prefix* Use *sym\_prefix* instead of **yy** as the prefix for all external names produced by *yacc*.  
 111490 The names affected shall include the functions *yyparse()*, *yylex()*, and *yyerror()*,  
 111491 and the variables *yyval*, *yychar*, and *yydebug*. (In the remainder of this section, the  
 111492 six symbols cited are referenced using their default names only as a notational  
 111493 convenience.) Local names may also be affected by the **-p** option; however, the **-p**  
 111494 option shall not affect **#define** symbols generated by *yacc*.

111496 **-t** Modify conditional compilation directives to permit compilation of debugging  
 111497 code in the code file. Runtime debugging statements shall always be contained in  
 111498 the code file, but by default conditional compilation directives prevent their  
 111499 compilation.

111500 **-v** Write a file containing a description of the parser and a report of conflicts  
 111501 generated by ambiguities in the grammar.

111502 **OPERANDS**

111503 The following operand is required:

111504 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a  
 111505 parser is to be created. The format for the grammar is described in the EXTENDED  
 111506 DESCRIPTION section.

111507 **STDIN**

111508 Not used.

111509 **INPUT FILES**111510 The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION  
 111511 section.111512 **ENVIRONMENT VARIABLES**111513 The following environment variables shall affect the execution of *yacc*:

111514 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
 111515 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 111516 variables used to determine the values of locale categories.)

111517 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 111518 internationalization variables.

111519 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 111520 characters (for example, single-byte as opposed to multi-byte characters in  
 111521 arguments and input files).

111522 *LC\_MESSAGES*

111523 Determine the locale that should be used to affect the format and contents of  
 111524 diagnostic messages written to standard error.

111525 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

111526 The *LANG* and *LC\_\** variables affect the execution of the *yacc* utility as stated. The *main()*  
 111527 function defined in *Yacc Library* (on page 3300) shall call:

111528 `setlocale(LC_ALL, "")`

111529 and thus the program generated by *yacc* shall also be affected by the contents of these variables  
 111530 at runtime.

111531 **ASYNCHRONOUS EVENTS**

111532 Default.

111533 **STDOUT**

111534 Not used.

111535 **STDERR**

111536 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of  
 111537 those conflicts to the standard error in an unspecified format.

111538 Standard error shall also be used for diagnostic messages.

111539 **OUTPUT FILES**

111540 The code file, the header file, and the description file shall be text files. All are described in the  
 111541 following sections.

111542 **Code File**

111543 This file shall contain the C source code for the *yyparse()* function. It shall contain code for the  
 111544 various semantic actions with macro substitution performed on them as described in the  
 111545 EXTENDED DESCRIPTION section. It also shall contain a copy of the **#define** statements in the  
 111546 header file. If a **%union** declaration is used, the declaration for YYSTYPE shall also be included  
 111547 in this file.

111548 **Header File**

111549 The header file shall contain **#define** statements that associate the token numbers with the token  
 111550 names. This allows source files other than the code file to access the token codes. If a **%union**  
 111551 declaration is used, the declaration for YYSTYPE and an *extern YYSTYPE yylval* declaration shall  
 111552 also be included in this file.

111553 **Description File**

111554 The description file shall be a text file containing a description of the state machine  
 111555 corresponding to the parser, using an unspecified format. Limits for internal tables (see [Limits](#),  
 111556 on page 3301) shall also be reported, in an implementation-defined manner. (Some  
 111557 implementations may use dynamic allocation techniques and have no specific limit values to  
 111558 report.)

111559 **EXTENDED DESCRIPTION**

111560 The *yacc* command accepts a language that is used to define a grammar for a target language to  
 111561 be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a  
 111562 grammar for the target language is described below using the *yacc* input language itself.

111563 The input *grammar* includes rules describing the input structure of the target language and code  
 111564 to be invoked when these rules are recognized to provide the associated semantic action. The  
 111565 code to be executed shall appear as bodies of text that are intended to be C-language code. The  
 111566 C-language inclusions are presumed to form a correct function when processed by *yacc* into its  
 111567 output files. The code included in this way shall be executed during the recognition of the target  
 111568 language.

111569 Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section.  
 111570 The code file can be compiled and linked using *c99*. If the declaration and programs sections of  
 111571 the grammar file did not include definitions of *main()*, *yylex()*, and *yyerror()*, the compiled  
 111572 output requires linking with externally supplied versions of those functions. Default versions of  
 111573 *main()* and *yyerror()* are supplied in the *yacc* library and can be linked in by using the **-ly**  
 111574 operand to *c99*. The *yacc* library interfaces need not support interfaces with other than the  
 111575 default **yy** symbol prefix. The application provides the lexical analyzer function, *yylex()*; the *lex*  
 111576 utility is specifically designed to generate such a routine.

111577 **Input Language**

111578 The application shall ensure that every specification file consists of three sections in order:  
 111579 *declarations*, *grammar rules*, and *programs*, separated by double percent signs ("%"). The  
 111580 declarations and programs sections can be empty. If the latter is empty, the preceding "%"  
 111581 mark separating it from the rules section can be omitted.

111582 The input is free form text following the structure of the grammar defined below.

111583 **Lexical Structure of the Grammar**

111584 The <blank>s, <newline>s, and <form-feed>s shall be ignored, except that the application shall  
 111585 ensure that they do not appear in names or multi-character reserved symbols. Comments shall  
 111586 be enclosed in `"/* . . . */"`, and can appear wherever a name is valid.

111587 Names are of arbitrary length, made up of letters, periods ('.'), underscores ('\_'), and non-  
 111588 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not  
 111589 use names beginning in `yy` or `YY` since the *yacc* parser uses such names. Many of the names  
 111590 appear in the final output of *yacc*, and thus they should be chosen to conform with any  
 111591 additional rules created by the C compiler to be used. In particular they appear in `#define`  
 111592 statements.

111593 A literal shall consist of a single character enclosed in single-quotes (''). All of the escape  
 111594 sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

111595 The relationship with the lexical analyzer is discussed in detail below.

111596 The application shall ensure that the NUL character is not used in grammar rules or literals.

111597 **Declarations Section**

111598 The declarations section is used to define the symbols used to define the target language and  
 111599 their relationship with each other. In particular, much of the additional information required to  
 111600 resolve ambiguities in the context-free grammar for the target language is provided here.

111601 Usually *yacc* assigns the relationship between the symbolic names it generates and their  
 111602 underlying numeric value. The declarations section makes it possible to control the assignment  
 111603 of these values.

111604 It is also possible to keep semantic information associated with the tokens currently on the parse  
 111605 stack in a user-defined C-language **union**, if the members of the union are associated with the  
 111606 various names in the grammar. The declarations section provides for this as well.

111607 The first group of declarators below all take a list of names as arguments. That list can optionally  
 111608 be preceded by the name of a C union member (called a *tag* below) appearing within '`<`' and  
 111609 '`>`'. (As an exception to the typographical conventions of the rest of this volume of  
 111610 POSIX.1-200x, in this case `<tag>` does not represent a metavariable, but the literal angle bracket  
 111611 characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this line  
 111612 shall be of the same C type as the union member referenced by *tag*. This is discussed in more  
 111613 detail below.

111614 For lists used to define tokens, the first appearance of a given token can be followed by a  
 111615 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it  
 111616 for lexical purposes shall be taken to be that number.

111617 The following declares *name* to be a token:

```
111618 %token [<tag>] name [number] [name [number]]...
```

111619 If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced  
 111620 by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

111621 The following declares *name* to be a token, and assigns precedence to it:

```
111622 %left [<tag>] name [number] [name [number]]...
```

```
111623 %right [<tag>] name [number] [name [number]]...
```

111624 One or more lines, each beginning with one of these symbols, can appear in this section. All  
 111625 tokens on the same line have the same precedence level and associativity; the lines are in order  
 111626 of increasing precedence or binding strength. `%left` denotes that the operators on that line are  
 111627 left associative, and `%right` similarly denotes right associative operators. If *tag* is present, it shall

111628 declare a C type for *names* as described for **%token**.

111629 The following declares *name* to be a token, and indicates that this cannot be used associatively:

111630 `%nonassoc [<tag>] name [number] [name [number]]...`

111631 If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall

111632 declare a C type for *names* as described for **%token**.

111633 The following declares that union member *names* are non-terminals, and thus it is required to

111634 have a *tag* field at its beginning:

111635 `%type <tag> name...`

111636 Because it deals with non-terminals only, assigning a token number or using a literal is also

111637 prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not

111638 present, the parse stack shall hold only the **int** type.

111639 Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration

111640 is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any non-

111641 terminal symbol that does not appear on the left side of at least one grammar rule.

111642 Once the type, precedence, or token number of a name is specified, it shall not be changed. If the

111643 first declaration of a token does not assign a token number, *yacc* shall assign a token number.

111644 Once this assignment is made, the token number shall not be changed by explicit assignment.

111645 The following declarators do not follow the previous pattern.

111646 The following declares the non-terminal *name* to be the *start symbol*, which represents the largest,

111647 most general structure described by the grammar rules:

111648 `%start name`

111649 By default, it is the left-hand side of the first grammar rule; this default can be overridden with

111650 this declaration.

111651 The following declares the *yacc* value stack to be a union of the various types of values desired:

111652 `%union { body of union (in C) }`

111653 By default, the values returned by actions (see below) and the lexical analyzer shall be of type

111654 **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names

111655 in order to perform strict type checking of the resulting parser.

111656 Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a

111657 header file (which shall be included in the declarations section by using a **#include** construct

111658 within `{` and `}`), and a **typedef** used to define the symbol `YYSTYPE` to represent this union.

111659 The effect of **%union** is to provide the declaration of `YYSTYPE` directly from the *yacc* input.

111660 C-language declarations and definitions can appear in the declarations section, enclosed by the

111661 following marks:

111662 `{ ... }`

111663 These statements shall be copied into the code file, and have global scope within it so that they

111664 can be used in the rules and program sections.

111665 The application shall ensure that the declarations section is terminated by the token `%%`.

111666 **Grammar Rules in yacc**

111667 The rules section defines the context-free grammar to be accepted by the function *yacc* generates,  
 111668 and associates with those rules C-language actions and additional precedence information. The  
 111669 grammar is described below, and a formal definition follows.

111670 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

111671 A : BODY ;

111672 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or  
 111673 more *names*, *literals*, and *semantic actions* that can then be followed by optional *precedence rules*.  
 111674 Only the names and literals participate in the formation of the grammar; the semantic actions  
 111675 and precedence rules are used in other ways. The colon and the semicolon are *yacc* punctuation.  
 111676 If there are several successive grammar rules with the same left-hand side, the vertical bar ' | '  
 111677 can be used to avoid rewriting the left-hand side; in this case the semicolon appears only after  
 111678 the last rule. The BODY part can be empty (or empty of names and literals) to indicate that the  
 111679 non-terminal symbol matches the empty string.

111680 The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are  
 111681 distinct rules. The number assigned to the rule appears in the description file.

111682 The elements comprising a BODY are:

111683 *name, literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*  
 111684 stands for itself (less the lexically required quotation marks).

111685 *semantic action*

111686 With each grammar rule, the user can associate actions to be performed each time  
 111687 the rule is recognized in the input process. (Note that the word "action" can also  
 111688 refer to the actions of the parser—shift, reduce, and so on.)

111689 These actions can return values and can obtain the values returned by previous  
 111690 actions. These values are kept in objects of type YYSTYPE (see %union). The  
 111691 result value of the action shall be kept on the parse stack with the left-hand side of  
 111692 the rule, to be accessed by other reductions as part of their right-hand side. By  
 111693 using the <tag> information provided in the declarations section, the code  
 111694 generated by *yacc* can be strictly type checked and contain arbitrary information. In  
 111695 addition, the lexical analyzer can provide the same kinds of values for tokens, if  
 111696 desired.

111697 An action is an arbitrary C statement and as such can do input or output, call  
 111698 subprograms, and alter external variables. An action is one or more C statements  
 111699 enclosed in curly braces ' { ' and ' } '.

111700 Certain pseudo-variables can be used in the action. These are macros for access to  
 111701 data structures known internally to *yacc*.

111702 \$\$ The value of the action can be set by assigning it to \$\$ . If type  
 111703 checking is enabled and the type of the value to be assigned cannot  
 111704 be determined, a diagnostic message may be generated.

111705 \$number This refers to the value returned by the component specified by the  
 111706 token *number* in the right side of a rule, reading from left to right;  
 111707 *number* can be zero or negative. If *number* is zero or negative, it refers  
 111708 to the data associated with the name on the parser's stack preceding  
 111709 the leftmost symbol of the current rule. (That is, "\$0" refers to the  
 111710 name immediately preceding the leftmost name in the current rule to  
 111711 be found on the parser's stack and "\$-1" refers to the symbol to *its*  
 111712 left.) If *number* refers to an element past the current point in the rule,  
 111713 or beyond the bottom of the stack, the result is undefined. If type

|        |                   |                                                                                                                  |
|--------|-------------------|------------------------------------------------------------------------------------------------------------------|
| 111714 |                   | checking is enabled and the type of the value to be assigned cannot                                              |
| 111715 |                   | be determined, a diagnostic message may be generated.                                                            |
| 111716 | $\$<tag>number$   |                                                                                                                  |
| 111717 |                   | These correspond exactly to the corresponding symbols without the                                                |
| 111718 |                   | <i>tag</i> inclusion, but allow for strict type checking (and preclude                                           |
| 111719 |                   | unwanted type conversions). The effect is that the macro is expanded                                             |
| 111720 |                   | to use <i>tag</i> to select an element from the YYSTYPE union (using                                             |
| 111721 |                   | <i>dataname.tag</i> ). This is particularly useful if <i>number</i> is not positive.                             |
| 111722 | $\$<tag>\$$       |                                                                                                                  |
| 111723 |                   | This imposes on the reference the type of the union member                                                       |
| 111724 |                   | referenced by <i>tag</i> . This construction is applicable when a reference to                                   |
| 111725 |                   | a left context value occurs in the grammar, and provides <i>yacc</i> with a                                      |
|        |                   | means for selecting a type.                                                                                      |
| 111726 |                   | Actions can occur anywhere in a rule (not just at the end); an action can access                                 |
| 111727 |                   | values returned by actions to its left, and in turn the value it returns can be                                  |
| 111728 |                   | accessed by actions to its right. An action appearing in the middle of a rule shall be                           |
| 111729 |                   | equivalent to replacing the action with a new non-terminal symbol and adding an                                  |
| 111730 |                   | empty rule with that non-terminal symbol on the left-hand side. The semantic                                     |
| 111731 |                   | action associated with the new rule shall be equivalent to the original action. The                              |
| 111732 |                   | use of actions within rules might introduce conflicts that would not otherwise                                   |
| 111733 |                   | exist.                                                                                                           |
| 111734 |                   | By default, the value of a rule shall be the value of the first element in it. If the first                      |
| 111735 |                   | element does not have a type (particularly in the case of a literal) and type                                    |
| 111736 |                   | checking is turned on by <b>%type</b> , an error message shall result.                                           |
| 111737 | <i>precedence</i> |                                                                                                                  |
| 111738 |                   | The keyword <b>%prec</b> can be used to change the precedence level associated with a                            |
| 111739 |                   | particular grammar rule. Examples of this are in cases where a unary and binary                                  |
| 111740 |                   | operator have the same symbolic representation, but need to be given different                                   |
| 111741 |                   | precedences, or where the handling of an ambiguous if-else construction is                                       |
| 111742 |                   | necessary. The reserved symbol <b>%prec</b> can appear immediately after the body of                             |
| 111743 |                   | the grammar rule and can be followed by a token name or a literal. It shall cause                                |
| 111744 |                   | the precedence of the grammar rule to become that of the following token name or                                 |
|        |                   | literal. The action for the rule as a whole can follow <b>%prec</b> .                                            |
| 111745 |                   | If a program section follows, the application shall ensure that the grammar rules are terminated                 |
| 111746 |                   | by <b>%%</b> .                                                                                                   |
| 111747 |                   | <b>Programs Section</b>                                                                                          |
| 111748 |                   | The <i>programs</i> section can include the definition of the lexical analyzer <i>yylex()</i> , and any other    |
| 111749 |                   | functions; for example, those used in the actions specified in the grammar rules. It is unspecified              |
| 111750 |                   | whether the programs section precedes or follows the semantic actions in the output file;                        |
| 111751 |                   | therefore, if the application contains any macro definitions and declarations intended to apply to               |
| 111752 |                   | the code in the semantic actions, it shall place them within "%{ . . . %}" in the declarations                   |
| 111753 |                   | section.                                                                                                         |
| 111754 |                   | <b>Input Grammar</b>                                                                                             |
| 111755 |                   | The following input to <i>yacc</i> yields a parser for the input to <i>yacc</i> . This formal syntax takes       |
| 111756 |                   | precedence over the preceding text syntax description.                                                           |
| 111757 |                   | The lexical structure is defined less precisely; <a href="#">Lexical Structure of the Grammar</a> (on page 3293) |
| 111758 |                   | defines most terms. The correspondence between the previous terms and the tokens below is as                     |
| 111759 |                   | follows.                                                                                                         |



```

111760 IDENTIFIER This corresponds to the concept of name, given previously. It also includes
111761 literals as defined previously.

111762 C_IDENTIFIER This is a name, and additionally it is known to be followed by a colon. A
111763 literal cannot yield this token.

111764 NUMBER A string of digits (a non-negative decimal integer).

111765 TYPE, LEFT, MARK, LCURL, RCURL
111766 These correspond directly to %type, %left, %%, %{, and %}.

111767 {...} This indicates C-language source code, with the possible inclusion of '$'
111768 macros as discussed previously.

111769 /* Grammar for the input to yacc. */
111770 /* Basic entries. */
111771 /* The following are recognized by the lexical analyzer. */

111772 %token IDENTIFIER /* Includes identifiers and literals */
111773 %token C_IDENTIFIER /* identifier (but not literal)
111774 followed by a :. */
111775 %token NUMBER /* [0-9][0-9]* */

111776 /* Reserved words : %type=>TYPE %left=>LEFT, and so on */

111777 %token LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

111778 %token MARK /* The %% mark. */
111779 %token LCURL /* The %{ mark. */
111780 %token RCURL /* The %} mark. */

111781 /* 8-bit character literals stand for themselves; */
111782 /* tokens have to be defined for multi-byte characters. */

111783 %start spec

111784 %%

111785 spec : defs MARK rules tail
111786 ;

111787 tail : MARK
111788 {
111789 /* In this action, set up the rest of the file. */
111790 }
111791 | /* Empty; the second MARK is optional. */
111792 ;

111793 defs : /* Empty. */
111794 | defs def
111795 ;

111796 def : START IDENTIFIER
111797 | UNION
111798 {
111799 /* Copy union definition to output. */
111800 }
111801 | LCURL
111802 {
111803 /* Copy C code to output file. */
111804 }
111805 | RCURL
111806 | rword tag nlist
111807 ;

```

```

111808 rword : TOKEN
111809 | LEFT
111810 | RIGHT
111811 | NONASSOC
111812 | TYPE
111813 ;
111814 tag : /* Empty: union tag ID optional. */
111815 | '<' IDENTIFIER '>'
111816 ;
111817 nlist : nmno
111818 | nlist nmno
111819 ;
111820 nmno : IDENTIFIER /* Note: literal invalid with % type. */
111821 | IDENTIFIER NUMBER /* Note: invalid with % type. */
111822 ;

111823 /* Rule section */
111824 rules : C_IDENTIFIER rbody prec
111825 | rules rule
111826 ;
111827 rule : C_IDENTIFIER rbody prec
111828 | '|' rbody prec
111829 ;
111830 rbody : /* empty */
111831 | rbody IDENTIFIER
111832 | rbody act
111833 ;
111834 act : '{'
111835 | {
111836 /* Copy action, translate $$, and so on. */
111837 }
111838 | '}'
111839 ;
111840 prec : /* Empty */
111841 | PREC IDENTIFIER
111842 | PREC IDENTIFIER act
111843 | prec ';'
111844 ;

```

## 111845 Conflicts

111846 The parser produced for an input grammar may contain states in which conflicts occur. The  
 111847 conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at  
 111848 least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or  
 111849 user-specified precedence rules.

111850 Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is  
 111851 where, for a given state and lookahead symbol, both a shift action and a reduce action are  
 111852 possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions  
 111853 by two different rules are possible.

111854 The rules below describe how to specify what actions to take when a conflict occurs. Not all  
 111855 shift/reduce conflicts can be successfully resolved this way because the conflict may be due to  
 111856 something other than ambiguity, so incautious use of these facilities can cause the language  
 111857 accepted by the parser to be much different from that which was intended. The description file

111858 shall contain sufficient information to understand the cause of the conflict. Where ambiguity is  
111859 the reason either the default or explicit rules should be adequate to produce a working parser.

111860 The declared precedences and associativities (see [Declarations Section](#), on page 3293) are used to  
111861 resolve parsing conflicts as follows:

- 111862 1. A precedence and associativity is associated with each grammar rule; it is the precedence  
111863 and associativity of the last token or literal in the body of the rule. If the `%prec` keyword  
111864 is used, it overrides this default. Some grammar rules might not have both precedence  
111865 and associativity.
- 111866 2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have  
111867 precedence and associativity associated with them, then the conflict is resolved in favor of  
111868 the action (shift or reduce) associated with the higher precedence. If the precedences are  
111869 the same, then the associativity is used; left associative implies reduce, right associative  
111870 implies shift, and non-associative implies an error in the string being parsed.
- 111871 3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done.  
111872 Conflicts resolved this way are counted in the diagnostic output described in [Error](#)  
111873 [Handling](#).
- 111874 4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that  
111875 occurs earlier in the input sequence. Conflicts resolved this way are counted in the  
111876 diagnostic output described in [Error Handling](#).

111877 Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and  
111878 reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

### 111879 **Error Handling**

111880 The token **error** shall be reserved for error handling. The name **error** can be used in grammar  
111881 rules. It indicates places where the parser can recover from a syntax error. The default value of  
111882 **error** shall be 256. Its value can be changed using a `%token` declaration. The lexical analyzer  
111883 should not return the value of **error**.

111884 The parser shall detect a syntax error when it is in a state where the action associated with the  
111885 lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by  
111886 executing the macro `YYERROR`. When `YYERROR` is executed, the semantic action passes control  
111887 back to the parser. `YYERROR` cannot be used outside of semantic actions.

111888 When the parser detects a syntax error, it normally calls `yyerror()` with the character string  
111889 "syntax error" as its argument. The call shall not be made if the parser is still recovering  
111890 from a previous error when the error is detected. The parser is considered to be recovering from  
111891 a previous error until the parser has shifted over at least three normal input symbols since the  
111892 last error was detected or a semantic action has executed the macro `yyerrok`. The parser shall not  
111893 call `yyerror()` when `YYERROR` is executed.

111894 The macro function `YYRECOVERING` shall return 1 if a syntax error has been detected and the  
111895 parser has not yet fully recovered from it. Otherwise, zero shall be returned.

111896 When a syntax error is detected by the parser, the parser shall check if a previous syntax error  
111897 has been detected. If a previous error was detected, and if no normal input symbols have been  
111898 shifted since the preceding error was detected, the parser checks if the lookahead symbol is an  
111899 endmarker (see [Interface to the Lexical Analyzer](#), on page 3300). If it is, the parser shall return  
111900 with a non-zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing  
111901 shall resume.

111902 When `YYERROR` is executed or when the parser detects a syntax error and no previous error has  
111903 been detected, or at least one normal input symbol has been shifted since the previous error was  
111904 detected, the parser shall pop back one state at a time until the parse stack is empty or the

111905 current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a  
 111906 non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser  
 111907 reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead  
 111908 symbol when parsing is resumed.

111909 The macro *yyerror* in a semantic action shall cause the parser to act as if it has fully recovered  
 111910 from any previous errors. The macro *yyclearin* shall cause the parser to discard the current  
 111911 lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no  
 111912 effect.

111913 The macro YYACCEPT shall cause the parser to return with the value zero. The macro  
 111914 YYABORT shall cause the parser to return with a non-zero value.

### 111915 Interface to the Lexical Analyzer

111916 The *yylex()* function is an integer-valued function that returns a *token number* representing the  
 111917 kind of token read. If there is a value associated with the token returned by *yylex()* (see the  
 111918 discussion of *tag* above), it shall be assigned to the external variable *yyval*.

111919 If the parser and *yylex()* do not agree on these token numbers, reliable communication between  
 111920 them cannot occur. For (single-byte character) literals, the token is simply the numeric value of  
 111921 the character in the current character set. The numbers for other tokens can either be chosen by  
 111922 *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex()*  
 111923 to return these numbers symbolically. The **#define** statements are put into the code file, and the  
 111924 header file if that file is requested. The set of characters permitted by *yacc* in an identifier is  
 111925 larger than that permitted by C. Token names found to contain such characters shall not be  
 111926 included in the **#define** declarations.

111927 If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers  
 111928 greater than 256, although no order is implied. A token can be explicitly assigned a number by  
 111929 following its first appearance in the declarations section with a number. Names and literals not  
 111930 defined this way retain their default definition. All token numbers assigned by *yacc* shall be  
 111931 unique and distinct from the token numbers used for literals and user-assigned tokens. If  
 111932 duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error;  
 111933 otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

111934 The end of the input is marked by a special token called the *endmarker*, which has a token  
 111935 number that is zero or negative. (These values are invalid for any other token.) All lexical  
 111936 analyzers shall return zero or negative as a token number upon reaching the end of their input.  
 111937 If the tokens up to, but excluding, the endmarker form a structure that matches the start symbol,  
 111938 the parser shall accept the input. If the endmarker is seen in any other context, it shall be  
 111939 considered an error.

### 111940 Completing the Program

111941 In addition to *yparse()* and *yylex()*, the functions *yyerror()* and *main()* are required to make a  
 111942 complete program. The application can supply *main()* and *yyerror()*, or those routines can be  
 111943 obtained from the *yacc* library.

### 111944 Yacc Library

111945 The following functions shall appear only in the *yacc* library accessible through the **-l y** operand  
 111946 to *c99*; they can therefore be redefined by a conforming application:

#### 111947 **int main(void)**

111948 This function shall call *yparse()* and exit with an unspecified value. Other actions within  
 111949 this function are unspecified.

111950 **int yyerror(const char \*s)**

111951 This function shall write the NUL-terminated argument to standard error, followed by a  
111952 <newline>.

111953 The order of the **-ly** and **-ll** operands given to *c99* is significant; the application shall either  
111954 provide its own *main()* function or ensure that **-ly** precedes **-ll**.

### 111955 **Debugging the Parser**

111956 The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled  
111957 at either compile time or at runtime (if enabled at compile time). The compilation of the runtime  
111958 debugging code is under the control of *YYDEBUG*, a preprocessor symbol. If *YYDEBUG* has a  
111959 non-zero value, the debugging code shall be included. If its value is zero, the code shall not be  
111960 included.

111961 In parsers where the debugging code has been included, the external **int yydebug** can be used to  
111962 turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of  
111963 *yydebug* shall be zero.

111964 When **-t** is specified, the code file shall be built such that, if *YYDEBUG* is not already defined at  
111965 compilation time (using the *c99* **-D YYDEBUG** option, for example), *YYDEBUG* shall be set  
111966 explicitly to 1. When **-t** is not specified, the code file shall be built such that, if *YYDEBUG* is not  
111967 already defined, it shall be set explicitly to zero.

111968 The format of the debugging output is unspecified but includes at least enough information to  
111969 determine the shift and reduce actions, and the input symbols. It also provides information  
111970 about error recovery.

### 111971 **Algorithms**

111972 The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the  
111973 literature. It is unspecified whether the parser is table-driven or direct-coded.

111974 A parser generated by *yacc* shall never request an input symbol from *yylex()* while in a state  
111975 where the only actions other than the error action are reductions by a single rule.

111976 The literature of parsing theory defines these concepts.

### 111977 **Limits**

111978 The *yacc* utility may have several internal tables. The minimum maximums for these tables are  
111979 shown in the following table. The exact meaning of these values is implementation-defined. The  
111980 implementation shall define the relationship between these values and between them and any  
111981 error messages that the implementation may generate should it run out of space for any internal  
111982 structure. An implementation may combine groups of these resources into a single pool as long  
111983 as the total available to the user does not fall below the sum of the sizes specified by this section.

Table 4-22 Internal Limits in *yacc*

| Limit      | Minimum<br>Maximum | Description                                                                                                                                                                                                                                                        |
|------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {NTERMS}   | 126                | Number of tokens.                                                                                                                                                                                                                                                  |
| {NNONTERM} | 200                | Number of non-terminals.                                                                                                                                                                                                                                           |
| {NPROD}    | 300                | Number of rules.                                                                                                                                                                                                                                                   |
| {NSTATES}  | 600                | Number of states.                                                                                                                                                                                                                                                  |
| {MEMSIZE}  | 5 200              | Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in <a href="#">Grammar Rules in yacc</a> (on page 3295). |
| {ACTSIZE}  | 4 000              | Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in <a href="#">Grammar Rules in yacc</a> (on page 3295).                                                        |

**EXIT STATUS**

The following exit values shall be returned:

- 0 Successful completion.
- >0 An error occurred.

**CONSEQUENCES OF ERRORS**

If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial code files and header files may be produced. The summary information in the description file shall always be produced if the `-v` flag is present.

**APPLICATION USAGE**

Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`, `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of *yacc* is running in a single directory at one time. The `-b` option was added to overcome this problem. The related problem of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a `-p` option to override the previously hard-coded `yy` variable prefix.

The description of the `-p` option specifies the minimal set of function and variable names that cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed. Instead, the programmer can use `-b` to give the header files for different parsers different names, and then the file with the `yylex()` for a given parser can include the header for that parser. Names such as `yyclearerr` do not need to be changed because they are used only in the actions; they do not have linkage. It is possible that an implementation has other names, either internal ones for implementing things such as `yyclearerr`, or providing non-standard features that it wants to change with `-p`.

Unary operators that are the same token as a binary operator in general need their precedence adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar rule defining that unary operator. (See [Grammar Rules in yacc](#) (on page 3295).) Applications are not required to use this operator for unary operators, but the grammars that do not require it are rare.

**EXAMPLES**

112029  
112030 Access to the *yacc* library is obtained with library search operands to *c99*. To use the *yacc* library  
112031 *main()*:

112032 `c99 y.tab.c -l y`

112033 Both the *lex* library and the *yacc* library contain *main()*. To access the *yacc main()*:

112034 `c99 y.tab.c lex.yy.c -l y -l l`

112035 This ensures that the *yacc* library is searched first, so that its *main()* is used.

112036 The historical *yacc* libraries have contained two simple functions that are normally coded by the  
112037 application programmer. These functions are similar to the following code:

```
112038 #include <locale.h>
112039 int main(void)
112040 {
112041 extern int yyparse();
112042 setlocale(LC_ALL, "");
112043 /* If the following parser is one created by lex, the
112044 application must be careful to ensure that LC_CTYPE
112045 and LC_COLLATE are set to the POSIX locale. */
112046 (void) yyparse();
112047 return (0);
112048 }
112049 #include <stdio.h>
112050 int yyerror(const char *msg)
112051 {
112052 (void) fprintf(stderr, "%s\n", msg);
112053 return (0);
112054 }
```

**RATIONALE**

112055 The references in [Referenced Documents](#) (on page 0) may be helpful in constructing the parser  
112056 generator. The referenced DeRemer and Pennello article (along with the works it references)  
112057 describes a technique to generate parsers that conform to this volume of POSIX.1-200x. Work in  
112058 this area continues to be done, so implementors should consult current literature before doing  
112059 any new implementations. The original Knuth article is the theoretical basis for this kind of  
112060 parser, but the tables it generates are impractically large for reasonable grammars and should  
112061 not be used. The “equivalent to” wording is intentional to assure that the best tables that are  
112062 LALR(1) can be generated.  
112063

112064 There has been confusion between the class of grammars, the algorithms needed to generate  
112065 parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal.  
112066 In particular, a parser generator that accepts the full range of LR(1) grammars need not generate  
112067 a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars)  
112068 for a grammar that happens to be SLR(1). Such an implementation need not recognize the case,  
112069 either; table compression can yield the SLR(1) table (or one even smaller than that) without  
112070 recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent  
112071 more upon the table representation and compression (or the code generation if a direct parser is  
112072 generated) than upon the class of grammar that the table generator handles.

112073 The speed of the parser generator is somewhat dependent upon the class of grammar it handles.  
112074 However, the original Knuth article algorithms for constructing LR parsers were judged by its  
112075 author to be impractically slow at that time. Although full LR is more complex than LALR(1), as  
112076 computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock

112077 execution time) is becoming less significant.

112078 Potential authors are cautioned that the referenced DeRemer and Pennello article previously  
 112079 cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in  
 112080 some of the LALR(1) algorithm statements that preceded it to publication. They should take the  
 112081 time to seek out that paper, as well as current relevant work, particularly Aho's.

112082 The **-b** option was added to provide a portable method for permitting *yacc* to work on multiple  
 112083 separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and  
 112084 both grammars are constructed at the same time (by, for example, a parallel *make* program),  
 112085 conflict results. While the solution is not historical practice, it corrects a known deficiency in  
 112086 historical implementations. Corresponding changes were made to all sections that referenced  
 112087 the filenames **y.tab.c** (now "the code file"), **y.tab.h** (now "the header file"), and **y.output** (now  
 112088 "the description file").

112089 The grammar for *yacc* input is based on System V documentation. The textual description shows  
 112090 there that the `' ; '` is required at the end of the rule. The grammar and the implementation do  
 112091 not require this. (The use of **C\_IDENTIFIER** causes a reduce to occur in the right place.)

112092 Also, in that implementation, the constructs such as **%token** can be terminated by a semicolon,  
 112093 but this is not permitted by the grammar. The keywords such as **%token** can also appear in  
 112094 uppercase, which is again not discussed. In most places where `' % '` is used, `' \ '` can be  
 112095 substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can  
 112096 be `" % < "` or even `" \ < "`).

112097 Historically, `<tag>` can contain any characters except `' > '`, including white space, in the  
 112098 implementation. However, since the *tag* must reference an ISO C standard union member, in  
 112099 practice conforming implementations need to support only the set of characters for ISO C  
 112100 standard identifiers in this context.

112101 Some historical implementations are known to accept actions that are terminated by a period.  
 112102 Historical implementations often allow `' $ '` in names. A conforming implementation does not  
 112103 need to support either of these behaviors.

112104 Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There  
 112105 may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot  
 112106 interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances  
 112107 be resolved by providing additional information, such as using **%type** or **%union** declarations.  
 112108 It is often easier and it usually yields a smaller parser to take this alternative when it is  
 112109 appropriate.

112110 The size and execution time of a program produced without the runtime debugging code is  
 112111 usually smaller and slightly faster in historical implementations.

112112 Statistics messages from several historical implementations include the following types of  
 112113 information:

112114 `n/512 terminals, n/300 non-terminals`  
 112115 `n/600 grammar rules, n/1500 states`  
 112116 `n shift/reduce, n reduce/reduce conflicts reported`  
 112117 `n/350 working sets used`  
 112118 `Memory: states, etc. n/15000, parser n/15000`  
 112119 `n/600 distinct lookahead sets`  
 112120 `n extra closures`  
 112121 `n shift entries, n exceptions`  
 112122 `n goto entries`  
 112123 `n entries saved by goto default`  
 112124 `Optimizer space used: input n/15000, output n/15000`  
 112125 `n table entries, n zero`



112126 Maximum spread: *n*, Maximum offset: *n*

112127 The report of internal tables in the description file is left implementation-defined because all  
112128 aspects of these limits are also implementation-defined. Some implementations may use  
112129 dynamic allocation techniques and have no specific limit values to report.

112130 The format of the **y.output** file is not given because specification of the format was not seen to  
112131 enhance applications portability. The listing is primarily intended to help human users  
112132 understand and debug the parser; use of **y.output** by a conforming application script would be  
112133 unusual. Furthermore, implementations have not produced consistent output and no popular  
112134 format was apparent. The format selected by the implementation should be human-readable, in  
112135 addition to the requirement that it be a text file.

112136 Standard error reports are not specifically described because they are seldom of use to  
112137 conforming applications and there was no reason to restrict implementations.

112138 Some implementations recognize "`={"`" as equivalent to '`{`' because it appears in historical  
112139 documentation. This construction was recognized and documented as obsolete as long ago as  
112140 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of POSIX.1-200x chose to  
112141 leave it as obsolete and omit it.

112142 Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They  
112143 should not be returned as multi-byte character literals. The token **error** that is used for error  
112144 recovery is normally assigned the value 256 in the historical implementation. Thus, the token  
112145 value 256, which is used in many multi-byte character sets, is not available for use as the value  
112146 of a user-defined token.

#### 112147 FUTURE DIRECTIONS

112148 None.

#### 112149 SEE ALSO

112150 [c99](#), [lex](#)

112151 [XBD Chapter 8](#) (on page 159), [Section 12.2](#) (on page 201)

#### 112152 CHANGE HISTORY

112153 First released in Issue 2.

##### 112154 Issue 5

112155 The FUTURE DIRECTIONS section is added.

##### 112156 Issue 6

112157 This utility is marked as part of the C-Language Development Utilities option.

112158 Minor changes have been added to align with the IEEE P1003.2b draft standard.

112159 The normative text is reworded to avoid use of the term "must" for application requirements.

112160 IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the `}%`  
112161 token to the `%}`.

##### 112162 Issue 7

112163 SD5-XCU-ERN-6 is applied, clarifying that Guideline 9 of the Utility Syntax Guidelines does not  
112164 apply.

112165 SD5-XCU-ERN-97 is applied, updating the SYNOPSIS.

112166 **NAME**

112167 zcat — expand and concatenate data

112168 **SYNOPSIS**112169 XSI zcat [*file...*]112170 **DESCRIPTION**

112171 The *zcat* utility shall write to standard output the uncompressed form of files that have been  
 112172 compressed using the *compress* utility. It is the equivalent of *uncompress -c*. Input files are not  
 112173 affected.

112174 **OPTIONS**

112175 None.

112176 **OPERANDS**

112177 The following operand shall be supported:

112178 *file* The pathname of a file previously processed by the *compress* utility. If *file* already  
 112179 has the *.Z* suffix specified, it is used as submitted. Otherwise, the *.Z* suffix is  
 112180 appended to the filename prior to processing.

112181 **STDIN**112182 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is *'-'*.112183 **INPUT FILES**112184 Input files shall be compressed files that are in the format produced by the *compress* utility.112185 **ENVIRONMENT VARIABLES**112186 The following environment variables shall affect the execution of *zcat*:

112187 *LANG* Provide a default value for the internationalization variables that are unset or null. |  
 112188 (See XBD Section 8.2 (on page 160) for the precedence of internationalization |  
 112189 variables used to determine the values of locale categories.)

112190 *LC\_ALL* If set to a non-empty string value, override the values of all the other  
 112191 internationalization variables.

112192 *LC\_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as  
 112193 characters (for example, single-byte as opposed to multi-byte characters in  
 112194 arguments).

112195 *LC\_MESSAGES*

112196 Determine the locale that should be used to affect the format and contents of  
 112197 diagnostic messages written to standard error.

112198 *NLSPATH* Determine the location of message catalogs for the processing of *LC\_MESSAGES*.

112199 **ASYNCHRONOUS EVENTS**

112200 Default.

112201 **STDOUT**

112202 The compressed files given as input shall be written on standard output in their uncompressed  
 112203 form.

112204 **STDERR**

112205 The standard error shall be used only for diagnostic messages.

112206 **OUTPUT FILES**

112207 None.

112208 **EXTENDED DESCRIPTION**

112209 None.

112210 **EXIT STATUS**

112211 The following exit values shall be returned:

112212 0 Successful completion.

112213 &gt;0 An error occurred.

112214 **CONSEQUENCES OF ERRORS**

112215 Default.

112216 **APPLICATION USAGE**

112217 None.

112218 **EXAMPLES**

112219 None.

112220 **RATIONALE**

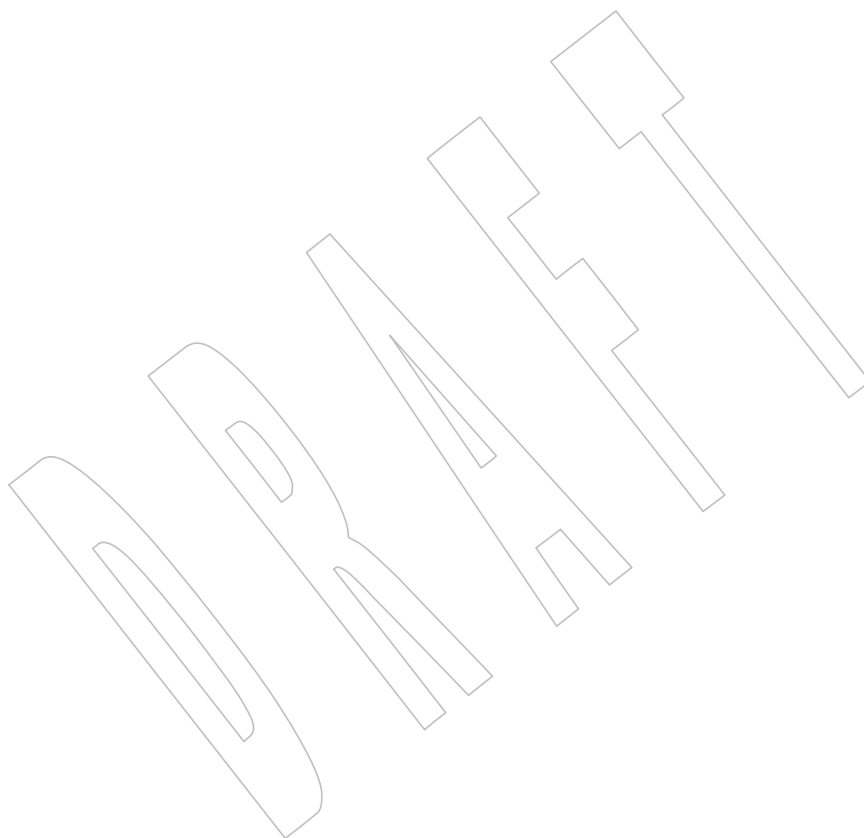
112221 None.

112222 **FUTURE DIRECTIONS**

112223 None.

112224 **SEE ALSO**112225 *compress, uncompress*112226 XBD [Chapter 8](#) (on page 159)112227 **CHANGE HISTORY**

112228 First released in Issue 4.



112229

**Technical Standard**

112230

**Volume 4:**

112231

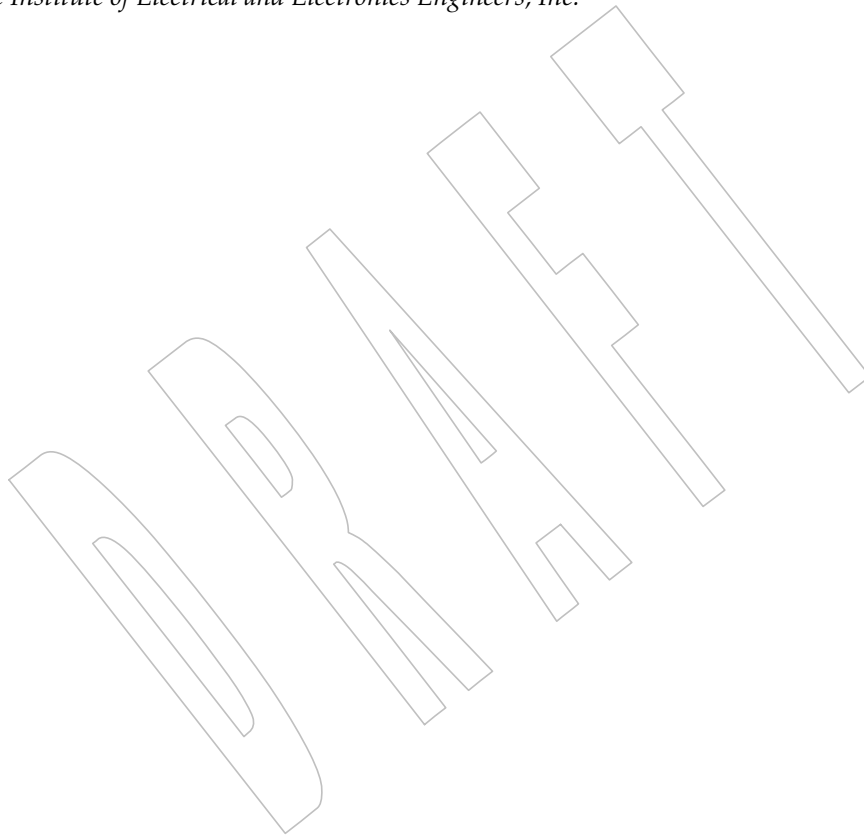
**Rationale (Informative), Issue 7**

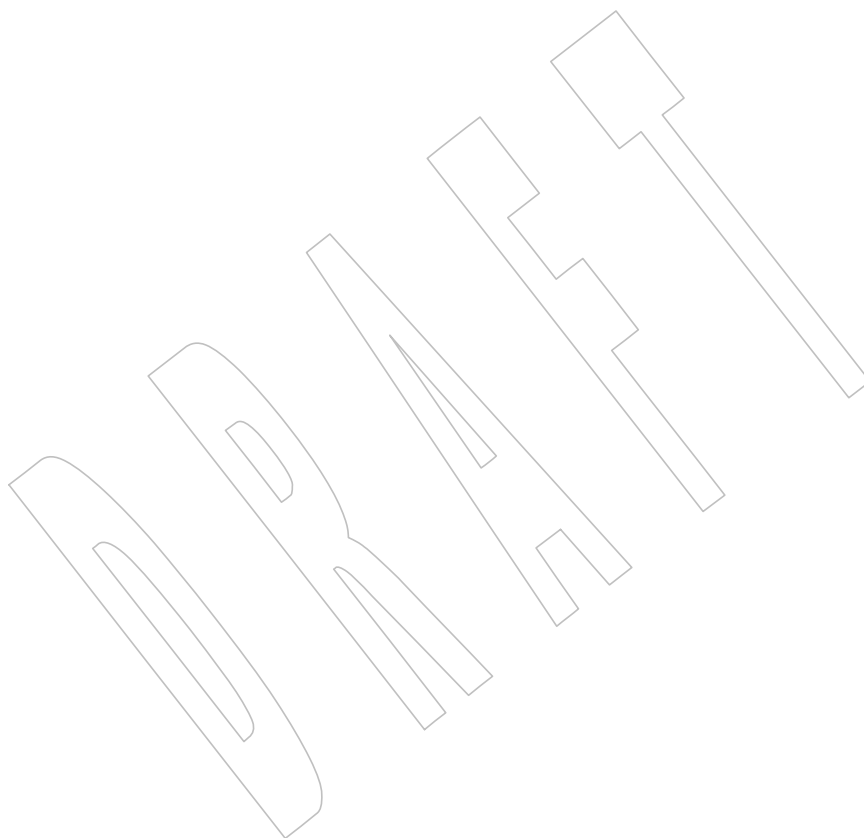
112232

*The Open Group*

112233

*The Institute of Electrical and Electronics Engineers, Inc.*





112234

# *Rationale (Informative)*

112235

## **Part A:**

112236

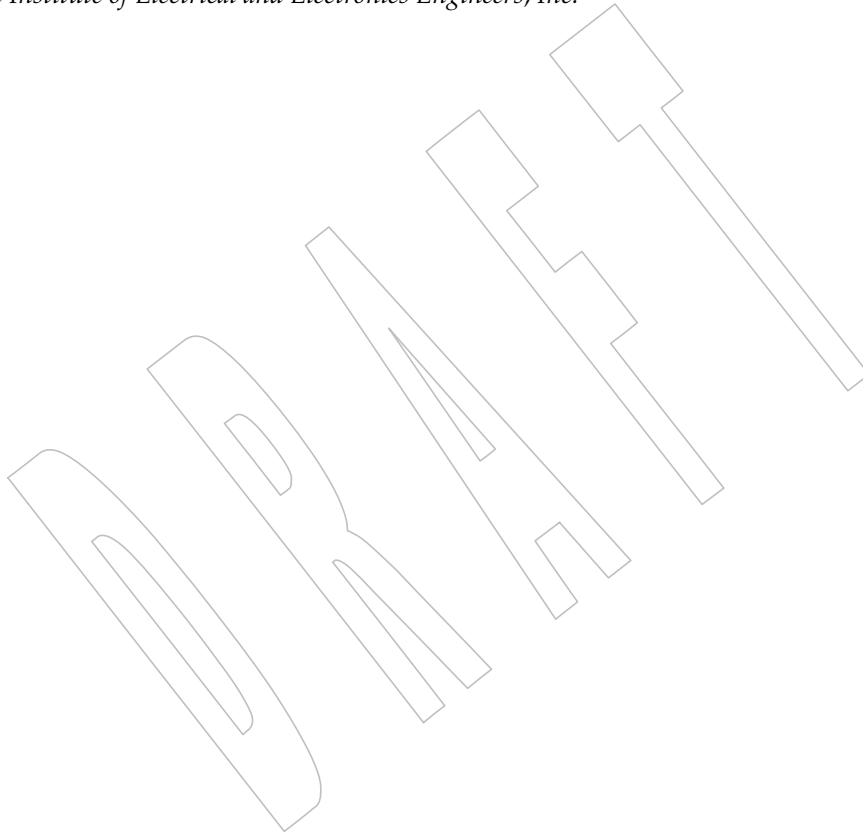
## **Base Definitions**

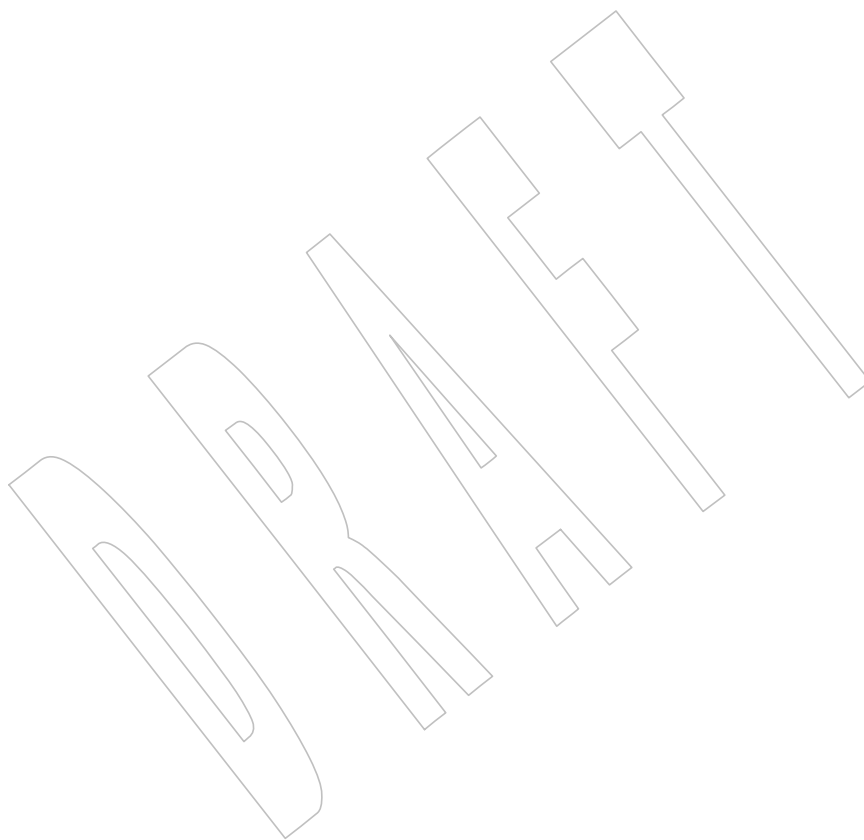
112237

*The Open Group*

112238

*The Institute of Electrical and Electronics Engineers, Inc.*







*Rationale for Base Definitions***A.1 Introduction****A.1.1 Scope**

POSIX.1-200x is one of a family of standards known as POSIX. The family of standards extends to many topics; POSIX.1-200x is known as POSIX.1 and consists of both operating system interfaces and shell and utilities. POSIX.1-200x is technically identical to The Open Group Base Specifications, Issue 7.

**Scope of POSIX.1-200x**

The (paraphrased) goals of this development were to revise the single document that is ISO/IEC 9945:2003 Parts 1 through 4, IEEE Std 1003.1, 2004 Edition, and the appropriate parts of The Open Group Single UNIX Specification, Version 3. This work has been undertaken by the Austin Group, a joint working group of IEEE, The Open Group, and ISO/IEC JTC 1/SC 22.

The following are the base documents in this version:

- IEEE Std 1003.1, 2004 Edition
- ISO/IEC 9899:1999, Programming Languages — C (including ISO/IEC 9899:1999/Cor.1:2001(E) and ISO/IEC 9899:1999/Cor.2:2004(E))
- The Open Group Extended API Sets, Parts 1 through 4

This version has addressed the following areas:

- Issues raised by Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and ISO/IEC defect reports against ISO/IEC 9945

The repository of interpretations can be accessed at [www.opengroup.org/austin/interps](http://www.opengroup.org/austin/interps).

- Issues raised in corrigenda for The Open Group Technical Standards and working group resolutions from The Open Group
- Issues arising from ISO TR 24715: 2006, Conflicts between POSIX and the LSB

This is a Type 3 informative technical report highlighting differences between the LSB 3.1 and the 2004 Edition of this standard.

- Changes to make the text self-consistent with the additional material merged

The new material merged has come from the The Open Group Extended API Sets Parts 1 through 4. A list of the new interfaces is included in [Section B.1.1](#) (on page 3393).

- Features, marked legacy or obsolescent in the base documents, have been considered for removal in this version

See [Section B.1.1](#) (on page 3393) and [Section C.1.1](#) (on page 3535).

- 112272 • A review and reorganization of the options within the standard

112273 This has included marking the following options obsolescent:

- 112274 — Batch Environment Services and Utilities
- 112275 — Tracing
- 112276 — XSI STREAMS

112277 The UUCP Utilities option is a new option for this version.

112278 Functionality from the following former options is now mandatory in this version:

|        |     |                                                        |
|--------|-----|--------------------------------------------------------|
| 112279 | AIO | _POSIX_ASYNCHRONOUS_IO (Asynchronous Input and Output) |
| 112280 | BAR | _POSIX_BARRIERS (Barriers)                             |
| 112281 | CS  | _POSIX_CLOCK_SELECTION (Clock Selection)               |
| 112282 | MF  | _POSIX_MAPPED_FILES (Memory Mapped Files)              |
| 112283 | MPR | _POSIX_MEMORY_PROTECTION (Memory Protection)           |
| 112284 | RTS | _POSIX_REALTIME_SIGNALS (Realtime Signals Extension)   |
| 112285 | RWL | _POSIX_READER_WRITER_LOCKS (Read-Write Locks)          |
| 112286 | SEM | _POSIX_SEMAPHORES (Semaphores)                         |
| 112287 | SPI | _POSIX_SPIN_LOCKS (Spin Locks)                         |
| 112288 | THR | _POSIX_THREADS (Threads)                               |
| 112289 | TMO | _POSIX_TIMEOUTS (Timeouts)                             |
| 112290 | TMR | _POSIX_TIMERS (Timers)                                 |
| 112291 | TSF | _POSIX_THREAD_SAFE_FUNCTIONS (Thread-Safe Functions)   |

- 112292 • Alignment with the ISO/IEC 9899:1999 standard, including Technical Corrigendum 2

- 112293 • A review of the use of fixed path filenames within the standard

112294 For example, the *at*, *batch*, and *crontab* utilities previously had a requirement for use of the  
112295 directory **/usr/lib/cron**.

112296 The following were requirements on POSIX.1-200x:

- 112297 • Backward-compatibility

112298 For interfaces carried forward, it was agreed that there should be no breakage of  
112299 functionality in the existing base documents. All strictly conforming applications will be  
112300 conforming but not necessarily strictly conforming to the revised standard. The goal is for  
112301 system implementations to be able to support the existing and revised standards  
112302 simultaneously.

- 112303 • Architecture and *n*-bit-neutral

112304 The common standard should not make any implicit assumptions about the system  
112305 architecture or size of data types; for example, previously some 32-bit implicit assumptions  
112306 had crept into the standards.

- 112307 • Extensibility

112308 It should be possible to extend the common standard without breaking backwards-  
112309 compatibility; for example, the name space should be reserved and structured to avoid  
112310 duplication of names between the standard and extensions to it.

112311 **POSIX.1 and the ISO C Standard**

112312 The standard developers believed it essential for a programmer to have a single complete  
 112313 reference place, but recognized that deference to the formal standard has to be addressed for the  
 112314 duplicate interface definitions between the ISO C standard and POSIX.1-200x.

112315 Where an interface has a version in the ISO C standard, the DESCRIPTION section describes the  
 112316 relationship to the ISO C standard and markings are included as appropriate to show where the  
 112317 ISO C standard has been extended in the text.

112318 A block of text is included at the start of each affected reference page stating whether the page is  
 112319 aligned with the ISO C standard or extended. Each page has been parsed for additions beyond  
 112320 the ISO C standard (that is, including both POSIX and UNIX extensions), and these extensions  
 112321 are marked as CX extensions (for C extensions).

112322 **FIPS Requirements**

112323 The Federal Information Processing Standards (FIPS) are a series of U.S. government  
 112324 procurement standards managed and maintained on behalf of the U.S. Department of  
 112325 Commerce by the National Institute of Standards and Technology (NIST).

112326 The following restrictions were integrated into IEEE Std 1003.1-2001. They originally came from  
 112327 FIPS 151-2 which was withdrawn by NIST on February 25 2000.

- 112328 • The implementation supports `_POSIX_CHOWN_RESTRICTED`.
- 112329 • The limit `{NGROUPS_MAX}` is greater than or equal to 8.
- 112330 • The implementation supports the setting of the group ID of a file (when it is created) to  
 112331 that of the parent directory.
- 112332 • The implementation supports `_POSIX_SAVED_IDS`.
- 112333 • The implementation supports `_POSIX_VDISABLE`.
- 112334 • The implementation supports `_POSIX_JOB_CONTROL`.
- 112335 • The implementation supports `_POSIX_NO_TRUNC`.
- 112336 • The `read()` function returns the number of bytes read when interrupted by a signal and  
 112337 does not return `-1`.
- 112338 • The `write()` function returns the number of bytes written when interrupted by a signal and  
 112339 does not return `-1`.
- 112340 • In the environment for the login shell, the environment variables `LOGNAME` and `HOME`  
 112341 are defined and have the properties described in POSIX.1-200x.
- 112342 • The value of `{CHILD_MAX}` is greater than or equal to 25.
- 112343 • The value of `{OPEN_MAX}` is greater than or equal to 20.
- 112344 • The implementation supports the functionality associated with the symbols `CS7`, `CS8`,  
 112345 `CSTOPB`, `PARODD`, and `PARENB` defined in `<termios.h>`.

112346 **A.1.2 Conformance**112347 See [Section A.2](#) (on page 3319).112348 **A.1.3 Normative References**

112349 There is no additional rationale provided for this section.

112350 **A.1.4 Change History**

112351 There is no additional rationale provided for this section. +

112352 **A.1.5 Terminology**112353 The meanings specified in POSIX.1-200x for the words *shall*, *should*, and *may* are mandated by  
112354 ISO/IEC directives.112355 In the Rationale (Informative) volume of POSIX.1-200x, the words *shall*, *should*, and *may* are  
112356 sometimes used to illustrate similar usages in POSIX.1-200x. However, the rationale itself does  
112357 not specify anything regarding implementations or applications.112358 **conformance document**112359 As a practical matter, the conformance document is effectively part of the system  
112360 documentation. Conformance documents are distinguished by POSIX.1-200x so that they  
112361 can be referred to distinctly.112362 **implementation-defined**112363 This definition is analogous to that of the ISO C standard and, together with “undefined”  
112364 and “unspecified”, provides a range of specification of freedom allowed to the interface  
112365 implementor.112366 **may**112367 The use of *may* has been limited as much as possible, due both to confusion stemming from  
112368 its ordinary English meaning and to objections regarding the desirability of having as few  
112369 options as possible and those as clearly specified as possible.112370 The usage of *can* and *may* were selected to contrast optional application behavior (can)  
112371 against optional implementation behavior (may).112372 **shall**112373 Declarative sentences are sometimes used in POSIX.1-200x as if they included the word  
112374 *shall*, and facilities thus specified are no less required. For example, the two statements:112375 1. The *foo()* function shall return zero.112376 2. The *foo()* function returns zero.

112377 are meant to be exactly equivalent.

112378 **should**112379 In POSIX.1-200x, the word *should* does not usually apply to the implementation, but rather  
112380 to the application. Thus, the important words regarding implementations are *shall*, which  
112381 indicates requirements, and *may*, which indicates options.112382 **obsolescent**112383 The term “obsolescent” means “do not use this feature in new applications”. A feature  
112384 noted as obsolescent is supported by all implementations, but may be removed in a future  
112385 version; new applications should not use these features. The obsolescence concept is not an  
112386 ideal solution, but was used as a method of increasing consensus: many more objections

112387 would be heard from the user community if some of these historical features were suddenly  
 112388 removed without the grace period obsolescence implies. The phrase “may be removed in a  
 112389 future version” implies that the result of that consideration might in fact keep those features  
 112390 indefinitely if the predominance of applications do not migrate away from them quickly.

#### 112391 **legacy**

112392 The term “legacy” was included in earlier versions of this standard but is no longer used in  
 112393 the current version.

#### 112394 **system documentation**

112395 The system documentation should normally describe the whole of the implementation,  
 112396 including any extensions provided by the implementation. Such documents normally  
 112397 contain information at least as detailed as the specifications in POSIX.1-200x. Few  
 112398 requirements are made on the system documentation, but the term is needed to avoid a  
 112399 dangling pointer where the conformance document is permitted to point to the system  
 112400 documentation.

#### 112401 **undefined**

112402 See *implementation-defined*.

#### 112403 **unspecified**

112404 See *implementation-defined*.

112405 The definitions for “unspecified” and “undefined” appear nearly identical at first  
 112406 examination, but are not. The term “unspecified” means that a conforming application may  
 112407 deal with the unspecified behavior, and it should not care what the outcome is. The term  
 112408 “undefined” says that a conforming application should not do it because no definition is  
 112409 provided for what it does (and implicitly it would care what the outcome was if it tried it).  
 112410 It is important to remember, however, that if the syntax permits the statement at all, it must  
 112411 have some outcome in a real implementation.

112412 Thus, the terms “undefined” and “unspecified” apply to the way the application should  
 112413 think about the feature. In terms of the implementation, it is always “defined”—there is  
 112414 always some result, even if it is an error. The implementation is free to choose the behavior  
 112415 it prefers.

112416 This also implies that an implementation, or another standard, could specify or define the  
 112417 result in a useful fashion. The terms apply to POSIX.1-200x specifically.

112418 The term “implementation-defined” implies requirements for documentation that are not  
 112419 required for “undefined” (or “unspecified”). Where there is no need for a conforming  
 112420 program to know the definition, the term “undefined” is used, even though  
 112421 “implementation-defined” could also have been used in this context. There could be a  
 112422 fourth term, specifying “this standard does not say what this does; it is acceptable to define  
 112423 it in an implementation, but it does not need to be documented”, and undefined would  
 112424 then be used very rarely for the few things for which any definition is not useful. In  
 112425 particular, implementation-defined is used where it is believed that certain classes of  
 112426 application will need to know such details to determine whether the application can be  
 112427 successfully ported to the implementation. Such applications are not always strictly  
 112428 portable, but nevertheless are common and useful; often the requirements met by the  
 112429 application cannot be met without dealing with the issues implied by “implementation-  
 112430 defined”. In some places the text refers to facilities supplied by the implementation that are  
 112431 outside the standard as implementation-supplied or implementation-provided. This is not  
 112432 intended to imply a requirement for documentation. If it were, the term “implementation-  
 112433 defined” would have been used.

112434 In many places POSIX.1-200x is silent about the behavior of some possible construct. For

112435 example, a variable may be defined for a specified range of values and behaviors are  
 112436 described for those values; nothing is said about what happens if the variable has any other  
 112437 value. That kind of silence can imply an error in the standard, but it may also imply that the  
 112438 standard was intentionally silent and that any behavior is permitted. There is a natural  
 112439 tendency to infer that if the standard is silent, a behavior is prohibited. That is not the intent.  
 112440 Silence is intended to be equivalent to the term “unspecified”.

112441 The term “application” is not defined in POSIX.1-200x; it is assumed to be a part of general  
 112442 computer science terminology.

112443 Three terms used within POSIX.1-200x overlap in meaning: “macro”, “symbolic name”, and  
 112444 “symbolic constant”.

#### 112445 **macro**

112446 This usually describes a C preprocessor symbol, the result of the **#define** operator, with or  
 112447 without an argument. It may also be used to describe similar mechanisms in editors and  
 112448 text processors.

#### 112449 **symbolic name**

112450 In earlier versions of this standard this was also sometimes used to refer to a C preprocessor  
 112451 symbol (without arguments), but the intention is for all such uses to have been removed. It  
 112452 is now mainly used to refer to the names for characters in character sets, but is sometimes  
 112453 used to refer to host names and even filenames.

#### 112454 **symbolic constant**

112455 This also refers to a C preprocessor symbol, with specific associated requirements. See the  
 112456 definition in [Section 3.372](#) (on page 84).

### 112457 **A.1.6 Definitions and Concepts**

112458 There is no additional rationale provided for this section.

### 112459 **A.1.7 Portability**

112460 To aid the identification of options within POSIX.1-200x, a notation consisting of margin codes  
 112461 and shading is used. This is based on the notation used in earlier versions of The Open Group  
 112462 Base specifications.

112463 The benefit of this approach is a reduction in the number of *if* statements within the running  
 112464 text, that makes the text easier to read, and also an identification to the programmer that they  
 112465 need to ensure that their target platforms support the underlying options. For example, if  
 112466 functionality is marked with THR in the margin, it will be available on all systems supporting  
 112467 the Threads option, but may not be available on some others.

#### 112468 **A.1.7.1 Codes**

112469 This section includes codes for options defined in XBD [Section 2.1.6](#) (on page 25), and the  
 112470 following additional codes for other purposes:

112471 **CX** This margin code is used to denote extensions beyond the ISO C standard. For  
 112472 interfaces that are duplicated between POSIX.1-200x and the ISO C standard, a CX  
 112473 introduction block describes the nature of the duplication, with any extensions  
 112474 appropriately CX marked and shaded.

112475 Where an interface is added to an ISO C standard header, within the header the  
 112476 interface has an appropriate margin marker and shading (for example, CX, XSI, TSE,  
 112477 and so on) and the same marking appears on the reference page in the SYNOPSIS  
 112478 section. This enables a programmer to easily identify that the interface is extending an

|        |    |                                                                                                                                                                                                                                                                                                                                 |
|--------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 112479 |    | ISO C standard header.                                                                                                                                                                                                                                                                                                          |
| 112480 | MX | This margin code is used to denote IEC 60559:1989 standard floating-point extensions.                                                                                                                                                                                                                                           |
| 112481 | OB | This margin code is used to denote obsolescent behavior and thus flag a possible future applications portability warning.                                                                                                                                                                                                       |
| 112482 |    |                                                                                                                                                                                                                                                                                                                                 |
| 112483 | OH | The Single UNIX Specification has historically tried to reduce the number of headers an application has had to include when using a particular interface. Sometimes this was fewer than the base standard, and hence a notation is used to flag which headers are optional if you are using a system supporting the XSI option. |
| 112484 |    |                                                                                                                                                                                                                                                                                                                                 |
| 112485 |    |                                                                                                                                                                                                                                                                                                                                 |
| 112486 |    |                                                                                                                                                                                                                                                                                                                                 |

#### 112487 A.1.7.2 Margin Code Notation

112488 Since some features may depend on one or more options, or require more than one option, a  
 112489 notation is used. Where a feature requires support of a single option, a single margin code will  
 112490 occur in the margin. If it depends on two options and both are required, then the codes will  
 112491 appear with a <space> separator. If either of two options are required, then a logical OR is  
 112492 denoted using the ' | ' symbol. If more than two codes are used, a special notation is used.

## 112493 A.2 Conformance

112494 The terms “profile” and “profiling” are used throughout this section.

112495 A profile of a standard or standards is a codified set of option selections, such that by being  
 112496 conformant to a profile, particular classes of users are specifically supported.

### 112497 A.2.1 Implementation Conformance

112498 These definitions allow application developers to know what to depend on in an  
 112499 implementation.

112500 There is no definition of a “strictly conforming implementation”; that would be an  
 112501 implementation that provides *only* those facilities specified by POSIX.1 with no extensions  
 112502 whatsoever. This is because no actual operating system implementation can exist without  
 112503 system administration and initialization facilities that are beyond the scope of POSIX.1.

#### 112504 A.2.1.1 Requirements

112505 The word “support” is used in certain instances, rather than “provide”, in order to allow an  
 112506 implementation that has no resident software development facilities, but that supports the  
 112507 execution of a *Strictly Conforming POSIX.1 Application*, to be a *conforming implementation*.

#### 112508 A.2.1.2 Documentation

112509 The conformance documentation is required to use the same numbering scheme as POSIX.1 for  
 112510 purposes of cross-referencing. All options that an implementation chooses are reflected in  
 112511 <limits.h> and <unistd.h>.

112512 Note that the use of “may” in terms of where conformance documents record where  
 112513 implementations may vary, implies that it is not required to describe those features identified as  
 112514 undefined or unspecified.

112515 Other aspects of systems must be evaluated by purchasers for suitability. Many systems  
 112516 incorporate buffering facilities, maintaining updated data in volatile storage and transferring  
 112517 such updates to non-volatile storage asynchronously. Various exception conditions, such as a  
 112518 power failure or a system crash, can cause this data to be lost. The data may be associated with a

112519 file that is still open, with one that has been closed, with a directory, or with any other internal  
 112520 system data structures associated with permanent storage. This data can be lost, in whole or  
 112521 part, so that only careful inspection of file contents could determine that an update did not  
 112522 occur.

112523 Also, interrelated file activities, where multiple files and/or directories are updated, or where  
 112524 space is allocated or released in the file system structures, can leave inconsistencies in the  
 112525 relationship between data in the various files and directories, or in the file system itself. Such  
 112526 inconsistencies can break applications that expect updates to occur in a specific sequence, so that  
 112527 updates in one place correspond with related updates in another place.

112528 For example, if a user creates a file, places information in the file, and then records this action in  
 112529 another file, a system or power failure at this point followed by restart may result in a state in  
 112530 which the record of the action is permanently recorded, but the file created (or some of its  
 112531 information) has been lost. The consequences of this to the user may be undesirable. For a user  
 112532 on such a system, the only safe action may be to require the system administrator to have a  
 112533 policy that requires, after any system or power failure, that the entire file system must be  
 112534 restored from the most recent backup copy (causing all intervening work to be lost).

112535 The characteristics of each implementation will vary in this respect and may or may not meet  
 112536 the requirements of a given application or user. Enforcement of such requirements is beyond the  
 112537 scope of POSIX.1. It is up to the purchaser to determine what facilities are provided in an  
 112538 implementation that affect the exposure to possible data or sequence loss, and also what  
 112539 underlying implementation techniques and/or facilities are provided that reduce or limit such  
 112540 loss or its consequences.

#### 112541 A.2.1.3 POSIX Conformance

112542 This really means conformance to the base standard; however, since this document includes the  
 112543 core material of the Single UNIX Specification, the standard developers decided that it was  
 112544 appropriate to segment the conformance requirements into two, the former for the base  
 112545 standard, and the latter for the Single UNIX Specification (denoted XSI Conformance).

112546 Within POSIX.1 there are some symbolic constants that, if defined to a certain value or range of  
 112547 values, indicate that a certain option is enabled. Other symbolic constants exist in POSIX.1 for  
 112548 other reasons.

112549 In this version, some features that were previously optional have been made mandatory. For  
 112550 backwards compatibility, the symbolic constants associated with the option are still required  
 112551 now with fixed allowable ranges or values. The following options from the previous version of  
 112552 this standard are now mandatory:

112553     \_POSIX\_ASYNCHRONOUS\_IO  
 112554     \_POSIX\_BARRIERS  
 112555     \_POSIX\_CLOCK\_SELECTION  
 112556     \_POSIX\_MAPPED\_FILES  
 112557     \_POSIX\_MEMORY\_PROTECTION  
 112558     \_POSIX\_READER\_WRITER\_LOCKS  
 112559     \_POSIX\_REALTIME\_SIGNALS  
 112560     \_POSIX\_SEMAPHORES  
 112561     \_POSIX\_SPIN\_LOCKS  
 112562     \_POSIX\_THREAD\_SAFE\_FUNCTIONS  
 112563     \_POSIX\_THREADS  
 112564     \_POSIX\_TIMEOUTS  
 112565     \_POSIX\_TIMERS



112566 A POSIX-conformant system may support the XSI option required by the Single UNIX  
 112567 Specification. This was intentional since the standard developers intend them to be upwards-  
 112568 compatible, so that a system conforming to the Single UNIX Specification can also conform to  
 112569 the base standard at the same time.

#### 112570 A.2.1.4 XSI Conformance

112571 This section is included to describe the conformance requirements for the base volumes of the  
 112572 Single UNIX Specification.

112573 XSI conformance can be thought of as a profile, selecting certain options from POSIX.1-200x.

#### 112574 A.2.1.5 Option Groups

112575 The concept of “Option Groups” is included to allow collections of related functions or options  
 112576 to be grouped together. This has been used as follows: the “XSI Option Groups” have been  
 112577 created to allow super-options, collections of underlying options and related functions, to be  
 112578 collectively supported by XSI-conforming systems.

112579 The standard developers considered the matter of subprofiling and decided it was better to  
 112580 include an enabling mechanism rather than detailed normative requirements. A set of  
 112581 subprofiling options was developed and included later in this volume of POSIX.1-200x as an  
 112582 informative illustration.

#### 112583 Subprofiling Considerations

112584 The goal of not simultaneously fixing maximums and minimums was to allow implementations  
 112585 of the base standard or standards to support multiple profiles without conflict.

112586 The following summarizes the rules for the limit types:

| Limit Type       | Fixed Value                | Minimum Acceptable Value                   | Maximum Acceptable Value                   |
|------------------|----------------------------|--------------------------------------------|--------------------------------------------|
| Standard Profile | $X_p = X_s$<br>(No change) | $Y_p \geq Y_s$<br>(May increase the limit) | $Z_p \leq Z_s$<br>(May decrease the limit) |

112592 The intent is that ranges specified by limits in profiles be entirely contained within the  
 112593 corresponding ranges of the base standard or standards being profiled, and that the unlimited  
 112594 end of a range in a base standard must remain unlimited in any profile of that standard.

112595 Thus, the fixed `_POSIX_*` limits are constants and must not be changed by a profile. The variable  
 112596 counterparts (typically without the leading `_POSIX_`) can be changed but still remain  
 112597 semantically the same; that is, they still allow implementation values to vary as long as they  
 112598 meet the requirements for that value (be it a minimum or maximum).

112599 Where a profile does not provide a feature upon which a limit is based, the limit is not relevant.  
 112600 Applications written to that profile should be written to operate independently of the value of  
 112601 the limit.

112602 An example which has previously allowed implementations to support both the base standard  
 112603 and two other profiles in a compatible manner follows:

```
112604 Base standard (POSIX.1-1996): _POSIX_CHILD_MAX 6
112605 Base standard: CHILD_MAX minimum maximum _POSIX_CHILD_MAX
112606 FIPS profile/SUSv2 CHILD_MAX 25 (minimum maximum)
```

112607 Another example:

112608 Base standard (POSIX.1-1996): `_POSIX_NGROUPS_MAX` 0  
 112609 Base standard: `NGROUPS_MAX` minimum maximum `_POSIX_NGROUP_MAX`  
 112610 FIPS profile/SUSv2 `NGROUPS_MAX` 8

112611 A profile may lower a minimum maximum below the equivalent `_POSIX` value:

112612 Base standard: `_POSIX_foo_MAX` Z  
 112613 Base standard: `foo_MAX` `_POSIX_foo_MAX`  
 112614 profile standard : `foo_MAX` X (X can be less than, equal to,  
 112615 or greater than `_POSIX_foo_MAX`)

112616 In this case an implementation conforming to the profile may not conform to the base standard,  
 112617 but an implementation to the base standard will conform to the profile.

#### 112618 A.2.1.6 Options

112619 The final subsections within *Implementation Conformance* list the core options within  
 112620 POSIX.1-200x. This includes both options for the System Interfaces volume of POSIX.1-200x and  
 112621 the Shell and Utilities volume of POSIX.1-200x.

### 112622 A.2.2 Application Conformance

112623 These definitions guide users or adaptors of applications in determining on which  
 112624 implementations an application will run and how much adaptation would be required to make  
 112625 it run on others. These definitions are modeled after related ones in the ISO C standard.

112626 POSIX.1 occasionally uses the expressions “portable application” or “conforming application”.  
 112627 As they are used, these are synonyms for any of these terms. The differences between the classes  
 112628 of application conformance relate to the requirements for other standards, the options supported  
 112629 (such as the XSI option) or, in the case of the Conforming POSIX.1 Application Using Extensions,  
 112630 to implementation extensions. When one of the less explicit expressions is used, it should be  
 112631 apparent from the context of the discussion which of the more explicit names is appropriate

#### 112632 A.2.2.1 Strictly Conforming POSIX Application

112633 This definition is analogous to that of an ISO C standard “conforming program”.

112634 The major difference between a Strictly Conforming POSIX Application and an ISO C standard  
 112635 strictly conforming program is that the latter is not allowed to use features of POSIX that are not  
 112636 in the ISO C standard.

#### 112637 A.2.2.2 Conforming POSIX Application

112638 Examples of <National Bodies> include ANSI, BSI, and AFNOR.

#### 112639 A.2.2.3 Conforming POSIX Application Using Extensions

112640 Due to possible requirements for configuration or implementation characteristics in excess of the  
 112641 specifications in <limits.h> or related to the hardware (such as array size or file space), not  
 112642 every Conforming POSIX Application Using Extensions will run on every conforming  
 112643 implementation.

112644 A.2.2.4 *Strictly Conforming XSI Application*

112645 This is intended to be upwards-compatible with the definition of a Strictly Conforming POSIX  
112646 Application, with the addition of the facilities and functionality included in the XSI option.

112647 A.2.2.5 *Conforming XSI Application Using Extensions*

112648 Such applications may use extensions beyond the facilities defined by POSIX.1-200x including  
112649 the XSI option, but need to document the additional requirements.

112650 **A.2.3 Language-Dependent Services for the C Programming Language**

112651 POSIX.1 is, for historical reasons, both a specification of an operating system interface, shell and  
112652 utilities, and a C binding for that specification. Efforts had been previously undertaken to  
112653 generate a language-independent specification; however, that had failed, and the fact that the  
112654 ISO C standard is the *de facto* primary language on POSIX and the UNIX system makes this a  
112655 necessary and workable situation.

112656 **A.2.4 Other Language-Related Specifications**

112657 There is no additional rationale provided for this section.

112658 **A.3 Definitions**

112659 The definitions in this section are stated so that they can be used as exact substitutes for the  
112660 terms in text. They should not contain requirements or cross-references to sections within  
112661 POSIX.1-200x; that is accomplished by using an informative note. In addition, the term should  
112662 not be included in its own definition. Where requirements or descriptions need to be addressed  
112663 but cannot be included in the definitions, due to not meeting the above criteria, these occur in  
112664 the General Concepts chapter.

112665 In this version, the definitions have been reworked extensively to meet style requirements and to  
112666 include terms from the base documents (see the Scope).

112667 Many of these definitions are necessarily circular, and some of the terms (such as “process”) are  
112668 variants of basic computing science terms that are inherently hard to define. Where some  
112669 definitions are more conceptual and contain requirements, these appear in the General Concepts  
112670 chapter. Those listed in this section appear in an alphabetical glossary format of terms.

112671 Some definitions must allow extension to cover terms or facilities that are not explicitly  
112672 mentioned in POSIX.1-200x. For example, the definition of “Extended Security Controls”  
112673 permits implementations beyond those defined in POSIX.1-200x.

112674 Some terms in the following list of notes do not appear in POSIX.1-200x; these are marked  
112675 suffixed with an asterisk (\*). Many of them have been specifically excluded from POSIX.1-200x  
112676 because they concern system administration, implementation, or other issues that are not  
112677 specific to the programming interface. Those are marked with a reason, such as  
112678 “implementation-defined”.

112679 **Appropriate Privileges**

112680 One of the fundamental security problems with many historical UNIX systems has been that the  
 112681 privilege mechanism is monolithic—a user has either no privileges or *all* privileges. Thus, a  
 112682 successful “trojan horse” attack on a privileged process defeats all security provisions.  
 112683 Therefore, POSIX.1 allows more granular privilege mechanisms to be defined. For many  
 112684 historical implementations of the UNIX system, the presence of the term “appropriate  
 112685 privileges” in POSIX.1 may be understood as a synonym for “superuser” (UID 0). However,  
 112686 other systems have emerged where this is not the case and each discrete controllable action has  
 112687 *appropriate privileges* associated with it. Because this mechanism is implementation-defined, it  
 112688 must be described in the conformance document. Although that description affects several parts  
 112689 of POSIX.1 where the term “appropriate privilege” is used, because the term “implementation-  
 112690 defined” only appears here, the description of the entire mechanism and its effects on these  
 112691 other sections belongs in this equivalent section of the conformance document. This is especially  
 112692 convenient for implementations with a single mechanism that applies in all areas, since it only  
 112693 needs to be described once.

112694 **Base Character\***

112695 The term “Base Character” has been removed, as it was felt that the use of this term within +  
 112696 POSIX.1-200x was common usage English.

112697 **Byte**

112698 The restriction that a byte is now exactly eight bits was a conscious decision by the standard  
 112699 developers. It came about due to a combination of factors, primarily the use of the type **int8\_t**  
 112700 within the networking functions and the alignment with the ISO/IEC 9899:1999 standard,  
 112701 where the **intN\_t** types are now defined.

112702 According to the ISO/IEC 9899:1999 standard:

- 112703 • The **[u]intN\_t** types must be two’s complement with no padding bits and no illegal values.
- 112704 • All types (apart from bit fields, which are not relevant here) must occupy an integral  
 112705 number of bytes.
- 112706 • If a type with width  $W$  occupies  $B$  bytes with  $C$  bits per byte ( $C$  is the value of  
 112707  $\{\text{CHAR\_BIT}\}$ ), then it has  $P$  padding bits where  $P+W=B*C$ .
- 112708 • Therefore, for **int8\_t**  $P=0$ ,  $W=8$ . Since  $B \geq 1$ ,  $C \geq 8$ , the only solution is  $B=1$ ,  $C=8$ .

112709 The standard developers also felt that this was not an undue restriction for the current state-of-  
 112710 the-art for this version of the standard, but recognize that if industry trends continue, a wider  
 112711 character type may be required in the future.

112712 **Character**

112713 The term “character” is used to mean a sequence of one or more bytes representing a single  
 112714 graphic symbol. The deviation in the exact text of the ISO C standard definition for “byte” meets  
 112715 the intent of the rationale of the ISO C standard also clears up the ambiguity raised by the term  
 112716 “basic execution character set”. The octet-minimum requirement is a reflection of the  
 112717  $\{\text{CHAR\_BIT}\}$  value.

- 112718 **Child Process**
- 112719 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/3 is applied, adding the *vfork()* function  
112720 to those listed.
- 112721 **Clock Tick**
- 112722 The ISO C standard defines a similar interval for use by the *clock()* function. There is no  
112723 requirement that these intervals be the same. In historical implementations these intervals are  
112724 different.
- 112725 **Command**
- 112726 The terms “command” and “utility” are related but have distinct meanings. Command is  
112727 defined as “a directive to a shell to perform a specific task”. The directive can be in the form of a  
112728 single utility name (for example, *ls*), or the directive can take the form of a compound command  
112729 (for example, “*ls | grep name | pr*”). A utility is a program that can be called by name  
112730 from a shell. Issuing only the name of the utility to a shell is the equivalent of a one-word  
112731 command. A utility may be invoked as a separate program that executes in a different process  
112732 than the command language interpreter, or it may be implemented as a part of the command  
112733 language interpreter. For example, the *echo* command (the directive to perform a specific task)  
112734 may be implemented such that the *echo* utility (the logic that performs the task of echoing) is in a  
112735 separate program; therefore, it is executed in a process that is different from the command  
112736 language interpreter. Conversely, the logic that performs the *echo* utility could be built into the  
112737 command language interpreter; therefore, it could execute in the same process as the command  
112738 language interpreter.
- 112739 The terms “tool” and “application” can be thought of as being synonymous with “utility” from  
112740 the perspective of the operating system kernel. Tools, applications, and utilities historically have  
112741 run, typically, in processes above the kernel level. Tools and utilities historically have been a part  
112742 of the operating system non-kernel code and have performed system-related functions, such as  
112743 listing directory contents, checking file systems, repairing file systems, or extracting system  
112744 status information. Applications have not generally been a part of the operating system, and  
112745 they perform non-system-related functions, such as word processing, architectural design,  
112746 mechanical design, workstation publishing, or financial analysis. Utilities have most frequently  
112747 been provided by the operating system distributor, applications by third-party software  
112748 distributors, or by the users themselves. Nevertheless, POSIX.1-200x does not differentiate  
112749 between tools, utilities, and applications when it comes to receiving services from the system, a  
112750 shell, or the standard utilities. (For example, the *xargs* utility invokes another utility; it would be  
112751 of fairly limited usefulness if the users could not run their own applications in place of the  
112752 standard utilities.) Utilities are not applications in the sense that they are not themselves subject  
112753 to the restrictions of POSIX.1-200x or any other standard—there is no requirement for *grep*, *stty*,  
112754 or any of the utilities defined here to be any of the classes of conforming applications.
- 112755 **Column Positions**
- 112756 In most 1-byte character sets, such as ASCII, the concept of column positions is identical to  
112757 character positions and to bytes. Therefore, it has been historically acceptable for some  
112758 implementations to describe line folding or tab stops or table column alignment in terms of  
112759 bytes or character positions. Other character sets pose complications, as they can have internal  
112760 representations longer than one octet and they can have display characters that have different  
112761 widths on the terminal screen or printer.
- 112762 In POSIX.1-200x the term “column positions” has been defined to mean character—not byte—  
112763 positions in input files (such as “column position 7 of the FORTRAN input”). Output files  
112764 describe the column position in terms of the display width of the narrowest printable character

112765 in the character set, adjusted to fit the characteristics of the output device. It is very possible that  
 112766  $n$  column positions will not be able to hold  $n$  characters in some character sets, unless all of those  
 112767 characters are of the narrowest width. It is assumed that the implementation is aware of the  
 112768 width of the various characters, deriving this information from the value of *LC\_CTYPE*, and  
 112769 thus can determine how many column positions to allot for each character in those utilities  
 112770 where it is important.

112771 The term “column position” was used instead of the more natural “column” because the latter is  
 112772 frequently used in the different contexts of columns of figures, columns of table values, and so  
 112773 on. Wherever confusion might result, these latter types of columns are referred to as “text  
 112774 columns”.

### 112775 **Controlling Terminal**

112776 The question of which of possibly several special files referring to the terminal is meant is not  
 112777 addressed in POSIX.1. The filename */dev/tty* is a synonym for the controlling terminal associated  
 112778 with a process.

### 112779 **Device Number\***

112780 The concept is handled in *stat()* as *ID of device*.

### 112781 **Direct I/O**

112782 Historically, direct I/O refers to the system bypassing intermediate buffering, but may be  
 112783 extended to cover implementation-defined optimizations.

### 112784 **Directory**

112785 The format of the directory file is implementation-defined and differs radically between  
 112786 System V and 4.3 BSD. However, routines (derived from 4.3 BSD) for accessing directories and  
 112787 certain constraints on the format of the information returned by those routines are described in  
 112788 the *<dirent.h>* header.

### 112789 **Directory Entry**

112790 Throughout POSIX.1-200x, the term “link” is used (about the *link()* function, for example) in  
 112791 describing the objects that point to files from directories.

### 112792 **Display**

112793 The Shell and Utilities volume of POSIX.1-200x assigns precise requirements for the terms  
 112794 “display” and “write”. Some historical systems have chosen to implement certain utilities  
 112795 without using the traditional file descriptor model. For example, the *vi* editor might employ  
 112796 direct screen memory updates on a personal computer, rather than a *write()* system call. An  
 112797 instance of user prompting might appear in a dialog box, rather than with standard error. When  
 112798 the Shell and Utilities volume of POSIX.1-200x uses the term “display”, the method of  
 112799 outputting to the terminal is unspecified; many historical implementations use *termcap* or  
 112800 *terminfo*, but this is not a requirement. The term “write” is used when the Shell and Utilities  
 112801 volume of POSIX.1-200x mandates that a file descriptor be used and that the output can be  
 112802 redirected. However, it is assumed that when the writing is directly to the terminal (it has not  
 112803 been redirected elsewhere), there is no practical way for a user or test suite to determine whether  
 112804 a file descriptor is being used. Therefore, the use of a file descriptor is mandated only for the  
 112805 redirection case and the implementation is free to use any method when the output is not  
 112806 redirected. The verb *write* is used almost exclusively, with the very few exceptions of those  
 112807 utilities where output redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

- 112808           **Dot**
- 112809           The symbolic name *dot* is carefully used in POSIX.1 to distinguish the working directory  
112810           filename from a period or a decimal point.
- 112811           **Dot-Dot**
- 112812           Historical implementations permit the use of these filenames without their special meanings.  
112813           Such use precludes any meaningful use of these filenames by a Conforming POSIX.1  
112814           Application. Therefore, such use is considered an extension, the use of which makes an  
112815           implementation non-conforming; see also [Section A.4.12](#) (on page 3350).
- 112816           **Epoch**
- 112817           Historically, the origin of UNIX system time was referred to as “00:00:00 GMT, January 1, 1970”.  
112818           Greenwich Mean Time is actually not a term acknowledged by the international standards  
112819           community; therefore, this term, “Epoch”, is used to abbreviate the reference to the actual  
112820           standard, Coordinated Universal Time.
- 112821           **FIFO Special File**
- 112822           See [Pipe](#) (on page 3334).
- 112823           **File**
- 112824           It is permissible for an implementation-defined file type to be non-readable or non-writable.
- 112825           **File Classes**
- 112826           These classes correspond to the historical sets of permission bits. The classes are general to  
112827           allow implementations flexibility in expanding the access mechanism for more stringent security  
112828           environments. Note that a process is in one and only one class, so there is no ambiguity.
- 112829           **Filename**
- 112830           At the present time, the primary responsibility for truncating filenames containing multi-byte  
112831           characters must reside with the application. Some industry groups involved in  
112832           internationalization believe that in the future the responsibility must reside with the kernel. For  
112833           the moment, a clearer understanding of the implications of making the kernel responsible for  
112834           truncation of multi-byte filenames is needed.
- 112835           Character-level truncation was not adopted because there is no support in POSIX.1 that advises  
112836           how the kernel distinguishes between single and multi-byte characters. Until that time, it must  
112837           be incumbent upon application developers to determine where multi-byte characters must be  
112838           truncated.
- 112839           **File System**
- 112840           Historically, the meaning of this term has been overloaded with two meanings: that of the  
112841           complete file hierarchy, and that of a mountable subset of that hierarchy; that is, a mounted file  
112842           system. POSIX.1 uses the term “file system” in the second sense, except that it is limited to the  
112843           scope of a process (and root directory of a process). This usage also clarifies the domain in which  
112844           a file serial number is unique.

- 112845           **Graphic Character**
- 112846           This definition is made available for those definitions (in particular, *TZ*) which must exclude  
112847           control characters.
- 112848           **Group Database**
- 112849           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/4 is applied, removing the words “of  
112850           implementation-defined format”. See [User Database](#) (on page 3344).
- 112851           **Group File\***
- 112852           Implementation-defined; see [User Database](#) (on page 3344).
- 112853           **Historical Implementations\***
- 112854           This refers to previously existing implementations of programming interfaces and operating  
112855           systems that are related to the interface specified by POSIX.1.
- 112856           **Hosted Implementation\***
- 112857           This refers to a POSIX.1 implementation that is accomplished through interfaces from the  
112858           POSIX.1 services to some alternate form of operating system kernel services. Note that the line  
112859           between a hosted implementation and a native implementation is blurred, since most  
112860           implementations will provide some services directly from the kernel and others through some  
112861           indirect path. (For example, *fopen()* might use *open()*; or *mkfifo()* might use *mknod()*.) There is  
112862           no necessary relationship between the type of implementation and its correctness, performance,  
112863           and/or reliability.
- 112864           **Implementation\***
- 112865           This term is generally used instead of its synonym, “system”, to emphasize the consequences of  
112866           decisions to be made by system implementors. Perhaps if no options or extensions to POSIX.1  
112867           were allowed, this usage would not have occurred.
- 112868           The term “specific implementation” is sometimes used as a synonym for “implementation”.  
112869           This should not be interpreted too narrowly; both terms can represent a relatively broad group  
112870           of systems. For example, a hardware vendor could market a very wide selection of systems that  
112871           all used the same instruction set, with some systems desktop models and others large multi-user  
112872           minicomputers. This wide range would probably share a common POSIX.1 operating system,  
112873           allowing an application compiled for one to be used on any of the others; this is a [*specific*]  
112874           *implementation*. However, such a wide range of machines probably has some differences  
112875           between the models. Some may have different clock rates, different file systems, different  
112876           resource limits, different network connections, and so on, depending on their sizes or intended  
112877           usages. Even on two identical machines, the system administrators may configure them  
112878           differently. Each of these different systems is known by the term “a specific instance of a specific  
112879           implementation”. This term is only used in the portions of POSIX.1 dealing with runtime  
112880           queries: *sysconf()* and *pathconf()*.



112881 **Incomplete Pathname\***

112882 Absolute pathname has been adequately defined.

112883 **Job Control**112884 In order to understand the job control facilities in POSIX.1 it is useful to understand how they  
112885 are used by a job control-cognizant shell to create the user interface effect of job control.112886 While the job control facilities supplied by POSIX.1 can, in theory, support different types of  
112887 interactive job control interfaces supplied by different types of shells, there was historically one  
112888 particular interface that was most common when the standard was originally developed  
112889 (provided by BSD C Shell).112890 This discussion describes that interface as a means of illustrating how the POSIX.1 job control  
112891 facilities can be used.112892 Job control allows users to selectively stop (suspend) the execution of processes and continue  
112893 (resume) their execution at a later point. The user typically employs this facility via the  
112894 interactive interface jointly supplied by the terminal I/O driver and a command interpreter  
112895 (shell).112896 The user can launch jobs (command pipelines) in either the foreground or background. When  
112897 launched in the foreground, the shell waits for the job to complete before prompting for  
112898 additional commands. When launched in the background, the shell does not wait, but  
112899 immediately prompts for new commands.112900 If the user launches a job in the foreground and subsequently regrets this, the user can type the  
112901 suspend character (typically set to <control>-Z), which causes the foreground job to stop and the  
112902 shell to begin prompting for new commands. The stopped job can be continued by the user (via  
112903 special shell commands) either as a foreground job or as a background job. Background jobs can  
112904 also be moved into the foreground via shell commands.112905 If a background job attempts to access the login terminal (controlling terminal), it is stopped by  
112906 the terminal driver and the shell is notified, which, in turn, notifies the user. (Terminal access  
112907 includes *read()* and certain terminal control functions, and conditionally includes *write()*.) The  
112908 user can continue the stopped job in the foreground, thus allowing the terminal access to  
112909 succeed in an orderly fashion. After the terminal access succeeds, the user can optionally move  
112910 the job into the background via the suspend character and shell commands.112911 *Implementing Job Control Shells*112912 The interactive interface described previously can be accomplished using the POSIX.1 job  
112913 control facilities in the following way.112914 The key feature necessary to provide job control is a way to group processes into jobs. This  
112915 grouping is necessary in order to direct signals to a single job and also to identify which job is in  
112916 the foreground. (There is at most one job that is in the foreground on any controlling terminal at  
112917 a time.)112918 The concept of process groups is used to provide this grouping. The shell places each job in a  
112919 separate process group via the *setpgid()* function. To do this, the *setpgid()* function is invoked by  
112920 the shell for each process in the job. It is actually useful to invoke *setpgid()* twice for each  
112921 process: once in the child process, after calling *fork()* to create the process, but before calling one  
112922 of the *exec* family of functions to begin execution of the program, and once in the parent shell  
112923 process, after calling *fork()* to create the child. The redundant invocation avoids a race condition  
112924 by ensuring that the child process is placed into the new process group before either the parent  
112925 or the child relies on this being the case. The process group ID for the job is selected by the shell  
112926 to be equal to the process ID of one of the processes in the job. Some shells choose to make one

112927 process in the job be the parent of the other processes in the job (if any). Other shells (for  
 112928 example, the C Shell) choose to make themselves the parent of all processes in the pipeline (job).  
 112929 In order to support this latter case, the *setpgid()* function accepts a process group ID parameter  
 112930 since the correct process group ID cannot be inherited from the shell. The shell itself is  
 112931 considered to be a job and is the sole process in its own process group.

112932 The shell also controls which job is currently in the foreground. A foreground and background  
 112933 job differ in two ways: the shell waits for a foreground command to complete (or stop) before  
 112934 continuing to read new commands, and the terminal I/O driver inhibits terminal access by  
 112935 background jobs (causing the processes to stop). Thus, the shell must work cooperatively with  
 112936 the terminal I/O driver and have a common understanding of which job is currently in the  
 112937 foreground. It is the user who decides which command should be currently in the foreground,  
 112938 and the user informs the shell via shell commands. The shell, in turn, informs the terminal I/O  
 112939 driver via the *tcsetpgrp()* function. This indicates to the terminal I/O driver the process group ID  
 112940 of the foreground process group (job). When the current foreground job either stops or  
 112941 terminates, the shell places itself in the foreground via *tcsetpgrp()* before prompting for  
 112942 additional commands. Note that when a job is created the new process group begins as a  
 112943 background process group. It requires an explicit act of the shell via *tcsetpgrp()* to move a  
 112944 process group (job) into the foreground.

112945 When a process in a job stops or terminates, its parent (for example, the shell) receives  
 112946 synchronous notification by calling the *waitpid()* function with the WUNTRACED flag set.  
 112947 Asynchronous notification is also provided when the parent establishes a signal handler for  
 112948 SIGCHLD and does not specify the SA\_NOCLDSTOP flag. Usually all processes in a job stop as  
 112949 a unit since the terminal I/O driver always sends job control stop signals to all processes in the  
 112950 process group.

112951 To continue a stopped job, the shell sends the SIGCONT signal to the process group of the job. In  
 112952 addition, if the job is being continued in the foreground, the shell invokes *tcsetpgrp()* to place the  
 112953 job in the foreground before sending SIGCONT. Otherwise, the shell leaves itself in the  
 112954 foreground and reads additional commands.

112955 There is additional flexibility in the POSIX.1 job control facilities that allows deviations from the  
 112956 typical interface. Clearing the TOSTOP terminal flag allows background jobs to perform *write()*  
 112957 functions without stopping. The same effect can be achieved on a per-process basis by having a  
 112958 process set the signal action for SIGTTOU to SIG\_IGN.

112959 Note that the terms “job” and “process group” can be used interchangeably. A login session that  
 112960 is not using the job control facilities can be thought of as a large collection of processes that are  
 112961 all in the same job (process group). Such a login session may have a partial distinction between  
 112962 foreground and background processes; that is, the shell may choose to wait for some processes  
 112963 before continuing to read new commands and may not wait for other processes. However, the  
 112964 terminal I/O driver will consider all these processes to be in the foreground since they are all  
 112965 members of the same process group.

112966 In addition to the basic job control operations already mentioned, a job control-cognizant shell  
 112967 needs to perform the following actions.

112968 When a foreground (not background) job stops, the shell must sample and remember the current  
 112969 terminal settings so that it can restore them later when it continues the stopped job in the  
 112970 foreground (via the *tcgetattr()* and *tcsetattr()* functions).

112971 Because a shell itself can be spawned from a shell, it must take special action to ensure that  
 112972 subshells interact well with their parent shells.

112973 A subshell can be spawned to perform an interactive function (prompting the terminal for  
 112974 commands) or a non-interactive function (reading commands from a file). When operating non-

112975 interactively, the job control shell will refrain from performing the job control-specific actions  
 112976 described above. It will behave as a shell that does not support job control. For example, all jobs  
 112977 will be left in the same process group as the shell, which itself remains in the process group  
 112978 established for it by its parent. This allows the shell and its children to be treated as a single job  
 112979 by a parent shell, and they can be affected as a unit by terminal keyboard signals.

112980 An interactive subshell can be spawned from another job control-cognizant shell in either the  
 112981 foreground or background. (For example, from the C Shell, the user can execute the command,  
 112982 `csch &`.) Before the subshell activates job control by calling `setpgid()` to place itself in its own  
 112983 process group and `tcsetgrp()` to place its new process group in the foreground, it needs to  
 112984 ensure that it has already been placed in the foreground by its parent. (Otherwise, there could  
 112985 be multiple job control shells that simultaneously attempt to control mediation of the terminal.)  
 112986 To determine this, the shell retrieves its own process group via `getpgrp()` and the process group  
 112987 of the current foreground job via `tcgetpgrp()`. If these are not equal, the shell sends SIGTTIN to  
 112988 its own process group, causing itself to stop. When continued later by its parent, the shell  
 112989 repeats the process group check. When the process groups finally match, the shell is in the  
 112990 foreground and it can proceed to take control. After this point, the shell ignores all the job  
 112991 control stop signals so that it does not inadvertently stop itself.

#### 112992 *Implementing Job Control Applications*

112993 Most applications do not need to be aware of job control signals and operations; the intuitively  
 112994 correct behavior happens by default. However, sometimes an application can inadvertently  
 112995 interfere with normal job control processing, or an application may choose to overtly effect job  
 112996 control in cooperation with normal shell procedures.

112997 An application can inadvertently subvert job control processing by “blindly” altering the  
 112998 handling of signals. A common application error is to learn how many signals the system  
 112999 supports and to ignore or catch them all. Such an application makes the assumption that it does  
 113000 not know what this signal is, but knows the right handling action for it. The system may  
 113001 initialize the handling of job control stop signals so that they are being ignored. This allows  
 113002 shells that do not support job control to inherit and propagate these settings and hence to be  
 113003 immune to stop signals. A job control shell will set the handling to the default action and  
 113004 propagate this, allowing processes to stop. In doing so, the job control shell is taking  
 113005 responsibility for restarting the stopped applications. If an application wishes to catch the stop  
 113006 signals itself, it should first determine their inherited handling states. If a stop signal is being  
 113007 ignored, the application should continue to ignore it. This is directly analogous to the  
 113008 recommended handling of SIGINT described in the referenced UNIX Programmer’s Manual.

113009 If an application is reading the terminal and has disabled the interpretation of special characters  
 113010 (by clearing the ISIG flag), the terminal I/O driver will not send SIGTSTP when the suspend  
 113011 character is typed. Such an application can simulate the effect of the suspend character by  
 113012 recognizing it and sending SIGTSTP to its process group as the terminal driver would have  
 113013 done. Note that the signal is sent to the process group, not just to the application itself; this  
 113014 ensures that other processes in the job also stop. (Note also that other processes in the job could  
 113015 be children, siblings, or even ancestors.) Applications should not assume that the suspend  
 113016 character is `<control>-Z` (or any particular value); they should retrieve the current setting at  
 113017 startup.

113018 *Implementing Job Control Systems*

113019 The intent in adding 4.2 BSD-style job control functionality was to adopt the necessary 4.2 BSD  
 113020 programmatic interface with only minimal changes to resolve syntactic or semantic conflicts  
 113021 with System V or to close recognized security holes. The goal was to maximize the ease of  
 113022 providing both conforming implementations and Conforming POSIX.1 Applications.

113023 It is only useful for a process to be affected by job control signals if it is the descendant of a job  
 113024 control shell. Otherwise, there will be nothing that continues the stopped process.

113025 POSIX.1 does not specify how controlling terminal access is affected by a user logging out (that  
 113026 is, by a controlling process terminating). 4.2 BSD uses the *vhangup()* function to prevent any  
 113027 access to the controlling terminal through file descriptors opened prior to logout. System V does  
 113028 not prevent controlling terminal access through file descriptors opened prior to logout (except  
 113029 for the case of the special file, */dev/tty*). Some implementations choose to make processes  
 113030 immune from job control after logout (that is, such processes are always treated as if in the  
 113031 foreground); other implementations continue to enforce foreground/background checks after  
 113032 logout. Therefore, a Conforming POSIX.1 Application should not attempt to access the  
 113033 controlling terminal after logout since such access is unreliable. If an implementation chooses to  
 113034 deny access to a controlling terminal after its controlling process exits, POSIX.1 requires a certain  
 113035 type of behavior (see [Controlling Terminal](#), on page 3326).

113036 **Kernel\***

113037 See [System Call\\*](#) (on page 3342).

113038 **Library Routine\***

113039 See [System Call\\*](#) (on page 3342).

113040 **Logical Device\***

113041 Implementation-defined.

113042 **Map**

113043 The definition of map is included to clarify the usage of mapped pages in the description of the  
 113044 behavior of process memory locking.

113045 **Memory-Resident**

113046 The term “memory-resident” is historically understood to mean that the so-called resident pages  
 113047 are actually present in the physical memory of the computer system and are immune from  
 113048 swapping, paging, copy-on-write faults, and so on. This is the actual intent of POSIX.1-200x  
 113049 in the process memory locking section for implementations where this is logical. But for some  
 113050 implementations—primarily mainframes—actually locking pages into primary storage is not  
 113051 advantageous to other system objectives, such as maximizing throughput. For such  
 113052 implementations, memory locking is a “hint” to the implementation that the application wishes  
 113053 to avoid situations that would cause long latencies in accessing memory. Furthermore, there are  
 113054 other implementation-defined issues with minimizing memory access latencies that “memory  
 113055 residency” does not address—such as MMU reload faults. The definition attempts to  
 113056 accommodate various implementations while allowing conforming applications to specify to the  
 113057 implementation that they want or need the best memory access times that the implementation  
 113058 can provide.

113059 **Memory Object\***

113060 The term “memory object” usually implies shared memory. If the object is the same as a  
 113061 filename in the file system name space of the implementation, it is expected that the data written  
 113062 into the memory object be preserved on disk. A memory object may also apply to a physical  
 113063 device on an implementation. In this case, writes to the memory object are sent to the controller  
 113064 for the device and reads result in control registers being returned.

113065 **Mount Point\***

113066 The directory on which a “mounted file system” is mounted. This term, like *mount()* and  
 113067 *umount()*, was not included because it was implementation-defined.

113068 **Mounted File System\***

113069 See [File System](#) (on page 3327).

113070 **Name**

113071 There are no explicit limits in POSIX.1-200x on the sizes of names, words (see the definition of  
 113072 word in the Base Definitions volume of POSIX.1-200x), lines, or other objects. However, other  
 113073 implicit limits do apply: shell script lines produced by many of the standard utilities cannot  
 113074 exceed {LINE\_MAX} and the sum of exported variables comes under the {ARG\_MAX} limit.  
 113075 Historical shells dynamically allocate memory for names and words and parse incoming lines a  
 113076 character at a time. Lines cannot have an arbitrary {LINE\_MAX} limit because of historical  
 113077 practice, such as *makefiles*, where *make* removes the <newline>s associated with the commands  
 113078 for a target and presents the shell with one very long line. The text on INPUT FILES in XCU  
 113079 [Section 1.4](#) (on page 2235) does allow a shell to run out of memory, but it cannot have arbitrary  
 113080 programming limits.

113081 **Native Implementation\***

113082 This refers to an implementation of POSIX.1 that interfaces directly to an operating system  
 113083 kernel; see also *hosted implementation*. A similar concept is a native UNIX system, which would  
 113084 be a kernel derived from one of the original UNIX system products.

113085 **Nice Value**

113086 This definition is not intended to suggest that all processes in a system have priorities that are  
 113087 comparable. Scheduling policy extensions, such as adding realtime priorities, make the notion of  
 113088 a single underlying priority for all scheduling policies problematic. Some implementations may  
 113089 implement the features related to *nice* to affect all processes on the system, others to affect just  
 113090 the general time-sharing activities implied by POSIX.1-200x, and others may have no effect at all.  
 113091 Because of the use of “implementation-defined” in *nice* and *renice*, a wide range of  
 113092 implementation strategies is possible.

113093 **Open File Description**

113094 An “open file description”, as it is currently named, describes how a file is being accessed. What  
 113095 is currently called a “file descriptor” is actually just an identifier or “handle”; it does not  
 113096 actually describe anything.

113097 The following alternate names were discussed:

- 113098 • For “open file description”:
- 113099 “open instance”, “file access description”, “open file information”, and “file access  
 113100 information”.

- 113101           • For “file descriptor”:  
 113102           “file handle”, “file number” (cf., *fileno()*). Some historical implementations use the term  
 113103           “file table entry”.

#### 113104           **Orphaned Process Group**

113105           Historical implementations have a concept of an orphaned process, which is a process whose  
 113106           parent process has exited. When job control is in use, it is necessary to prevent processes from  
 113107           being stopped in response to interactions with the terminal after they no longer are controlled by  
 113108           a job control-cognizant program. Because signals generated by the terminal are sent to a process  
 113109           group and not to individual processes, and because a signal may be provoked by a process that  
 113110           is not orphaned, but sent to another process that is orphaned, it is necessary to define an  
 113111           orphaned process group. The definition assumes that a process group will be manipulated as a  
 113112           group and that the job control-cognizant process controlling the group is outside of the group  
 113113           and is the parent of at least one process in the group (so that state changes may be reported via  
 113114           *waitpid()*). Therefore, a group is considered to be controlled as long as at least one process in the  
 113115           group has a parent that is outside of the process group, but within the session.

113116           This definition of orphaned process groups ensures that a session leader’s process group is  
 113117           always considered to be orphaned, and thus it is prevented from stopping in response to  
 113118           terminal signals.

#### 113119           **Page**

113120           The term “page” is defined to support the description of the behavior of memory mapping for  
 113121           shared memory and memory mapped files, and the description of the behavior of process  
 113122           memory locking. It is not intended to imply that shared memory/file mapping and memory  
 113123           locking are applicable only to “paged” architectures. For the purposes of POSIX.1-200x,  
 113124           whatever the granularity on which an architecture supports mapping or locking, this is  
 113125           considered to be a “page”. If an architecture cannot support the memory mapping or locking  
 113126           functions specified by POSIX.1-200x on any granularity, then these options will not be  
 113127           implemented on the architecture.

#### 113128           **Passwd File\***

113129           Implementation-defined; see [User Database](#) (on page 3344).

#### 113130           **Parent Directory**

113131           There may be more than one directory entry pointing to a given directory in some  
 113132           implementations. The wording here identifies that exactly one of those is the parent directory. In  
 113133           pathname resolution, dot-dot is identified as the way that the unique directory is identified.  
 113134           (That is, the parent directory is the one to which dot-dot points.) In the case of a remote file  
 113135           system, if the same file system is mounted several times, it would appear as if they were distinct  
 113136           file systems (with interesting synchronization properties).

#### 113137           **Pipe**

113138           It proved convenient to define a pipe as a special case of a FIFO, even though historically the  
 113139           latter was not introduced until System III and does not exist at all in 4.3 BSD.

113140 **Portable Filename Character Set**

113141 The encoding of this character set is not specified—specifically, ASCII is not required. But the  
 113142 implementation must provide a unique character code for each of the printable graphics  
 113143 specified by POSIX.1; see also [Section A.4.6](#) (on page 3346).

113144 Situations where characters beyond the portable filename character set (or historically ASCII or  
 113145 the ISO/IEC 646:1991 standard) would be used (in a context where the portable filename  
 113146 character set or the ISO/IEC 646:1991 standard is required by POSIX.1) are expected to be  
 113147 common. Although such a situation renders the use technically non-compliant, mutual  
 113148 agreement among the users of an extended character set will make such use portable between  
 113149 those users. Such a mutual agreement could be formalized as an optional extension to POSIX.1.  
 113150 (Making it required would eliminate too many possible systems, as even those systems using the  
 113151 ISO/IEC 646:1991 standard as a base character set extend their character sets for Western  
 113152 Europe and the rest of the world in different ways.)

113153 Nothing in POSIX.1 is intended to preclude the use of extended characters where interchange is  
 113154 not required or where mutual agreement is obtained. It has been suggested that in several places  
 113155 “should” be used instead of “shall”. Because (in the worst case) use of any character beyond the  
 113156 portable filename character set would render the program or data not portable to all possible  
 113157 systems, no extensions are permitted in this context.

113158 **Process Lifetime**

113159 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/5 is applied, adding *fork()*, *posix\_spawn()*,  
 113160 *posix\_spawnp()*, and *vfork()* to the list of functions.

113161 **Process Termination**

113162 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/6 is applied, rewording the definition to  
 113163 address the “passive exit” on termination of the last thread or the *\_Exit()* function.

113164 **Regular File**

113165 POSIX.1 does not intend to preclude the addition of structuring data (for example, record  
 113166 lengths) in the file, as long as such data is not visible to an application that uses the features  
 113167 described in POSIX.1.

113168 **Root Directory**

113169 This definition permits the operation of *chroot()*, even though that function is not in POSIX.1; see  
 113170 also [Section A.4.5](#) (on page 3346).

113171 **Root File System\***

113172 Implementation-defined.

113173 **Root of a File System\***

113174 Implementation-defined; see [Mount Point\\*](#) (on page 3333).

113175 **Signal**

113176 The definition implies a double meaning for the term. Although a signal is an event, common  
113177 usage implies that a signal is an identifier of the class of event.

113178 **Superuser\***

113179 This concept, with great historical significance to UNIX system users, has been replaced with the  
113180 notion of appropriate privileges.

113181 **Supplementary Group ID**

113182 The POSIX.1-1990 standard is inconsistent in its treatment of supplementary groups. The  
113183 definition of supplementary group ID explicitly permits the effective group ID to be included in  
113184 the set, but wording in the description of the *setuid()* and *setgid()* functions states: “Any  
113185 supplementary group IDs of the calling process remain unchanged by these function calls”. In  
113186 the case of *setgid()* this contradicts that definition. In addition, some felt that the unspecified  
113187 behavior in the definition of supplementary group IDs adds unnecessary portability problems.  
113188 The standard developers considered several solutions to this problem:

- 113189 1. Reword the description of *setgid()* to permit it to change the supplementary group IDs to  
113190 reflect the new effective group ID. A problem with this is that it adds more “may”s to the  
113191 wording and does not address the portability problems of this optional behavior.
- 113192 2. Mandate the inclusion of the effective group ID in the supplementary set (giving  
113193 {NGROUPS\_MAX} a minimum value of 1). This is the behavior of 4.4 BSD. In that  
113194 system, the effective group ID is the first element of the array of supplementary group  
113195 IDs (there is no separate copy stored, and changes to the effective group ID are made only  
113196 in the supplementary group set). By convention, the initial value of the effective group ID  
113197 is duplicated elsewhere in the array so that the initial value is not lost when executing a  
113198 set-group-ID program.
- 113199 3. Change the definition of supplementary group ID to exclude the effective group ID and  
113200 specify that the effective group ID does not change the set of supplementary group IDs.  
113201 This is the behavior of 4.2 BSD, 4.3 BSD, and System V Release 4.
- 113202 4. Change the definition of supplementary group ID to exclude the effective group ID, and  
113203 require that *getgroups()* return the union of the effective group ID and the supplementary  
113204 group IDs.
- 113205 5. Change the definition of {NGROUPS\_MAX} to be one more than the number of  
113206 supplementary group IDs, so it continues to be the number of values returned by  
113207 *getgroups()* and existing applications continue to work. This alternative is effectively the  
113208 same as the second (and might actually have the same implementation).

113209 The standard developers decided to permit either 2 or 3. The effective group ID is orthogonal to  
113210 the set of supplementary group IDs, and it is implementation-defined whether *getgroups()*  
113211 returns this. If the effective group ID is returned with the set of supplementary group IDs, then  
113212 all changes to the effective group ID affect the supplementary group set returned by *getgroups()*.  
113213 It is permissible to eliminate duplicates from the list returned by *getgroups()*. However, if a  
113214 group ID is contained in the set of supplementary group IDs, setting the group ID to that value  
113215 and then to a different value should not remove that value from the supplementary group IDs.

113216 The definition of supplementary group IDs has been changed to not include the effective group  
113217 ID. This simplifies permanent rationale and makes the relevant functions easier to understand.  
113218 The *getgroups()* function has been modified so that it can, on an implementation-defined basis,  
113219 return the effective group ID. By making this change, functions that modify the effective group  
113220 ID do not need to discuss adding to the supplementary group list; the only view into the



113221 supplementary group list that the application developer has is through the *getgroups()* function. |

### 113222 **Symbolic Constant**

113223 Earlier versions of this standard used a variety of terms other than “macro” for many of the +  
 113224 constants defined in headers, and it was not clear in which of these cases they were required to +  
 113225 be macros or not, or to be pre-processor constants (i.e., usable in **#if**) or not. In cases where the +  
 113226 symbols had a reserved prefix or suffix, there was often inconsistency between whether the +  
 113227 prefix/suffix was reserved only for macros or for any use, and whether the term “macro” or a +  
 113228 different term was used in the descriptions of the symbols. There were also some unintentional +  
 113229 differences from the ISO C standard. +

113230 One of the most commonly used terms was “symbolic constant”. This has now been designated +  
 113231 as the default term to be used wherever appropriate, and a formal definition of the term has +  
 113232 been added giving the exact requirements for symbols that are described as symbolic constants. +

113233 The standard developers have performed a major rationalization of the header descriptions of +  
 113234 symbols with constant values according to the following policy: +

- 113235 • Where symbols are from the ISO C standard, the wording from the ISO C standard (or +  
 113236 equivalent, in cases where the exact wording is not appropriate) is used to describe them. +
- 113237 • For all other constants, the first choice is to use “symbolic constant” when the +  
 113238 requirements for the symbol are a reasonably close fit with those of the term. +

113239 The description of the symbol can override individual requirements for symbolic +  
 113240 constants; e.g., to specify a non-integer type, or to add a requirement that the symbol is +  
 113241 usable in **#if** preprocessor directives. +

- 113242 • When neither of the above apply, the exact requirements are stated in the description. +  
 113243 (Note that macros are not required to be usable in **#if**, or even to expand to constant +  
 113244 expressions, unless explicitly stated.) +

- 113245 • In cases where there is a reserved prefix or suffix, if the symbol(s) with that prefix/suffix +  
 113246 are from the ISO C standard and are required to be macros, or if the symbol is required to +  
 113247 be usable in **#if**, then the prefix/suffix is reserved for use only as macros. If the symbol(s) +  
 113248 are “symbolic constants” and not required to be usable in **#if**, the prefix/suffix is reserved +  
 113249 for any use except in a few special cases. +

113250 Where a constant is required to be a macro but is also allowed to be another type of constant +  
 113251 such as an enumeration constant, on implementations which do define it as another type of +  
 113252 constant the macro is typically defined as follows: +

```
113253 #define macro_name macro_name +
```

113254 This allows applications to use **#ifdef**, etc. to determine whether the macro is defined, but the +  
 113255 macro is not usable in **#if** preprocessor directives because the preprocessor will treat the +  
 113256 unexpanded word *macro\_name* as having the value zero. +

### 113257 **Symbolic Link**

113258 Many implementations associate no attributes, including ownership with symbolic links. +  
 113259 Security experts encouraged consideration for defining these attributes as optional. +  
 113260 Consideration was given to changing *utime()* to allow modification of the times for a symbolic +  
 113261 link, or as an alternative adding an *lutime()* interface. Modifications to *chown()* were also +  
 113262 considered: allow changing symbolic link ownership or alternatively adding *lchown()*. As a +  
 113263 result of alignment with the Single UNIX Specification, the *lchown()* function is included as part +  
 113264 of the XSI option and XSI-conformant systems may support an owner and a group associated +  
 113265 with a symbolic link. There was no consensus to define further attributes for symbolic links, +

113266 and for systems not supporting the XSI option only the file type bits in the *st\_mode* member and  
113267 the *st\_size* member of the **stat** structure are required to be applicable to symbolic links.

113268 Historical implementations were followed when determining which interfaces should apply to  
113269 symbolic links. Interfaces that historically followed symbolic links include *chmod()*, *link()*, and  
113270 *utime()*. Interfaces that historically do not follow symbolic links include *chown()*, *lstat()*,  
113271 *readlink()*, *rename()*, *remove()*, *rmdir()*, and *unlink()*. POSIX.1-200x deviates from historical  
113272 practice only in the case of *chown()*. Because there is no requirement for systems not supporting  
113273 the XSI extension that there is an association of ownership with symbolic links, there was no  
113274 interface in the base standard to change ownership. In addition, other implementations of  
113275 symbolic links have modified *chown()* to follow symbolic links.

113276 In the case of symbolic links, POSIX.1-200x states that a trailing slash is considered to be the final  
113277 component of a pathname rather than the pathname component that preceded it. This is the  
113278 behavior of historical implementations. For example, for */a/b* and */a/b/*, if */a/b* is a symbolic link  
113279 to a directory, then */a/b* refers to the symbolic link, and */a/b/* is the same as */a/b/.*, which is the  
113280 directory to which the symbolic link points.

113281 For multi-level security purposes, it is possible to have the link read mode govern permission  
113282 for the *readlink()* function. It is also possible that the read permissions of the directory containing  
113283 the link be used for this purpose. Implementations may choose to use either of these methods;  
113284 however, this is not current practice and neither method is specified.

113285 Several reasons were advanced for requiring that when a symbolic link is used as the source  
113286 argument to the *link()* function, the resulting link will apply to the file named by the contents of  
113287 the symbolic link rather than to the symbolic link itself. This is the case in historical  
113288 implementations. This action was preferred, as it supported the traditional idea of persistence  
113289 with respect to the target of a hard link. This decision is appropriate in light of a previous  
113290 decision not to require association of attributes with symbolic links, thereby allowing  
113291 implementations which do not use inodes. Opposition centered on the lack of symmetry on the  
113292 part of the *link()* and *unlink()* function pair with respect to symbolic links.

113293 Because a symbolic link and its referenced object coexist in the file system name space, confusion  
113294 can arise in distinguishing between the link itself and the referenced object. Historically, utilities  
113295 and system calls have adopted their own link following conventions in a somewhat *ad hoc*  
113296 fashion. Rules for a uniform approach are outlined here, although historical practice has been  
113297 adhered to as much as was possible. To promote consistent system use, user-written utilities are  
113298 encouraged to follow these same rules.

113299 Symbolic links are handled either by operating on the link itself, or by operating on the object  
113300 referenced by the link. In the latter case, an application or system call is said to “follow” the link.  
113301 Symbolic links may reference other symbolic links, in which case links are dereferenced until an  
113302 object that is not a symbolic link is found, a symbolic link that references a file that does not exist  
113303 is found, or a loop is detected. (Current implementations do not detect loops, but have a limit on  
113304 the number of symbolic links that they will dereference before declaring it an error.)

113305 There are four domains for which default symbolic link policy is established in a system. In  
113306 almost all cases, there are utility options that override this default behavior. The four domains  
113307 are as follows:

- 113308 1. Symbolic links specified to system calls that take filename arguments
- 113309 2. Symbolic links specified as command line filename arguments to utilities that are not  
113310 performing a traversal of a file hierarchy
- 113311 3. Symbolic links referencing files not of type directory, specified to utilities that are  
113312 performing a traversal of a file hierarchy

- 113313 4. Symbolic links referencing files of type directory, specified to utilities that are performing  
113314 a traversal of a file hierarchy

113315 *First Domain*

113316 The first domain is considered in earlier rationale.

113317 *Second Domain*

113318 The reason this category is restricted to utilities that are not traversing the file hierarchy is that  
113319 some standard utilities take an option that specifies a hierarchical traversal, but by default  
113320 operate on the arguments themselves. Generally, users specifying the option for a file hierarchy  
113321 traversal wish to operate on a single, physical hierarchy, and therefore symbolic links, which  
113322 may reference files outside of the hierarchy, are ignored. For example, *chown owner file* is a  
113323 different operation from the same command with the **-R** option specified. In this example, the  
113324 behavior of the command *chown owner file* is described here, while the behavior of the command  
113325 *chown -R owner file* is described in the third and fourth domains.

113326 The general rule is that the utilities in this category follow symbolic links named as arguments.

113327 Exceptions in the second domain are:

- 113328 • The *mv* and *rm* utilities do not follow symbolic links named as arguments, but respectively  
113329 attempt to rename or delete them.
- 113330 • The *ls* utility is also an exception to this rule. For compatibility with historical systems,  
113331 when the **-R** option is not specified, the *ls* utility follows symbolic links named as  
113332 arguments if the **-L** option is specified or if the **-F**, **-d**, or **-l** options are not specified. (If  
113333 the **-L** option is specified, *ls* always follows symbolic links; it is the only utility where the  
113334 **-L** option affects its behavior even though a tree walk is not being performed.)

113335 All other standard utilities, when not traversing a file hierarchy, always follow symbolic links  
113336 named as arguments.

113337 Historical practice is that the **-h** option is specified if standard utilities are to act upon symbolic  
113338 links instead of upon their targets. Examples of commands that have historically had a **-h** option  
113339 for this purpose are the *chgrp*, *chown*, *file*, and *test* utilities.

113340 *Third Domain*

113341 The third domain is symbolic links, referencing files not of type directory, specified to utilities  
113342 that are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
113343 command line filename arguments or encountered during the traversal.)

113344 The intention of the Shell and Utilities volume of POSIX.1-200x is that the operation that the  
113345 utility is performing is applied to the symbolic link itself, if that operation is applicable to  
113346 symbolic links. The reason that the operation is not required is that symbolic links in some  
113347 implementations do not have such attributes as a file owner, and therefore the *chown* operation  
113348 would be meaningless. If symbolic links on the system have an owner, it is the intention that the  
113349 utility *chown* cause the owner of the symbolic link to change. If symbolic links do not have an  
113350 owner, the symbolic link should be ignored. Specifically, by default, no change should be made  
113351 to the file referenced by the symbolic link.

113352 *Fourth Domain*

113353 The fourth domain is symbolic links referencing files of type directory, specified to utilities that  
 113354 are performing a traversal of a file hierarchy. (This includes symbolic links specified as  
 113355 command line filename arguments or encountered during the traversal.)

113356 Most standard utilities do not, by default, indirect into the file hierarchy referenced by the  
 113357 symbolic link. (The Shell and Utilities volume of POSIX.1-200x uses the informal term “physical  
 113358 walk” to describe this case. The case where the utility does indirect through the symbolic link is  
 113359 termed a “logical walk”.)

113360 There are three reasons for the default to be a physical walk:

- 113361 1. With very few exceptions, a physical walk has been the historical default on UNIX  
 113362 systems supporting symbolic links. Because some utilities (that is, *rm*) must default to a  
 113363 physical walk, regardless, changing historical practice in this regard would be confusing  
 113364 to users and needlessly incompatible.
- 113365 2. For systems where symbolic links have the historical file attributes (that is, *owner*, *group*,  
 113366 *mode*), defaulting to a logical traversal would require the addition of a new option to the  
 113367 commands to modify the attributes of the link itself. This is painful and more complex  
 113368 than the alternatives.
- 113369 3. There is a security issue with defaulting to a logical walk. Historically, the command  
 113370 *chown -R user file* has been safe for the superuser because *setuid* and *setgid* bits were lost  
 113371 when the ownership of the file was changed. If the walk were logical, changing  
 113372 ownership would no longer be safe because a user might have inserted a symbolic link  
 113373 pointing to any file in the tree. Again, this would necessitate the addition of an option to  
 113374 the commands doing hierarchy traversal to not indirect through the symbolic links, and  
 113375 historical scripts doing recursive walks would instantly become security problems. While  
 113376 this is mostly an issue for system administrators, it is preferable to not have different  
 113377 defaults for different classes of users.

113378 However, the standard developers agreed to leave it unspecified to achieve consensus.

113379 As consistently as possible, users may cause standard utilities performing a file hierarchy  
 113380 traversal to follow any symbolic links named on the command line, regardless of the type of file  
 113381 they reference, by specifying the **-H** (for half logical) option. This option is intended to make the  
 113382 command line name space look like the logical name space.

113383 As consistently as possible, users may cause standard utilities performing a file hierarchy  
 113384 traversal to follow any symbolic links named on the command line as well as any symbolic links  
 113385 encountered during the traversal, regardless of the type of file they reference, by specifying the  
 113386 **-L** (for logical) option. This option is intended to make the entire name space look like the  
 113387 logical name space.

113388 For consistency, implementors are encouraged to use the **-P** (for “physical”) flag to specify the  
 113389 physical walk in utilities that do logical walks by default for whatever reason. The only standard  
 113390 utilities that require the **-P** option are *cd* and *pwd*; see the note below.

113391 When one or more of the **-H**, **-L**, and **-P** flags can be specified, the last one specified determines  
 113392 the behavior of the utility. This permits users to alias commands so that the default behavior is a  
 113393 logical walk and then override that behavior on the command line.

113394 *Exceptions in the Third and Fourth Domains*

113395 The *ls* and *rm* utilities are exceptions to these rules. The *rm* utility never follows symbolic links  
 113396 and does not support the **-H**, **-L**, or **-P** options. Some historical versions of *ls* always followed  
 113397 symbolic links given on the command line whether the **-L** option was specified or not.

113398 Historical versions of *ls* did not support the **-H** option. In POSIX.1-200x, unless one of the **-H** or  
113399 **-L** options is specified, the *ls* utility only follows symbolic links to directories that are given as  
113400 operands. The *ls* utility does not support the **-P** option.

113401 The Shell and Utilities volume of POSIX.1-200x requires that the standard utilities *ls*, *find*, and  
113402 *pax* detect infinite loops when doing logical walks; that is, a directory, or more commonly a  
113403 symbolic link, that refers to an ancestor in the current file hierarchy. If the file system itself is  
113404 corrupted, causing the infinite loop, it may be impossible to recover. Because *find* and *ls* are often  
113405 used in system administration and security applications, they should attempt to recover and  
113406 continue as best as they can. The *pax* utility should terminate because the archive it was creating  
113407 is by definition corrupted. Other, less vital, utilities should probably simply terminate as well.  
113408 Implementations are strongly encouraged to detect infinite loops in all utilities.

113409 Historical practice is shown in [Table A-1](#) (on page 3342). The heading **SVID3** stands for the  
113410 Third Edition of the System V Interface Definition.

113411 Historically, several shells have had built-in versions of the *pwd* utility. In some of these shells,  
113412 *pwd* reported the physical path, and in others, the logical path. Implementations of the shell  
113413 corresponding to POSIX.1-200x must report the logical path by default. Earlier versions of this  
113414 standard did not require the *pwd* utility to be a built-in utility. Now that *pwd* is required to set an  
113415 environment variable in the current shell execution environment, it must be a built-in utility.

113416 The *cd* command is required, by default, to treat the filename dot-dot logically. Implementors are  
113417 required to support the **-P** flag in *cd* so that users can have their current environment handled  
113418 physically. In 4.3 BSD, *chgrp* during tree traversal changed the group of the symbolic link, not  
113419 the target. Symbolic links in 4.4 BSD do not have *owner*, *group*, *mode*, or other standard UNIX  
113420 system file attributes.

113421 **Table A-1** Historical Practice for Symbolic Links

| Utility             | SVID3   | 4.3 BSD | 4.4 BSD | POSIX | Comments                                |
|---------------------|---------|---------|---------|-------|-----------------------------------------|
| 113422 <i>cd</i>    |         |         |         | -L    | Treat ". ." logically.                  |
| 113423 <i>cd</i>    |         |         |         | -P    | Treat ". ." physically.                 |
| 113424 <i>chgrp</i> |         |         | -H      | -H    | Follow command line symlinks.           |
| 113425 <i>chgrp</i> |         |         | -h      | -L    | Follow symlinks.                        |
| 113426 <i>chgrp</i> | -h      |         |         | -h    | Affect the symlink.                     |
| 113427 <i>chmod</i> |         |         |         |       | Affect the symlink.                     |
| 113428 <i>chmod</i> |         |         | -H      |       | Follow command line symlinks.           |
| 113429 <i>chmod</i> |         |         | -h      |       | Follow symlinks.                        |
| 113430 <i>chown</i> |         |         | -H      | -H    | Follow command line symlinks.           |
| 113431 <i>chown</i> |         |         | -h      | -L    | Follow symlinks.                        |
| 113432 <i>chown</i> | -h      |         |         | -h    | Affect the symlink.                     |
| 113433 <i>cp</i>    |         |         | -H      | -H    | Follow command line symlinks.           |
| 113434 <i>cp</i>    |         |         | -h      | -L    | Follow symlinks.                        |
| 113435 <i>cpio</i>  | -L      |         | -L      |       | Follow symlinks.                        |
| 113436 <i>du</i>    |         |         | -H      | -H    | Follow command line symlinks.           |
| 113437 <i>du</i>    |         |         | -h      | -L    | Follow symlinks.                        |
| 113438 <i>file</i>  | -h      |         |         | -h    | Affect the symlink.                     |
| 113439 <i>find</i>  |         |         | -H      | -H    | Follow command line symlinks.           |
| 113440 <i>find</i>  |         |         | -h      | -L    | Follow symlinks.                        |
| 113441 <i>find</i>  | -follow |         | -follow |       | Follow symlinks.                        |
| 113442 <i>ln</i>    | -s      | -s      | -s      | -s    | Create a symbolic link.                 |
| 113443 <i>ls</i>    | -L      | -L      | -L      | -L    | Follow symlinks.                        |
| 113444 <i>ls</i>    |         |         |         | -H    | Follow command line symlinks.           |
| 113445 <i>mv</i>    |         |         |         |       | Operates on the symlink.                |
| 113446 <i>pax</i>   |         |         | -H      | -H    | Follow command line symlinks.           |
| 113447 <i>pax</i>   |         |         | -h      | -L    | Follow symlinks.                        |
| 113448 <i>pwd</i>   |         |         |         | -L    | Printed path may contain symlinks.      |
| 113449 <i>pwd</i>   |         |         |         | -P    | Printed path will not contain symlinks. |
| 113450 <i>rm</i>    |         |         |         |       | Operates on the symlink.                |
| 113451 <i>tar</i>   |         |         | -H      |       | Follow command line symlinks.           |
| 113452 <i>tar</i>   |         | -h      | -h      |       | Follow symlinks.                        |
| 113453 <i>test</i>  | -h      |         | -h      | -h    | Affect the symlink.                     |

113455 **Synchronously-Generated Signal**

113456 Those signals that may be generated synchronously include SIGABRT, SIGBUS, SIGILL, SIGFPE,  
113457 SIGPIPE, and SIGSEGV.

113458 Any signal sent via the *raise()* function or a *kill()* function targeting the current process is also  
113459 considered synchronous.

113460 **System Call\***

113461 The distinction between a "system call" and a "library routine" is an implementation detail that  
113462 may differ between implementations and has thus been excluded from POSIX.1.

113463 See "Interface, Not Implementation" in the Preface.

- 113464           **System Console**
- 113465           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/7 is applied, changing from “An  
113466 implementation-defined device” to “A device”.
- 113467           **System Databases**
- 113468           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/9 is applied, rewording the definition to  
113469 reference the existing definitions for “group database” and “user database”.
- 113470           **System Process**
- 113471           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/8 is applied, rewording the definition to  
113472 remove the requirement for an implementation to define the object.
- 113473           **System Reboot**
- 113474           A “system reboot” is an event initiated by an unspecified circumstance that causes all processes  
113475 (other than special system processes) to be terminated in an implementation-defined manner,  
113476 after which any changes to the state and contents of files created or written to by a Conforming  
113477 POSIX.1 Application prior to the event are implementation-defined.
- 113478           IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/10 is applied, changing “An  
113479 implementation-defined sequence of events” to “An unspecified sequence of events”.
- 113480           **Synchronized I/O Data (and File) Integrity Completion**
- 113481           These terms specify that for synchronized read operations, pending writes must be successfully  
113482 completed before the read operation can complete. This is motivated by two circumstances.  
113483 Firstly, when synchronizing processes can access the same file, but not share common buffers  
113484 (such as for a remote file system), this requirement permits the reading process to guarantee that  
113485 it can read data written remotely. Secondly, having data written synchronously is insufficient to  
113486 guarantee the order with respect to a subsequent write by a reading process, and thus this extra  
113487 read semantic is necessary.
- 113488           **Text File**
- 113489           The term “text file” does not prevent the inclusion of control or other non-printable characters  
113490 (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either  
113491 able to process the special characters or they explicitly describe their limitations within their  
113492 individual descriptions. The definition of “text file” has caused controversy. The only difference  
113493 between text and binary files is that text files have lines of less than {LINE\_MAX} bytes, with no  
113494 NUL characters, each terminated by a <newline>. The definition allows a file with a single  
113495 <newline>, but not a totally empty file, to be called a text file. If a file ends with an incomplete  
113496 line it is not strictly a text file by this definition. The <newline> referred to in POSIX.1-200x is not  
113497 some generic line separator, but a single character; files created on systems where they use  
113498 multiple characters for ends of lines are not portable to all conforming systems without some  
113499 translation process unspecified by POSIX.1-200x.

113500 **Thread**

113501 POSIX.1-200x defines a thread to be a flow of control within a process. Each thread has a  
 113502 minimal amount of private state; most of the state associated with a process is shared among all  
 113503 of the threads in the process. While most multi-thread extensions to POSIX have taken this  
 113504 approach, others have made different decisions.

113505 **Note:** The choice to put threads within a process does not constrain implementations to implement  
 113506 threads in that manner. However, all functions have to behave as though threads share the  
 113507 indicated state information with the process from which they were created.

113508 Threads need to share resources in order to cooperate. Memory has to be widely shared between  
 113509 threads in order for the threads to cooperate at a fine level of granularity. Threads keep data  
 113510 structures and the locks protecting those data structures in shared memory. For a data structure  
 113511 to be usefully shared between threads, such structures should not refer to any data that can only  
 113512 be interpreted meaningfully by a single thread. Thus, any system resources that might be  
 113513 referred to in data structures need to be shared between all threads. File descriptors, pathnames,  
 113514 and pointers to stack variables are all things that programmers want to share between their  
 113515 threads. Thus, the file descriptor table, the root directory, the current working directory, and the  
 113516 address space have to be shared.

113517 Library implementations are possible as long as the effective behavior is as if system services  
 113518 invoked by one thread do not suspend other threads. This may be difficult for some library  
 113519 implementations on systems that do not provide asynchronous facilities.

113520 See [Section B.2.9](#) (on page 3463) for additional rationale.

113521 **Thread ID**

113522 See [Section B.2.9.2](#) (on page 3480) for additional rationale.

113523 **Thread-Safe Function**

113524 All functions required by POSIX.1-200x need to be thread-safe; see [Section A.4.17](#) (on page 3353)  
 113525 and [Section B.2.9.1](#) (on page 3477) for additional rationale.

113526 **User Database**

113527 There are no references in POSIX.1-200x to a “passwd file” or a “group file”, and there is no  
 113528 requirement that the *group* or *passwd* databases be kept in files containing editable text. Many  
 113529 large timesharing systems use *passwd* databases that are hashed for speed. Certain security  
 113530 classifications prohibit certain information in the *passwd* database from being publicly readable.

113531 The term “encoded” is used instead of “encrypted” in order to avoid the implementation  
 113532 connotations (such as reversibility or use of a particular algorithm) of the latter term.

113533 The *getgrent()*, *setgrent()*, *endgrent()*, *getpwent()*, *setpwent()*, and *endpwent()* functions are not  
 113534 included as part of the base standard because they provide a linear database search capability  
 113535 that is not generally useful (the *getpwuid()*, *getpwnam()*, *getgrgid()*, and *getgrnam()* functions are  
 113536 provided for keyed lookup) and because in certain distributed systems, especially those with  
 113537 different authentication domains, it may not be possible or desirable to provide an application  
 113538 with the ability to browse the system databases indiscriminately. They are provided on XSI-  
 113539 conformant systems due to their historical usage by many existing applications.

113540 A change from historical implementations is that the structures used by these functions have  
 113541 fields of the types **gid\_t** and **uid\_t**, which are required to be defined in the **<sys/types.h>** header.  
 113542 POSIX.1-200x requires implementations to ensure that these types are defined by inclusion of  
 113543 **<grp.h>** and **<pwd.h>**, respectively, without imposing any name space pollution or errors from  
 113544 redefinition of types.



113545 POSIX.1-200x is silent about the content of the strings containing user or group names. These  
 113546 could be digit strings. POSIX.1-200x is also silent as to whether such digit strings bear any  
 113547 relationship to the corresponding (numeric) user or group ID.

113548 *Database Access*

113549 The thread-safe versions of the user and group database access functions return values in user-  
 113550 supplied buffers instead of possibly using static data areas that may be overwritten by each call.

113551 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/11 is applied, removing the words “of  
 113552 implementation-defined format”.

113553 **Virtual Processor\***

113554 The term “virtual processor” was chosen as a neutral term describing all kernel-level  
 113555 schedulable entities, such as processes, Mach tasks, or lightweight processes. Implementing  
 113556 threads using multiple processes as virtual processors, or implementing multiplexed threads  
 113557 above a virtual processor layer, should be possible, provided some mechanism has also been  
 113558 implemented for sharing state between processes or virtual processors. Many systems may also  
 113559 wish to provide implementations of threads on systems providing “shared processes” or  
 113560 “variable-weight processes”. It was felt that exposing such implementation details would  
 113561 severely limit the type of systems upon which the threads interface could be supported and  
 113562 prevent certain types of valid implementations. It was also determined that a virtual processor  
 113563 interface was out of the scope of the Rationale (Informative) volume of POSIX.1-200x.

113564 **XSI**

113565 This is included to allow POSIX.1-200x to be adopted as an IEEE standard and an Open Group  
 113566 Technical Standard, serving both the POSIX and the Single UNIX Specification in a core set of  
 113567 volumes.

113568 The term “XSI” has been used for 10 years in connection with the XPG series and the first and  
 113569 second versions of the base volumes of the Single UNIX Specification. The XSI margin code was  
 113570 introduced to denote the extended or more restrictive semantics beyond POSIX that are  
 113571 applicable to UNIX systems.

## 113572 **A.4 General Concepts**

113573 The general concepts are similar in nature to the definitions section, with the exception that a  
 113574 term defined in general concepts can contain normative requirements.

### 113575 **A.4.1 Concurrent Execution**

113576 There is no additional rationale provided for this section.

### 113577 **A.4.2 Directory Protection**

113578 There is no additional rationale provided for this section.

### 113579 A.4.3 Extended Security Controls

113580 Allowing an implementation to define extended security controls enables the use of  
 113581 POSIX.1-200x in environments that require different or more rigorous security than that  
 113582 provided in POSIX.1. Extensions are allowed in two areas: privilege and file access permissions.  
 113583 The semantics of these areas have been defined to permit extensions with reasonable, but not  
 113584 exact, compatibility with all existing practices. For example, the elimination of the superuser  
 113585 definition precludes identifying a process as privileged or not by virtue of its effective user ID.

### 113586 A.4.4 File Access Permissions

113587 A process should not try to anticipate the result of an attempt to access data by *a priori* use of  
 113588 these rules. Rather, it should make the attempt to access data and examine the return value (and  
 113589 possibly *errno* as well), or use *access()*. An implementation may include other security  
 113590 mechanisms in addition to those specified in POSIX.1, and an access attempt may fail because of  
 113591 those additional mechanisms, even though it would succeed according to the rules given in this  
 113592 section. (For example, the user's security level might be lower than that of the object of the  
 113593 access attempt.) The supplementary group IDs provide another reason for a process to not  
 113594 attempt to anticipate the result of an access attempt.

113595 Since the current standard does not specify a method for opening a directory for searching, it is  
 113596 unspecified whether search permission on the *fd* argument to *openat()* and related functions is  
 113597 based on whether the directory was opened with search mode or on the current permissions  
 113598 allowed by the directory at the time a search is performed. When there is existing practice that  
 113599 supports opening directories for searching, it is expected that these functions will be modified to  
 113600 specify that the search permissions will be granted based on the file access modes of the  
 113601 directory's file descriptor identified by *fd*, and not on the mode of the directory at the time the  
 113602 directory is searched.

### 113603 A.4.5 File Hierarchy

113604 Though the file hierarchy is commonly regarded to be a tree, POSIX.1 does not define it as such  
 113605 for three reasons:

- 113606 1. Links may join branches.
- 113607 2. In some network implementations, there may be no single absolute root directory; see  
 113608 *pathname resolution*.
- 113609 3. With symbolic links, the file system need not be a tree or even a directed acyclic graph.

### 113610 A.4.6 Filenames

113611 Historically, certain filenames have been reserved. This list includes **core**, **/etc/passwd**, and so  
 113612 on. Conforming applications should avoid these.

113613 Most historical implementations prohibit case folding in filenames; that is, treating uppercase  
 113614 and lowercase alphabetic characters as identical. However, some consider case folding desirable:

- 113615 • For user convenience
- 113616 • For ease-of-implementation of the POSIX.1 interface as a hosted system on some popular  
 113617 operating systems

113618 Variants, such as maintaining case distinctions in filenames, but ignoring them in comparisons,  
 113619 have been suggested. Methods of allowing escaped characters of the case opposite the default  
 113620 have been proposed.

- 113621 Many reasons have been expressed for not allowing case folding, including:
- 113622 • No solid evidence has been produced as to whether case-sensitivity or case-insensitivity is  
113623 more convenient for users.
  - 113624 • Making case-insensitivity a POSIX.1 implementation option would be worse than either  
113625 having it or not having it, because:
    - 113626 — More confusion would be caused among users.
    - 113627 — Application developers would have to account for both cases in their code.
    - 113628 — POSIX.1 implementors would still have other problems with native file systems, such  
113629 as short or otherwise constrained filenames or pathnames, and the lack of  
113630 hierarchical directory structure.
  - 113631 • Case folding is not easily defined in many European languages, both because many of  
113632 them use characters outside the US ASCII alphabetic set, and because:
    - 113633 — In Spanish, the digraph "ll" is considered to be a single letter, the capitalized form  
113634 of which may be either "Ll" or "LL", depending on context.
    - 113635 — In French, the capitalized form of a letter with an accent may or may not retain the  
113636 accent, depending on the country in which it is written.
    - 113637 — In German, the sharp ess may be represented as a single character resembling a  
113638 Greek beta ( $\beta$ ) in lowercase, but as the digraph "SS" in uppercase.
    - 113639 — In Greek, there are several lowercase forms of some letters; the one to use depends on  
113640 its position in the word. Arabic has similar rules.
  - 113641 • Many East Asian languages, including Japanese, Chinese, and Korean, do not distinguish  
113642 case and are sometimes encoded in character sets that use more than one byte per  
113643 character.
  - 113644 • Multiple character codes may be used on the same machine simultaneously. There are  
113645 several ISO character sets for European alphabets. In Japan, several Japanese character  
113646 codes are commonly used together, sometimes even in filenames; this is evidently also the  
113647 case in China. To handle case insensitivity, the kernel would have to at least be able to  
113648 distinguish for which character sets the concept made sense.
  - 113649 • The file system implementation historically deals only with bytes, not with characters,  
113650 except for slash and the null byte.
  - 113651 • The purpose of POSIX.1 is to standardize the common, existing definition, not to change it.  
113652 Mandating case-insensitivity would make all historical implementations non-standard.
  - 113653 • Not only the interface, but also application programs would need to change, counter to the  
113654 purpose of having minimal changes to existing application code.
  - 113655 • At least one of the original developers of the UNIX system has expressed objection in the  
113656 strongest terms to either requiring case-insensitivity or making it an option, mostly on the  
113657 basis that POSIX.1 should not hinder portability of application programs across related  
113658 implementations in order to allow compatibility with unrelated operating systems.
- 113659 Two proposals were entertained regarding case folding in filenames:
- 113660 1. Remove all wording that previously permitted case folding.
- 113661 Rationale Case folding is inconsistent with portable filename character set definition  
113662 and filename definition (all characters except slash and null). No known  
113663 implementations allowing all characters except slash and null also do case

- 113664 folding.
- 113665 2. Change “though this practice is not recommended:” to “although this practice is strongly  
113666 discouraged.”
- 113667 Rationale If case folding must be included in POSIX.1, the wording should be stronger  
113668 to discourage the practice.
- 113669 The consensus selected the first proposal. Otherwise, a conforming application would have to  
113670 assume that case folding would occur when it was not wanted, but that it would not occur when  
113671 it was wanted.

#### 113672 **A.4.7 Filename Portability**

113673 Filenames should be constructed from the portable filename character set because the use of  
113674 other characters can be confusing or ambiguous in certain contexts. (For example, the use of a  
113675 colon ( ‘ : ’ ) in a pathname could cause ambiguity if that pathname were included in a *PATH*  
113676 definition.)

113677 The constraint on use of the hyphen character as the first character of a portable filename is a  
113678 constraint on application behavior and not on implementations, since applications might not  
113679 work as expected when such a filename is passed as a command line argument.

#### 113680 **A.4.8 File Times Update**

113681 This section reflects the actions of historical implementations. The times are not updated  
113682 immediately, but are only marked for update by the functions. An implementation may update  
113683 these times immediately.

113684 The accuracy of the time update values is intentionally left unspecified so that systems can  
113685 control the bandwidth of a possible covert channel.

113686 The wording was carefully chosen to make it clear that there is no requirement that the  
113687 conformance document contain information that might incidentally affect file timestamps. Any  
113688 function that performs pathname resolution might update several last data access timestamps. |  
113689 Functions such as *getpwnam()* and *getgrnam()* might update the last data access timestamp of |  
113690 some specific file or files. It is intended that these are not required to be documented in the |  
113691 conformance document, but they should appear in the system documentation.

#### 113692 **A.4.9 Host and Network Byte Order**

113693 There is no additional rationale provided for this section.

#### 113694 **A.4.10 Measurement of Execution Time**

113695 The methods used to measure the execution time of processes and threads, and the precision of  
113696 these measurements, may vary considerably depending on the software architecture of the  
113697 implementation, and on the underlying hardware. Implementations can also make tradeoffs  
113698 between the scheduling overhead and the precision of the execution time measurements.  
113699 POSIX.1-200x does not impose any requirement on the accuracy of the execution time; it instead  
113700 specifies that the measurement mechanism and its precision are implementation-defined.

113701 **A.4.11 Memory Synchronization**

113702 In older multi-processors, access to memory by the processors was strictly multiplexed. This  
 113703 meant that a processor executing program code interrogates or modifies memory in the order  
 113704 specified by the code and that all the memory operation of all the processors in the system  
 113705 appear to happen in some global order, though the operation histories of different processors are  
 113706 interleaved arbitrarily. The memory operations of such machines are said to be sequentially  
 113707 consistent. In this environment, threads can synchronize using ordinary memory operations. For  
 113708 example, a producer thread and a consumer thread can synchronize access to a circular data  
 113709 buffer as follows:

```
113710 int rdptr = 0;
113711 int wrptr = 0;
113712 data_t buf[BUFSIZE];

113713 Thread 1:
113714 while (work_to_do) {
113715 int next;
113716
113717 buf[wrptr] = produce();
113718 next = (wrptr + 1) % BUFSIZE;
113719 while (rdptr == next)
113720 ;
113721 wrptr = next;
113722 }

113723 Thread 2:
113724 while (work_to_do) {
113725 while (rdptr == wrptr)
113726 ;
113727 consume(buf[rdptr]);
113728 rdptr = (rdptr + 1) % BUFSIZE;
113729 }
```

113729 In modern multi-processors, these conditions are relaxed to achieve greater performance. If one  
 113730 processor stores values in location A and then location B, then other processors loading data  
 113731 from location B and then location A may see the new value of B but the old value of A. The  
 113732 memory operations of such machines are said to be weakly ordered. On these machines, the  
 113733 circular buffer technique shown in the example will fail because the consumer may see the new  
 113734 value of *wrptr* but the old value of the data in the buffer. In such machines, synchronization can  
 113735 only be achieved through the use of special instructions that enforce an order on memory  
 113736 operations. Most high-level language compilers only generate ordinary memory operations to  
 113737 take advantage of the increased performance. They usually cannot determine when memory  
 113738 operation order is important and generate the special ordering instructions. Instead, they rely on  
 113739 the programmer to use synchronization primitives correctly to ensure that modifications to a  
 113740 location in memory are ordered with respect to modifications and/or access to the same location  
 113741 in other threads. Access to read-only data need not be synchronized. The resulting program is  
 113742 said to be data race-free.

113743 Synchronization is still important even when accessing a single primitive variable (for example,  
 113744 an integer). On machines where the integer may not be aligned to the bus data width or be  
 113745 larger than the data width, a single memory load may require multiple memory cycles. This  
 113746 means that it may be possible for some parts of the integer to have an old value while other  
 113747 parts have a newer value. On some processor architectures this cannot happen, but portable  
 113748 programs cannot rely on this.

113749 In summary, a portable multi-threaded program, or a multi-process program that shares

113750 writable memory between processes, has to use the synchronization primitives to synchronize  
 113751 data access. It cannot rely on modifications to memory being observed by other threads in the  
 113752 order written in the application or even on modification of a single variable being seen  
 113753 atomically.

113754 Conforming applications may only use the functions listed to synchronize threads of control  
 113755 with respect to memory access. There are many other candidates for functions that might also be  
 113756 used. Examples are: signal sending and reception, or pipe writing and reading. In general, any  
 113757 function that allows one thread of control to wait for an action caused by another thread of  
 113758 control is a candidate. POSIX.1-200x does not require these additional functions to synchronize  
 113759 memory access since this would imply the following:

- 113760 • All these functions would have to be recognized by advanced compilation systems so that
- 113761 memory operations and calls to these functions are not reordered by optimization.
- 113762 • All these functions would potentially have to have memory synchronization instructions
- 113763 added, depending on the particular machine.
- 113764 • The additional functions complicate the model of how memory is synchronized and make
- 113765 automatic data race detection techniques impractical.

113766 Formal definitions of the memory model were rejected as unreadable by the vast majority of  
 113767 programmers. In addition, most of the formal work in the literature has concentrated on the  
 113768 memory as provided by the hardware as opposed to the application programmer through the  
 113769 compiler and runtime system. It was believed that a simple statement intuitive to most  
 113770 programmers would be most effective. POSIX.1-200x defines functions that can be used to  
 113771 synchronize access to memory, but it leaves open exactly how one relates those functions to the  
 113772 semantics of each function as specified elsewhere in POSIX.1-200x. POSIX.1-200x also does not  
 113773 make a formal specification of the partial ordering in time that the functions can impose, as that  
 113774 is implied in the description of the semantics of each function. It simply states that the  
 113775 programmer has to ensure that modifications do not occur “simultaneously” with other access  
 113776 to a memory location.

113777 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/4 is applied, adding a new paragraph  
 113778 beneath the table of functions: “The *pthread\_once()* function shall synchronize memory for the  
 113779 first call in each thread for a given **pthread\_once\_t** object.”.

#### 113780 A.4.12 Pathname Resolution

113781 It is necessary to differentiate between the definition of pathname and the concept of pathname  
 113782 resolution with respect to the handling of trailing slashes. By specifying the behavior here, it is  
 113783 not possible to provide an implementation that is conforming but extends all interfaces that  
 113784 handle pathnames to also handle strings that are not legal pathnames (because they have  
 113785 trailing slashes).

113786 Pathnames that end with one or more trailing slash characters must refer to directory paths. |  
 113787 Earlier versions of this standard were not specific about the distinction between trailing slashes |  
 113788 on files and directories, and both were permitted.

113789 Two types of implementation have been prevalent; those that ignored trailing slash characters  
 113790 on all pathnames regardless, and those that permitted them only on existing directories.

113791 POSIX.1-200x requires that a pathname with a trailing slash character be treated as if it had a  
 113792 trailing " / . " everywhere.

113793 Note that this change does not break any conforming applications; since there were two different  
 113794 types of implementation, no application could have portably depended on either behavior. This  
 113795 change does however require some implementations to be altered to remain compliant.

113796 Substantial discussion over a three-year period has shown that the benefits to application  
113797 developers outweighs the disadvantages for some vendors.

113798 On a historical note, some early applications automatically appended a `'/'` to every path.  
113799 Rather than fix the applications, the system implementation was modified to accept this  
113800 behavior by ignoring any trailing slash.

113801 Each directory has exactly one parent directory which is represented by the name **dot-dot** in the  
113802 first directory. No other directory, regardless of linkages established by symbolic links, is  
113803 considered the parent directory by POSIX.1-200x.

113804 There are two general categories of interfaces involving pathname resolution: those that follow  
113805 the symbolic link, and those that do not. There are several exceptions to this rule; for example,  
113806 `open(path,O_CREAT|O_EXCL)` will fail when `path` names a symbolic link. However, in all other  
113807 situations, the `open()` function will follow the link.

113808 What the filename **dot-dot** refers to relative to the root directory is implementation-defined. In  
113809 Version 7 it refers to the root directory itself; this is the behavior mentioned in POSIX.1-200x. In  
113810 some networked systems the construction `././hostname/` is used to refer to the root directory of  
113811 another host, and POSIX.1 permits this behavior.

113812 Other networked systems use the construct `//hostname` for the same purpose; that is, a double  
113813 initial slash is used. There is a potential problem with existing applications that create full  
113814 pathnames by taking a trunk and a relative pathname and making them into a single string  
113815 separated by `'/'`, because they can accidentally create networked pathnames when the trunk is  
113816 `'/'`. This practice is not prohibited because such applications can be made to conform by  
113817 simply changing to use `"/"` as a separator instead of `'/'`:

- 113818 • If the trunk is `'/'`, the full pathname will begin with `"/"` (the initial `'/'` and the  
113819 separator `"/"`). This is the same as `'/'`, which is what is desired. (This is the general  
113820 case of making a relative pathname into an absolute one by prefixing with `"/"` instead  
113821 of `'/'`.)
- 113822 • If the trunk is `"/A"`, the result is `"/A//..."`; since non-leading sequences of two or more  
113823 slashes are treated as a single slash, this is equivalent to the desired `"/A/..."`.
- 113824 • If the trunk is `"//A"`, the implementation-defined semantics will apply. (The multiple  
113825 slash rule would apply.)

113826 Application developers should avoid generating pathnames that start with `"/"`.  
113827 Implementations are strongly encouraged to avoid using this special interpretation since a  
113828 number of applications currently do not follow this practice and may inadvertently generate  
113829 `"//..."`.

113830 The term "root directory" is only defined in POSIX.1 relative to the process. In some  
113831 implementations, there may be no absolute root directory. The initialization of the root directory  
113832 of a process is implementation-defined.

#### 113833 A.4.13 Process ID Reuse

113834 There is no additional rationale provided for this section.

113835 **A.4.14 Scheduling Policy**

113836 There is no additional rationale provided for this section.

113837 **A.4.15 Seconds Since the Epoch**

113838 Coordinated Universal Time (UTC) includes leap seconds. However, in POSIX time (seconds  
 113839 since the Epoch), leap seconds are ignored (not applied) to provide an easy and compatible  
 113840 method of computing time differences. Broken-down POSIX time is therefore not necessarily  
 113841 UTC, despite its appearance.

113842 As of September 2000, 24 leap seconds had been added to UTC since the Epoch, 1 January, 1970.  
 113843 Historically, one leap second is added every 15 months on average, so this offset can be expected  
 113844 to grow steadily with time.

113845 Most systems' notion of "time" is that of a continuously increasing value, so this value should  
 113846 increase even during leap seconds. However, not only do most systems not keep track of leap  
 113847 seconds, but most systems are probably not synchronized to any standard time reference.  
 113848 Therefore, it is inappropriate to require that a time represented as seconds since the Epoch  
 113849 precisely represent the number of seconds between the referenced time and the Epoch.

113850 It is sufficient to require that applications be allowed to treat this time as if it represented the  
 113851 number of seconds between the referenced time and the Epoch. It is the responsibility of the  
 113852 vendor of the system, and the administrator of the system, to ensure that this value represents  
 113853 the number of seconds between the referenced time and the Epoch as closely as necessary for the  
 113854 application being run on that system.

113855 It is important that the interpretation of time names and seconds since the Epoch values be  
 113856 consistent across conforming systems; that is, it is important that all conforming systems  
 113857 interpret "536 457 599 seconds since the Epoch" as 59 seconds, 59 minutes, 23 hours 31 December  
 113858 1986, regardless of the accuracy of the system's idea of the current time. The expression is given  
 113859 to ensure a consistent interpretation, not to attempt to specify the calendar. The relationship  
 113860 between *tm\_yday* and the day of week, day of month, and month is in accordance with the  
 113861 Gregorian calendar, and so is not specified in POSIX.1.

113862 Consistent interpretation of seconds since the Epoch can be critical to certain types of distributed  
 113863 applications that rely on such timestamps to synchronize events. The accrual of leap seconds in a  
 113864 time standard is not predictable. The number of leap seconds since the Epoch will likely  
 113865 increase. POSIX.1 is more concerned about the synchronization of time between applications of  
 113866 astronomically short duration.

113867 Note that *tm\_yday* is zero-based, not one-based, so the day number in the example above is 364.  
 113868 Note also that the division is an integer division (discarding remainder) as in the C language.

113869 Note also that the meaning of *gmtime()*, *localtime()*, and *mkttime()* is specified in terms of this  
 113870 expression. However, the ISO C standard computes *tm\_yday* from *tm\_mday*, *tm\_mon*, and *tm\_year*  
 113871 in *mkttime()*. Because it is stated as a (bidirectional) relationship, not a function, and because the  
 113872 conversion between month-day-year and day-of-year dates is presumed well known and is also  
 113873 a relationship, this is not a problem.

113874 Implementations that implement **time\_t** as a signed 32-bit integer will overflow in 2038. The  
 113875 data size for **time\_t** is as per the ISO C standard definition, which is implementation-defined.

113876 See also [Epoch](#) (on page 3327).

113877 The topic of whether seconds since the Epoch should account for leap seconds has been debated  
 113878 on a number of occasions, and each time consensus was reached (with acknowledged dissent  
 113879 each time) that the majority of users are best served by treating all days identically. (That is, the



113880 majority of applications were judged to assume a single length—as measured in seconds since  
 113881 the Epoch—for all days. Thus, leap seconds are not applied to seconds since the Epoch.) Those  
 113882 applications which do care about leap seconds can determine how to handle them in whatever  
 113883 way those applications feel is best. This was particularly emphasized because there was  
 113884 disagreement about what the best way of handling leap seconds might be. It is a practical  
 113885 impossibility to mandate that a conforming implementation must have a fixed relationship to  
 113886 any particular official clock (consider isolated systems, or systems performing “reruns” by  
 113887 setting the clock to some arbitrary time).

113888 Note that as a practical consequence of this, the length of a second as measured by some external  
 113889 standard is not specified. This unspecified second is nominally equal to an International System  
 113890 (SI) second in duration. Applications must be matched to a system that provides the particular  
 113891 handling of external time in the way required by the application.

113892 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/12 is applied, making an editorial  
 113893 correction to the paragraph commencing “How any changes to the value of seconds ...”.

#### 113894 **A.4.16 Semaphore**

113895 There is no additional rationale provided for this section.

#### 113896 **A.4.17 Thread-Safety**

113897 Where the interface of a function required by POSIX.1-200x precludes thread-safety, an alternate  
 113898 thread-safe form is provided. The names of these thread-safe forms are the same as the non-  
 113899 thread-safe forms with the addition of the suffix “\_r”. The suffix “\_r” is historical, where the  
 113900 ‘r’ stood for “reentrant”.

113901 In some cases, thread-safety is provided by restricting the arguments to an existing function.

113902 See also [Section B.2.9.1](#) (on page 3477).

#### 113903 **A.4.18 Tracing**

113904 Refer to [Section B.2.11](#) (on page 3492).

#### 113905 **A.4.19 Treatment of Error Conditions for Mathematical Functions**

113906 There is no additional rationale provided for this section.

#### 113907 **A.4.20 Treatment of NaN Arguments for Mathematical Functions**

113908 There is no additional rationale provided for this section.

#### 113909 **A.4.21 Utility**

113910 There is no additional rationale provided for this section.

113911 **A.4.22 Variable Assignment**

113912 There is no additional rationale provided for this section.

113913 **A.5 File Format Notation**

113914 The notation for spaces allows some flexibility for application output. Note that an empty  
 113915 character position in *format* represents one or more <blank>s on the output (not *white space*,  
 113916 which can include <newline>s). Therefore, another utility that reads that output as its input  
 113917 must be prepared to parse the data using *scanf()*, *awk*, and so on. The 'Δ' character is used  
 113918 when exactly one <space> is output.

113919 The treatment of integers and spaces is different from the *printf()* function in that they can be  
 113920 surrounded with <blank>s. This was done so that, given a format such as:

113921 `"%d\n", <foo>`113922 the implementation could use a *printf()* call such as:113923 `printf("%6d\n", foo);`113924 and still conform. This notation is thus somewhat like *scanf()* in addition to *printf()*.

113925 The *printf()* function was chosen as a model because most of the standard developers were  
 113926 familiar with it. One difference from the C function *printf()* is that the *l* and *h* conversion  
 113927 specifier characters are not used. As expressed by the Shell and Utilities volume of  
 113928 POSIX.1-200x, there is no differentiation between decimal values for type **int**, type **long**, or type  
 113929 **short**. The conversion specifications `%d` or `%i` should be interpreted as an arbitrary length  
 113930 sequence of digits. Also, no distinction is made between single precision and double precision  
 113931 numbers (**float** or **double** in C). These are simply referred to as floating-point numbers.

113932 Many of the output descriptions in the Shell and Utilities volume of POSIX.1-200x use the term  
 113933 "line", such as:

113934 `"%s", <input line>`

113935 Since the definition of *line* includes the trailing <newline> already, there is no need to include a  
 113936 '\n' in the format; a double <newline> would otherwise result.

113937 **A.6 Character Set**113938 **A.6.1 Portable Character Set**

113939 The portable character set is listed in full so there is no dependency on the ISO/IEC 646: 1991  
 113940 standard (or historically ASCII) encoded character set, although the set is identical to the  
 113941 characters defined in the International Reference version of the ISO/IEC 646: 1991 standard.

113942 POSIX.1-200x poses no requirement that multiple character sets or codesets be supported,  
 113943 leaving this as a marketing differentiation for implementors. Although multiple charmap files  
 113944 are supported, it is the responsibility of the implementation to provide the file(s); if only one is  
 113945 provided, only that one will be accessible using the *localedef -f* option.

113946 The statement about invariance in codesets for the portable character set is worded to avoid  
 113947 precluding implementations where multiple incompatible codesets are available (for instance,  
 113948 ASCII and EBCDIC). The standard utilities cannot be expected to produce predictable results if

113949 they access portable characters that vary on the same implementation.

113950 Not all character sets need include the portable character set, but each locale must include it. For  
 113951 example, a Japanese-based locale might be supported by a mixture of character sets: JIS X 0201  
 113952 Roman (a Japanese version of the ISO/IEC 646:1991 standard), JIS X 0208, and JIS X 0201  
 113953 Katakana. Not all of these character sets include the portable characters, but at least one does  
 113954 (JIS X 0201 Roman).

## 113955 **A.6.2 Character Encoding**

113956 Encoding mechanisms based on single shifts, such as the EUC encoding used in some Asian and  
 113957 other countries, can be supported via the current charmap mechanism. With single-shift  
 113958 encoding, each character is preceded by a shift code (SS2 or SS3). A complete EUC code,  
 113959 consisting of the portable character set (G0) and up to three additional character sets (G1, G2,  
 113960 G3), can be described using the current charmap mechanism; the encoding for each character in  
 113961 additional character sets G2 and G3 must then include their single-shift code. Other mechanisms  
 113962 to support locales based on encoding mechanisms such as locking shift are not addressed by this  
 113963 volume of POSIX.1-200x.

## 113964 **A.6.3 C Language Wide-Character Codes**

113965 The standard does not specify how wide characters are encoded or provide a method for  
 113966 defining wide characters in a charmap. It specifies ways of translating between wide characters  
 113967 and multi-byte characters. The standard does not prevent an extension from providing a method  
 113968 to define wide characters.

113969 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/13 is applied, adding a statement that the  
 113970 standard has no means of defining a wide-character codeset.

## 113971 **A.6.4 Character Set Description File**

113972 IEEE PASC Interpretation 1003.2 #196 is applied, removing three lines of text dealing with  
 113973 ranges of symbolic names using position constant values which had been erroneously included  
 113974 in the final IEEE P1003.2b draft standard.

113975 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/14 is applied, correcting the example and  
 113976 adding a statement that the standard provides no means of defining a wide-character codeset.

113977 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/15 is applied, allowing the value zero for  
 113978 the width value of **WIDTH** and **WIDTH\_DEFAULT**. This is required to cover some existing  
 113979 locales.

### 113980 **A.6.4.1 State-Dependent Character Encodings**

113981 A requirement was considered that would force utilities to eliminate any redundant locking  
 113982 shifts, but this was left as a quality of implementation issue.

113983 This change satisfies the following requirement from the ISO POSIX-2:1993 standard, Annex  
 113984 H.1:

113985 *The support of state-dependent (shift encoding) character sets should be addressed fully. See |*  
 113986 *descriptions of these in XBD Section 6.2 (on page 114). If such character encodings are supported, |*  
 113987 *it is expected that this will impact XBD Section 6.2 (on page 114), Chapter 7 (on page 121), |*  
 113988 *Chapter 9 (on page 167), and the comm, cut, diff, grep, head, join, paste, and tail utilities.*

113989 The character set description file provides:

113990 • The capability to describe character set attributes (such as collation order or character  
 113991 classes) independent of character set encoding, and using only the characters in the  
 113992 portable character set. This makes it possible to create generic *localedef* source files for all  
 113993 codesets that share the portable character set (such as the ISO 8859 family or IBM Extended  
 113994 ASCII).

113995 • Standardized symbolic names for all characters in the portable character set, making it  
 113996 possible to refer to any such character regardless of encoding.

113997 Implementations are free to choose their own symbolic names, as long as the names identified  
 113998 by the Base Definitions volume of POSIX.1-200x are also defined; this provides support for  
 113999 already existing “character names”.

114000 The names selected for the members of the portable character set follow the  
 114001 ISO/IEC 8859-1:1998 standard and the ISO/IEC 10646-1:2000 standard. However, several  
 114002 commonly used UNIX system names occur as synonyms in the list:

- 114003 • The historical UNIX system names are used for control characters.
- 114004 • The word “slash” is given in addition to “solidus”.
- 114005 • The word “backslash” is given in addition to “reverse-solidus”.
- 114006 • The word “hyphen” is given in addition to “hyphen-minus”.
- 114007 • The word “period” is given in addition to “full-stop”.
- 114008 • For digits, the word “digit” is eliminated.
- 114009 • For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
- 114010 • The words “left brace” and “right brace” are given in addition to “left-curly-bracket” and  
 114011 “right-curly-bracket”.
- 114012 • The names of the digits are preferred over the numbers to avoid possible confusion  
 114013 between ‘0’ and ‘O’, and between ‘1’ and ‘l’ (one and the letter ell).

114014 The names for the control characters in XBD [Chapter 6](#) (on page 111) were taken from the  
 114015 ISO/IEC 4873:1991 standard.

114016 The charmap file was introduced to resolve problems with the portability of, especially, *localedef*  
 114017 sources. POSIX.1-200x assumes that the portable character set is constant across all locales, but  
 114018 does not prohibit implementations from supporting two incompatible codings, such as both  
 114019 ASCII and EBCDIC. Such dual-support implementations should have all charmaps and *localedef*  
 114020 sources encoded using one portable character set, in effect cross-compiling for the other  
 114021 environment. Naturally, charmaps (and *localedef* sources) are only portable without  
 114022 transformation between systems using the same encodings for the portable character set. They  
 114023 can, however, be transformed between two sets using only a subset of the actual characters (the  
 114024 portable character set). However, the particular coded character set used for an application or an  
 114025 implementation does not necessarily imply different characteristics or collation; on the contrary,  
 114026 these attributes should in many cases be identical, regardless of codeset. The charmap provides  
 114027 the capability to define a common locale definition for multiple codesets (the same *localedef*  
 114028 source can be used for codesets with different extended characters; the ability in the charmap to  
 114029 define empty names allows for characters missing in certain codesets).

114030 The `<escape_char>` declaration was added at the request of the international community to ease  
 114031 the creation of portable charmap files on terminals not implementing the default backslash  
 114032 escape. The `<comment_char>` declaration was added at the request of the international  
 114033 community to eliminate the potential confusion between the number sign and the pound sign.

114034 The octal number notation with no leading zero required was selected to match those of *awk* and

114035 *tr* and is consistent with that used by *localedef*. To avoid confusion between an octal constant and  
 114036 the back-references used in *localedef* source, the octal, hexadecimal, and decimal constants must  
 114037 contain at least two digits. As single-digit constants are relatively rare, this should not impose  
 114038 any significant hardship. Provision is made for more digits to account for systems in which the  
 114039 byte size is larger than 8 bits. For example, a Unicode (ISO/IEC 10646-1:2000 standard) system  
 114040 that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.

114041 The decimal notation is supported because some newer international standards define character  
 114042 values in decimal, rather than in the old column/row notation.

114043 The charmap identifies the coded character sets supported by an implementation. At least one  
 114044 charmap must be provided, but no implementation is required to provide more than one.  
 114045 Likewise, implementations can allow users to generate new charmaps (for instance, for a new  
 114046 version of the ISO 8859 family of coded character sets), but does not have to do so. If users are  
 114047 allowed to create new charmaps, the system documentation describes the rules that apply (for  
 114048 instance, “only coded character sets that are supersets of the ISO/IEC 646:1991 standard IRV, no  
 114049 multi-byte characters”).

114050 This addition of the **WIDTH** specification satisfies the following requirement from the  
 114051 ISO POSIX-2:1993 standard, Annex H.1:

114052 (9) *The definition of column position relies on the implementation’s knowledge of the integral*  
 114053 *width of the characters. The charmap or LC\_CTYPE locale definitions should be enhanced to*  
 114054 *allow application specification of these widths.*

114055 The character “width” information was first considered for inclusion under *LC\_CTYPE* but was  
 114056 moved because it is more closely associated with the information in the charmap than  
 114057 information in the locale source (cultural conventions information). Concerns were raised that  
 114058 formalizing this type of information is moving the locale source definition from the codeset-  
 114059 independent entity that it was designed to be to a repository of codeset-specific information. A  
 114060 similar issue occurred with the `<code_set_name>`, `<mb_cur_max>`, and `<mb_cur_min>`  
 114061 information, which was resolved to reside in the charmap definition.

114062 The width definition was added to the IEEE P1003.2b draft standard with the intent that the  
 114063 *wcswidth()* and/or *wcwidth()* functions (currently specified in the System Interfaces volume of  
 114064 POSIX.1-200x) be the mechanism to retrieve the character width information.

## 114065 **A.7 Locale**

### 114066 **A.7.1 General**

114067 The description of locales is based on work performed in the UniForum Technical Committee,  
 114068 Subcommittee on Internationalization. Wherever appropriate, keywords are taken from the  
 114069 ISO C standard or the X/Open Portability Guide.

114070 The value used to specify a locale with environment variables is the name specified as the *name*  
 114071 operand to the *localedef* utility when the locale was created. This provides a verifiable method to  
 114072 create and invoke a locale.

114073 The “object” definitions need not be portable, as long as “source” definitions are. Strictly  
 114074 speaking, source definitions are portable only between implementations using the same  
 114075 character set(s). Such source definitions, if they use symbolic names only, easily can be ported  
 114076 between systems using different codesets, as long as the characters in the portable character set

114077 (see XBD Section 6.1, on page 111) have common values between the codesets; this is frequently |  
 114078 the case in historical implementations. Of source, this requires that the symbolic names used for |  
 114079 characters outside the portable character set be identical between character sets. The definition |  
 114080 of symbolic names for characters is outside the scope of POSIX.1-200x, but is certainly within the |  
 114081 scope of other standards organizations.

114082 Applications can select the desired locale by invoking the *setlocale()* function (or equivalent) |  
 114083 with the appropriate value. If the function is invoked with an empty string, the value of the |  
 114084 corresponding environment variable is used. If the environment variable is not set or is set to the |  
 114085 empty string, the implementation sets the appropriate environment as defined in XBD Chapter 8 |  
 114086 (on page 159).

## 114087 A.7.2 POSIX Locale

114088 The POSIX locale is equal to the C locale. To avoid being classified as a C-language function, the |  
 114089 name has been changed to the POSIX locale; the environment variable value can be either |  
 114090 "POSIX" or, for historical reasons, "C".

114091 The POSIX definitions mirror the historical UNIX system behavior.

114092 The use of symbolic names for characters in the tables does not imply that the POSIX locale must |  
 114093 be described using symbolic character names, but merely that it may be advantageous to do so.

## 114094 A.7.3 Locale Definition

114095 The decision to separate the file format from the *localedef* utility description was only partially |  
 114096 editorial. Implementations may provide other interfaces than *localedef*. Requirements on "the |  
 114097 utility", mostly concerning error messages, are described in this way because they are meant to |  
 114098 affect the other interfaces implementations may provide as well as *localedef*.

114099 The text about POSIX2\_LOCALEDEF does not mean that internationalization is optional; only |  
 114100 that the functionality of the *localedef* utility is. REs, for instance, must still be able to recognize, |  
 114101 for example, character class expressions such as "[[:alpha:]]". A possible analogy is with |  
 114102 an applications development environment; while all conforming implementations must be |  
 114103 capable of executing applications, not all need to have the development environment installed. |  
 114104 The assumption is that the capability to modify the behavior of utilities (and applications) via |  
 114105 locale settings must be supported. If the *localedef* utility is not present, then the only choice is to |  
 114106 select an existing (presumably implementation-documented) locale. An implementation could, |  
 114107 for example, choose to support only the POSIX locale, which would in effect limit the amount of |  
 114108 changes from historical implementations quite drastically. The *localedef* utility is still required, |  
 114109 but would always terminate with an exit code indicating that no locale could be created. |  
 114110 Supported locales must be documented using the syntax defined in this chapter. (This ensures |  
 114111 that users can accurately determine what capabilities are provided. If the implementation |  
 114112 decides to provide additional capabilities to the ones in this chapter, that is already provided |  
 114113 for.)

114114 If the option is present (that is, locales can be created), then the *localedef* utility must be capable |  
 114115 of creating locales based on the syntax and rules defined in this chapter. This does not mean that |  
 114116 the implementation cannot also provide alternate means for creating locales.

114117 The octal, decimal, and hexadecimal notations are the same employed by the charmap facility |  
 114118 (see XBD Section 6.4, on page 115). To avoid confusion between an octal constant and a back- |  
 114119 reference, the octal, hexadecimal, and decimal constants must contain at least two digits. As |  
 114120 single-digit constants are relatively rare, this should not impose any significant hardship. |  
 114121 Provision is made for more digits to account for systems in which the byte size is larger than 8 |  
 114122 bits. For example, a Unicode (see the ISO/IEC 10646-1:2000 standard) system that has defined |

114123 16-bit bytes may require six octal, four hexadecimal, and five decimal digits. As with the  
 114124 charmap file, multi-byte characters are described in the locale definition file using “big-endian”  
 114125 notation for reasons of portability. There is no requirement that the internal representation in the  
 114126 computer memory be in this same order.

114127 One of the guidelines used for the development of this volume of POSIX.1-200x is that  
 114128 characters outside the invariant part of the ISO/IEC 646:1991 standard should not be used in  
 114129 portable specifications. The backslash character is not in the invariant part; the number sign is,  
 114130 but with multiple representations: as a number sign, and as a pound sign. As far as general  
 114131 usage of these symbols, they are covered by the “grandfather clause”, but for newly defined  
 114132 interfaces, the WG15 POSIX working group has requested that POSIX provide alternate  
 114133 representations. Consequently, while the default escape character remains the backslash and the  
 114134 default comment character is the number sign, implementations are required to recognize  
 114135 alternative representations, identified in the applicable source file via the `<escape_char>` and  
 114136 `<comment_char>` keywords.

### 114137 A.7.3.1 *LC\_CTYPE*

114138 The *LC\_CTYPE* category is primarily used to define the encoding-independent aspects of a  
 114139 character set, such as character classification. In addition, certain encoding-dependent  
 114140 characteristics are also defined for an application via the *LC\_CTYPE* category. POSIX.1-200x  
 114141 does not mandate that the encoding used in the locale is the same as the one used by the  
 114142 application because an implementation may decide that it is advantageous to define locales in a  
 114143 system-wide encoding rather than having multiple, logically identical locales in different  
 114144 encodings, and to convert from the application encoding to the system-wide encoding on usage.  
 114145 Other implementations could require encoding-dependent locales.

114146 In either case, the *LC\_CTYPE* attributes that are directly dependent on the encoding, such as  
 114147 `<mb_cur_max>` and the display width of characters, are not user-specifiable in a locale source  
 114148 and are consequently not defined as keywords.

114149 Implementations may define additional keywords or extend the *LC\_CTYPE* mechanism to allow  
 114150 application-defined keywords.

114151 The text “The ellipsis specification shall only be valid within a single encoded character set” is  
 114152 present because it is possible to have a locale supported by multiple character encodings, as  
 114153 explained in the rationale for XBD [Section 6.1](#) (on page 111). An example given there is of a  
 114154 possible Japanese-based locale supported by a mixture of the character sets JIS X 0201 Roman,  
 114155 JIS X 0208, and JIS X 0201 Katakana. Attempting to express a range of characters across these sets  
 114156 is not logical and the implementation is free to reject such attempts.

114157 As the *LC\_CTYPE* character classes are based on the ISO C standard character class definition,  
 114158 the category does not support multi-character elements. For instance, the German character  
 114159 `<sharp-s>` is traditionally classified as a lowercase letter. There is no corresponding uppercase  
 114160 letter; in proper capitalization of German text, the `<sharp-s>` will be replaced by “SS”; that is, by  
 114161 two characters. This kind of conversion is outside the scope of the **toupper** and **tolower**  
 114162 keywords.

114163 Where POSIX.1-200x specifies that only certain characters can be specified, as for the keywords  
 114164 **digit** and **xdigit**, the specified characters must be from the portable character set, as shown. As  
 114165 an example, only the Arabic digits 0 through 9 are acceptable as digits.

114166 The character classes **digit**, **xdigit**, **lower**, **upper**, and **space** have a set of automatically included  
 114167 characters. These only need to be specified if the character values (that is, encoding) differs from  
 114168 the implementation default values. It is not possible to define a locale without these  
 114169 automatically included characters unless some implementation extension is used to prevent  
 114170 their inclusion. Such a definition would not be a proper superset of the C locale, and thus, it

114171 might not be possible for the standard utilities to be implemented as programs conforming to  
114172 the ISO C standard.

114173 The definition of character class **digit** requires that only ten characters—the ones defining  
114174 digits—can be specified; alternate digits (for example, Hindi or Kanji) cannot be specified here.  
114175 However, the encoding may vary if an implementation supports more than one encoding.

114176 The definition of character class **xdigit** requires that the characters included in character class  
114177 **digit** are included here also and allows for different symbols for the hexadecimal digits 10  
114178 through 15.

114179 The inclusion of the **charclass** keyword satisfies the following requirement from the  
114180 ISO POSIX-2: 1993 standard, Annex H.1:

114181 (3) *The LC\_CTYPE (2.5.2.1) locale definition should be enhanced to allow user-specified additional*  
114182 *character classes, similar in concept to the ISO C standard Multibyte Support Extension (MSE)*  
114183 *iswctype() function.*

114184 This keyword was previously included in The Open Group specifications and is now mandated  
114185 in the Shell and Utilities volume of POSIX.1-200x.

114186 The symbolic constant {CHARCLASS\_NAME\_MAX} was also adopted from The Open Group  
114187 specifications. Applications portability is enhanced by the use of symbolic constants.

#### 114188 A.7.3.2 LC\_COLLATE

114189 The rules governing collation depend to some extent on the use. At least five different levels of  
114190 increasingly complex collation rules can be distinguished:

- 114191 1. *Byte/machine code order*: This is the historical collation order in the UNIX system and many  
114192 proprietary operating systems. Collation is here performed character by character,  
114193 without any regard to context. The primary virtue is that it usually is quite fast and also  
114194 completely deterministic; it works well when the native machine collation sequence  
114195 matches the user expectations.
- 114196 2. *Character order*: On this level, collation is also performed character by character, without  
114197 regard to context. The order between characters is, however, not determined by the code  
114198 values, but on the expectations by the user of the “correct” order between characters. In  
114199 addition, such a (simple) collation order can specify that certain characters collate equally  
114200 (for example, uppercase and lowercase letters).
- 114201 3. *String ordering*: On this level, entire strings are compared based on relatively  
114202 straightforward rules. Several “passes” may be required to determine the order between  
114203 two strings. Characters may be ignored in some passes, but not in others; the strings may  
114204 be compared in different directions; and simple string substitutions may be performed  
114205 before strings are compared. This level is best described as “dictionary” ordering; it is  
114206 based on the spelling, not the pronunciation, or meaning, of the words.
- 114207 4. *Text search ordering*: This is a further refinement of the previous level, best described as  
114208 “telephone book ordering”; some common homonyms (words spelled differently but  
114209 with the same pronunciation) are collated together; numbers are collated as if they were  
114210 spelled out, and so on.
- 114211 5. *Semantic-level ordering*: Words and strings are collated based on their meaning; entire  
114212 words (such as “the”) are eliminated; the ordering is not deterministic. This usually  
114213 requires special software and is highly dependent on the intended use.

114214 While the historical collation order formally is at level 1, for the English language it corresponds



114215 roughly to elements at level 2. The user expects to see the output from the *ls* utility sorted very  
 114216 much as it would be in a dictionary. While telephone book ordering would be an optimal goal  
 114217 for standard collation, this was ruled out as the order would be language-dependent.  
 114218 Furthermore, a requirement was that the order must be determined solely from the text string  
 114219 and the collation rules; no external information (for example, “pronunciation dictionaries”)  
 114220 could be required.

114221 As a result, the goal for the collation support is at level 3. This also matches the requirements for  
 114222 the Canadian collation order, as well as other, known collation requirements for alphabetic  
 114223 scripts. It specifically rules out collation based on pronunciation rules or based on semantic  
 114224 analysis of the text.

114225 The syntax for the *LC\_COLLATE* category source meets the requirements for level 3 and has  
 114226 been verified to produce the correct result with examples based on French, Canadian, and  
 114227 Danish collation order. Because it supports multi-character collating elements, it is also capable  
 114228 of supporting collation in codesets where a character is expressed using non-spacing characters  
 114229 followed by the base character (such as the ISO/IEC 6937:2001 standard).

114230 The directives that can be specified in an operand to the **order\_start** keyword are based on the  
 114231 requirements specified in several proposed standards and in customary use. The following is a  
 114232 rephrasing of rules defined for “lexical ordering in English and French” by the Canadian  
 114233 Standards Association (the text in square brackets is rephrased):

- 114234 • Once special characters [punctuation] have been removed from original strings, the  
 114235 ordering is determined by scanning forwards (left to right) [disregarding case and  
 114236 diacriticals].
- 114237 • In case of equivalence, special characters are once again removed from original strings and  
 114238 the ordering is determined by scanning backwards (starting from the rightmost character  
 114239 of the string and back), character by character [disregarding case but considering  
 114240 diacriticals].
- 114241 • In case of repeated equivalence, special characters are removed again from original strings  
 114242 and the ordering is determined by scanning forwards, character by character [considering  
 114243 both case and diacriticals].
- 114244 • If there is still an ordering equivalence after the first three rules have been applied, then  
 114245 only special characters and the position they occupy in the string are considered to  
 114246 determine ordering. The string that has a special character in the lowest position comes  
 114247 first. If two strings have a special character in the same position, the character [with the  
 114248 lowest collation value] comes first. In case of equality, the other special characters are  
 114249 considered until there is a difference or until all special characters have been exhausted.

114250 It is estimated that this part of POSIX.1-200x covers the requirements for all European  
 114251 languages, and no particular problems are anticipated with Slavic or Middle East character sets.

114252 The Far East (particularly Japanese/Chinese) collations are often based on contextual  
 114253 information and pronunciation rules (the same ideogram can have different meanings and  
 114254 different pronunciations). Such collation, in general, falls outside the desired goal of  
 114255 POSIX.1-200x. There are, however, several other collation rules (stroke/radical or “most  
 114256 common pronunciation”) that can be supported with the mechanism described here.

114257 The character order is defined by the order in which characters and elements are specified  
 114258 between the **order\_start** and **order\_end** keywords. Weights assigned to the characters and  
 114259 elements define the collation sequence; in the absence of weights, the character order is also the  
 114260 collation sequence.

114261 The **position** keyword provides the capability to consider, in a compare, the relative position of

114262 characters not subject to **IGNORE**. As an example, consider the two strings "o-ring" and  
 114263 "or-ing". Assuming the hyphen is subject to **IGNORE** on the first pass, the two strings  
 114264 compare equal, and the position of the hyphen is immaterial. On second pass, all characters  
 114265 except the hyphen are subject to **IGNORE**, and in the normal case the two strings would again  
 114266 compare equal. By taking position into account, the first collates before the second.

### 114267 A.7.3.3 LC\_MONETARY

114268 The currency symbol does not appear in *LC\_MONETARY* because it is not defined in the C  
 114269 locale of the ISO C standard.

114270 The ISO C standard limits the size of decimal points and thousands delimiters to single-byte  
 114271 values. In locales based on multi-byte coded character sets, this cannot be enforced;  
 114272 POSIX.1-200x does not prohibit such characters, but makes the behavior unspecified (in the text  
 114273 "In contexts where other standards ...").

114274 The grouping specification is based on, but not identical to, the ISO C standard. The -1 indicates  
 114275 that no further grouping is performed; the equivalent of {CHAR\_MAX} in the ISO C standard.

114276 The text "the value is not available in the locale" is taken from the ISO C standard and is used  
 114277 instead of the "unspecified" text in early proposals. There is no implication that omitting these  
 114278 keywords or assigning them values of " " or -1 produces unspecified results; such omissions or  
 114279 assignments eliminate the effects described for the keyword or produce zero-length strings, as  
 114280 appropriate.

114281 The locale definition is an extension of the ISO C standard *localeconv()* specification. In  
 114282 particular, rules on how **currency\_symbol** is treated are extended to also cover **int\_curr\_symbol**,  
 114283 and **p\_sep\_by\_space** and **n\_sep\_by\_space** have been augmented with the value 2, which places  
 114284 a <space> between the sign and the symbol. This has been updated to match the  
 114285 ISO/IEC 9899:1999 standard requirements and is an incompatible change from UNIX 98 and the  
 114286 ISO POSIX-2 standard and the ISO POSIX-1:1996 standard requirements. The following table  
 114287 shows the result of various combinations:

|                   |                 | p_sep_by_space |           |          |
|-------------------|-----------------|----------------|-----------|----------|
|                   |                 | 2              | 1         | 0        |
| p_cs_precedes = 1 | p_sign_posn = 0 | (\$1,25)       | (\$ 1.25) | (\$1.25) |
|                   | p_sign_posn = 1 | + \$1.25       | +\$ 1.25  | +\$1.25  |
|                   | p_sign_posn = 2 | \$1.25 +       | \$ 1.25+  | \$1.25+  |
|                   | p_sign_posn = 3 | + \$1.25       | +\$ 1.25  | +\$1.25  |
|                   | p_sign_posn = 4 | \$ +1.25       | +\$ 1.25  | +\$1.25  |
| p_cs_precedes = 0 | p_sign_posn = 0 | (1.25 \$)      | (1.25 \$) | (1.25\$) |
|                   | p_sign_posn = 1 | +1.25 \$       | +1.25 \$  | +1.25\$  |
|                   | p_sign_posn = 2 | 1.25\$ +       | 1.25 \$+  | 1.25\$+  |
|                   | p_sign_posn = 3 | 1.25+ \$       | 1.25 +\$  | 1.25+\$  |
|                   | p_sign_posn = 4 | 1.25\$ +       | 1.25 \$+  | 1.25\$+  |

114300 The following is an example of the interpretation of the **mon\_grouping** keyword. Assuming that  
 114301 the value to be formatted is 123 456 789 and the **mon\_thousands\_sep** is ' ', then the following  
 114302 table shows the result. The third column shows the equivalent string in the ISO C standard that  
 114303 would be used by the *localeconv()* function to accommodate this grouping.

|        | mon_grouping | Formatted Value | ISO C String |
|--------|--------------|-----------------|--------------|
| 114304 | 3;-1         | 123456'789      | "\3\177"     |
| 114305 | 3            | 123'456'789     | "\3"         |
| 114306 | 3;2;-1       | 1234'56'789     | "\3\2\177"   |
| 114307 | 3;2          | 12'34'56'789    | "\3\2"       |
| 114308 | -1           | 123456789       | "\177"       |

114310 In these examples, the octal value of {CHAR\_MAX} is 177.

114311 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/6 adds a correction that permits the Euro  
114312 currency symbol and addresses extensibility. The correction is stated using the term "should"  
114313 intentionally, in order to make this a recommendation rather than a restriction on  
114314 implementations. This allows for flexibility in implementations on how they handle future  
114315 currency symbol additions.

114316 IEEE Std 1003.1-2001/Cor 1-2002, tem XBD/TC1/D6/5 is applied, adding the `int_[np]_*` values  
114317 to the POSIX locale definition of `LC_MONETARY`.

114318 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/16 is applied, updating the descriptions  
114319 of `p_sep_by_space`, `n_sep_by_space`, `int_p_sep_by_space`, and `int_n_sep_by_space` to match  
114320 the description of these keywords in the ISO C standard and the System Interfaces volume of  
114321 POSIX.1-200x, `localeconv()`.

#### 114322 A.7.3.4 `LC_NUMERIC`

114323 See the rationale for `LC_MONETARY` for a description of the behavior of grouping.

#### 114324 A.7.3.5 `LC_TIME`

114325 Although certain of the conversion specifications in the POSIX locale (such as the name of the  
114326 month) are shown with initial capital letters, this need not be the case in other locales. Programs  
114327 using these conversion specifications may need to adjust the capitalization if the output is going  
114328 to be used at the beginning of a sentence.

114329 The `LC_TIME` descriptions of `abday`, `day`, `mon`, and `abmon` imply a Gregorian style calendar  
114330 (7-day weeks, 12-month years, leap years, and so on). Formatting time strings for other types of  
114331 calendars is outside the scope of POSIX.1-200x.

114332 While the ISO 8601:2000 standard numbers the weekdays starting with Monday, historical  
114333 practice is to use the Sunday as the first day. Rather than change the order and introduce  
114334 potential confusion, the days must be specified beginning with Sunday; previous references to  
114335 "first day" have been removed. Note also that the Shell and Utilities volume of POSIX.1-200x  
114336 `date` utility supports numbering compliant with the ISO 8601:2000 standard.

114337 As specified under `date` in the Shell and Utilities volume of POSIX.1-200x and `strftime()` in the  
114338 System Interfaces volume of POSIX.1-200x, the conversion specifications corresponding to the  
114339 optional keywords consist of a modifier followed by a traditional conversion specification (for  
114340 instance, `%Ex`). If the optional keywords are not supported by the implementation or are  
114341 unspecified for the current locale, these modified conversion specifications are treated as the  
114342 traditional conversion specifications. For example, assume the following keywords:

```
114343 alt_digits "0th"; "1st"; "2nd"; "3rd"; "4th"; "5th"; \
114344 "6th"; "7th"; "8th"; "9th"; "10th"

114345 d_fmt "The %Od day of %B in %Y"
```

114346 On July 4th 1776, the `%x` conversion specifications would result in "The 4th day of July  
114347 in 1776", while on July 14th 1789 it would result in "The 14 day of July in 1789". It

114348 can be noted that the above example is for illustrative purposes only; the %O modifier is  
114349 primarily intended to provide for Kanji or Hindi digits in *date* formats.

114350 The following is an example for Japan that supports the current plus last three Emperors and  
114351 reverts to Western style numbering for years prior to the Meiji era. The example also allows for  
114352 the custom of using a special name for the first year of an era instead of using 1. (The examples  
114353 substitute romaji where kanji should be used.)

```
114354 era_d_fmt "%EY%mgatsu%dnichi (%a)"
114355 era "+:2:1990/01/01:+*:Heisei:%EC%Eynen";\
114356 "+:1:1989/01/08:1989/12/31:Heisei:%ECgannen";\
114357 "+:2:1927/01/01:1989/01/07:Shouwa:%EC%Eynen";\
114358 "+:1:1926/12/25:1926/12/31:Shouwa:%ECgannen";\
114359 "+:2:1913/01/01:1926/12/24:Taishou:%EC%Eynen";\
114360 "+:1:1912/07/30:1912/12/31:Taishou:%ECgannen";\
114361 "+:2:1869/01/01:1912/07/29:Meiji:%EC%Eynen";\
114362 "+:1:1868/09/08:1868/12/31:Meiji:%ECgannen";\
114363 "-:1868:1868/09/07:-*::%Ey"
```

114364 Assuming that the current date is September 21, 1991, a request to *date* or *strftime()* would yield  
114365 the following results:

```
114366 %Ec - Heisei3nen9gatsu2lnichi (Sat) 14:39:26
114367 %EC - Heisei
114368 %Ex - Heisei3nen9gatsu2lnichi (Sat)
114369 %Ey - 3
114370 %EY - Heisei3nen
```

114371 Example era definitions for the Republic of China:

```
114372 era "+:2:1913/01/01:+*:ChungHwaMingGuo:%EC%EyNen";\
114373 "+:1:1912/1/1:1912/12/31:ChungHwaMingGuo:%EYuenNen";\
114374 "+:1:1911/12/31:-*:MingChien:%EC%EyNen"
```

114375 Example definitions for the Christian Era:

```
114376 era "+:1:0001/01/01:+*:AD:%EC %Ey";\
114377 "+:1:-0001/12/31:-*:BC:%Ey %EC"
```

#### 114378 A.7.3.6 LC\_MESSAGES

114379 The **yesstr** and **nostr** locale keywords and the YESSTR and NOSTR *langinfo* items were formerly  
114380 used to match user affirmative and negative responses. In POSIX.1-200x, the **yesexpr**, **noexpr**,  
114381 YESEXPR, and NOEXPR extended regular expressions have replaced them. Applications  
114382 should use the general locale-based messaging facilities to issue prompting messages which  
114383 include sample desired responses.

### 114384 A.7.4 Locale Definition Grammar

114385 There is no additional rationale provided for this section.

114386 A.7.4.1 *Locale Lexical Conventions*

114387 There is no additional rationale provided for this section.

114388 A.7.4.2 *Locale Grammar*

114389 There is no additional rationale provided for this section.

114390 **A.7.5 Locale Definition Example**

114391 The following is an example of a locale definition file that could be used as input to the *localedef*  
 114392 utility. It assumes that the utility is executed with the *-f* option, naming a charmap file with (at  
 114393 least) the following content:

```

114394 CHARMAP
114395 <space> \x20
114396 <dollar> \x24
114397 <A> \101
114398 <a> \141
114399 <A-acute> \346
114400 <a-acute> \365
114401 <A-grave> \300
114402 <a-grave> \366
114403 \142
114404 <C> \103
114405 <c> \143
114406 <c-cedilla> \347
114407 <d> \x64
114408 <H> \110
114409 <h> \150
114410 <eszet> \xb7
114411 <s> \x73
114412 <z> \x7a
114413 END CHARMAP

```

114414 It should not be taken as complete or to represent any actual locale, but only to illustrate the  
 114415 syntax.

```

114416 #
114417 LC_CTYPE
114418 lower <a>;;<c>;<c-cedilla>;<d>;...;<z>
114419 upper A;B;C;Ç;...;Z
114420 space \x20;\x09;\x0a;\x0b;\x0c;\x0d
114421 blank \040;\011
114422 toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
114423 END LC_CTYPE
114424 #
114425 LC_COLLATE
114426 #
114427 # The following example of collation is based on
114428 # Canadian standard Z243.4.1-1998, "Canadian Alphanumeric
114429 # Ordering Standard for Character Sets of CSA Z234.4 Standard".
114430 # (Other parts of this example locale definition file do not
114431 # purport to relate to Canada, or to any other real culture.)
114432 # The proposed standard defines a 4-weight collation, such that
114433 # in the first pass, characters are compared without regard to

```

```

114434 # case or accents; in the second pass, backwards-compare without
114435 # regard to case; in the third pass, forwards-compare without
114436 # regard to diacriticals. In the 3 first passes, non-alphabetic
114437 # characters are ignored; in the fourth pass, only special
114438 # characters are considered, such that "The string that has a
114439 # special character in the lowest position comes first. If two
114440 # strings have a special character in the same position, the
114441 # collation value of the special character determines ordering.
114442 #
114443 # Only a subset of the character set is used here; mostly to
114444 # illustrate the set-up.
114445 #
114446 collating-symbol <NULL>
114447 collating-symbol <LOW_VALUE>
114448 collating-symbol <LOWER-CASE>
114449 collating-symbol <SUBSCRIPT-LOWER>
114450 collating-symbol <SUPERSCRIPT-LOWER>
114451 collating-symbol <UPPER-CASE>
114452 collating-symbol <NO-ACCENT>
114453 collating-symbol <PECULIAR>
114454 collating-symbol <LIGATURE>
114455 collating-symbol <ACUTE>
114456 collating-symbol <GRAVE>
114457 # Further collating-symbols follow.
114458 #
114459 # Properly, the standard does not include any multi-character
114460 # collating elements; the one below is added for completeness.
114461 #
114462 collating_element <ch> from "<c><h>"
114463 collating_element <CH> from "<C><H>"
114464 collating_element <Ch> from "<C><h>"
114465 #
114466 order_start forward;backward;forward;forward,position
114467 #
114468 # Collating symbols are specified first in the sequence to allocate
114469 # basic collation values to them, lower than that of any character.
114470 <NULL>
114471 <LOW_VALUE>
114472 <LOWER-CASE>
114473 <SUBSCRIPT-LOWER>
114474 <SUPERSCRIPT-LOWER>
114475 <UPPER-CASE>
114476 <NO-ACCENT>
114477 <PECULIAR>
114478 <LIGATURE>
114479 <ACUTE>
114480 <GRAVE>
114481 <RING-ABOVE>
114482 <DIAERESIS>
114483 <TILDE>
114484 # Further collating symbols are given a basic collating value here.
114485 #
114486 # Here follow special characters.

```

```

114487 <space> IGNORE; IGNORE; IGNORE; <space>
114488 # Other special characters follow here.
114489 #
114490 # Here follow the regular characters.
114491 <a> <a>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
114492 <A> <a>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
114493 <a-acute> <a>; <ACUTE>; <LOWER-CASE>; IGNORE
114494 <A-acute> <a>; <ACUTE>; <UPPER-CASE>; IGNORE
114495 <a-grave> <a>; <GRAVE>; <LOWER-CASE>; IGNORE
114496 <A-grave> <a>; <GRAVE>; <UPPER-CASE>; IGNORE
114497 <ae> " <a><e> " ; " <LIGATURE><LIGATURE> " ; \
114498 " <LOWER-CASE><LOWER-CASE> " ; IGNORE
114499 <AE> " <a><e> " ; " <LIGATURE><LIGATURE> " ; \
114500 " <UPPER-CASE><UPPER-CASE> " ; IGNORE
114501 ; <NO-ACCENT>; <LOWER-CASE>; IGNORE
114502 ; <NO-ACCENT>; <UPPER-CASE>; IGNORE
114503 <c> <c>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
114504 <C> <c>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
114505 <ch> <ch>; <NO-ACCENT>; <LOWER-CASE>; IGNORE
114506 <Ch> <ch>; <NO-ACCENT>; <PECULIAR>; IGNORE
114507 <CH> <ch>; <NO-ACCENT>; <UPPER-CASE>; IGNORE
114508 #
114509 # As an example, the strings "Bach" and "bach" could be encoded (for
114510 # compare purposes) as:
114511 # "Bach" ; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
114512 # <NO_ACCENT>; <LOW_VALUE>; <UPPER-CASE>; <LOWER-CASE>; \
114513 # <LOWER-CASE>; <NULL>
114514 # "bach" ; <a>; <ch>; <LOW_VALUE>; <NO_ACCENT>; <NO_ACCENT>; \
114515 # <NO_ACCENT>; <LOW_VALUE>; <LOWER-CASE>; <LOWER-CASE>; \
114516 # <LOWER-CASE>; <NULL>
114517 #
114518 # The two strings are equal in pass 1 and 2, but differ in pass 3.
114519 #
114520 # Further characters follow.
114521 #
114522 UNDEFINED IGNORE; IGNORE; IGNORE; IGNORE
114523 #
114524 order_end
114525 #
114526 END LC_COLLATE
114527 #
114528 LC_MONETARY
114529 int_curr_symbol "USD "
114530 currency_symbol "$ "
114531 mon_decimal_point "."
114532 mon_grouping 3;0
114533 positive_sign ""
114534 negative_sign "- "
114535 p_cs_precedes 1
114536 n_sign_posn 0
114537 END LC_MONETARY
114538 #
114539 LC_NUMERIC

```

```

114540 copy "US_en.ASCII"
114541 END LC_NUMERIC
114542 #
114543 LC_TIME
114544 abday "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
114545 #
114546 day "Sunday";"Monday";"Tuesday";"Wednesday";\
114547 "Thursday";"Friday";"Saturday"
114548 #
114549 abmon "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
114550 "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
114551 #
114552 mon "January";"February";"March";"April";\
114553 "May";"June";"July";"August";"September";\
114554 "October";"November";"December"
114555 #
114556 d_t_fmt "%a %b %d %T %Z %Y\n"
114557 END LC_TIME
114558 #
114559 LC_MESSAGES
114560 yesexpr "^[yY][[:alpha:]]*"|(OK)"
114561 #
114562 noexpr "^[nN][[:alpha:]]*"
114563 END LC_MESSAGES

```

## 114564 A.8 Environment Variables

### 114565 A.8.1 Environment Variable Definition

114566 The variable *environ* is not intended to be declared in any header, but rather to be declared by  
 114567 the user for accessing the array of strings that is the environment. This is the traditional usage of  
 114568 the symbol. Putting it into a header could break some programs that use the symbol for their  
 114569 own purposes.

114570 The decision to restrict conforming systems to the use of digits, uppercase letters, and  
 114571 underscores for environment variable names allows applications to use lowercase letters in their  
 114572 environment variable names without conflicting with any conforming system.

114573 In addition to the obvious conflict with the shell syntax for positional parameter substitution,  
 114574 some historical applications (including some shells) exclude names with leading digits from the  
 114575 environment.



## 114576 A.8.2 Internationalization Variables

114577 Utilities conforming to the Shell and Utilities volume of POSIX.1-200x and written in standard C  
114578 can access the locale variables by issuing the following call:

```
114579 setlocale(LC_ALL, " ")
```

114580 If this were omitted, the ISO C standard specifies that the C locale would be used.

114581 The DESCRIPTION of *setlocale()* requires that when setting all categories of a locale, if the value  
114582 of any of the environment variable searches yields a locale that is not supported (and non-null),  
114583 the *setlocale()* function returns a null pointer and the locale of the process is unchanged.

114584 For the standard utilities, if any of the environment variables are invalid, it makes sense to  
114585 default to an implementation-defined, consistent locale environment. It is more confusing for a  
114586 user to have partial settings occur in case of a mistake. All utilities would then behave in one  
114587 language/cultural environment. Furthermore, it provides a way of forcing the whole  
114588 environment to be the implementation-defined default. Disastrous results could occur if a  
114589 pipeline of utilities partially uses the environment variables in different ways. In this case, it  
114590 would be appropriate for utilities that use *LANG* and related variables to exit with an error if  
114591 any of the variables are invalid. For example, users typing individual commands at a terminal  
114592 might want *date* to work if *LC\_MONETARY* is invalid as long as *LC\_TIME* is valid. Since these  
114593 are conflicting reasonable alternatives, POSIX.1-200x leaves the results unspecified if the locale  
114594 environment variables would not produce a complete locale matching the specification of the  
114595 user.

114596 The locale settings of individual categories cannot be truly independent and still guarantee  
114597 correct results. For example, when collating two strings, characters must first be extracted from  
114598 each string (governed by *LC\_CTYPE*) before being mapped to collating elements (governed by  
114599 *LC\_COLLATE*) for comparison. That is, if *LC\_CTYPE* is causing parsing according to the rules of  
114600 a large, multi-byte code set (potentially returning 20 000 or more distinct character codeset  
114601 values), but *LC\_COLLATE* is set to handle only an 8-bit codeset with 256 distinct characters,  
114602 meaningful results are obviously impossible.

114603 The *LC\_MESSAGES* variable affects the language of messages generated by the standard  
114604 utilities.

114605 The description of the environment variable names starting with the characters "LC\_"  
114606 acknowledges the fact that the interfaces presented may be extended as new international  
114607 functionality is required. In the ISO C standard, names preceded by "LC\_" are reserved in the  
114608 name space for future categories.

114609 To avoid name clashes, new categories and environment variables are divided into two  
114610 classifications: "implementation-independent" and "implementation-defined".

114611 Implementation-independent names will have the following format:

```
114612 LC_NAME
```

114613 where *NAME* is the name of the new category and environment variable. Capital letters must be  
114614 used for implementation-independent names.

114615 Implementation-defined names must be in lowercase letters, as below:

```
114616 LC_name
```

### 114617 A.8.3 Other Environment Variables

#### 114618 COLUMNS, LINES

114619 The default values for the number of column positions, *COLUMNS*, and screen height, *LINES*,  
 114620 are unspecified because historical implementations use different methods to determine values  
 114621 corresponding to the size of the screen in which the utility is run. This size is typically known to  
 114622 the implementation through the value of *TERM*, or by more elaborate methods such as  
 114623 extensions to the *stty* utility or knowledge of how the user is dynamically resizing windows on a  
 114624 bit-mapped display terminal. Users should not need to set these variables in the environment  
 114625 unless there is a specific reason to override the default behavior of the implementation, such as  
 114626 to display data in an area arbitrarily smaller than the terminal or window. Values for these  
 114627 variables that are not decimal integers greater than zero are implicitly undefined values; it is  
 114628 unnecessary to enumerate all of the possible values outside of the acceptable set.

#### 114629 LOGNAME

114630 In most implementations, the value of such a variable is easily forged, so security-critical  
 114631 applications should rely on other means of determining user identity. *LOGNAME* is required to  
 114632 be constructed from the portable filename character set for reasons of interchange. No diagnostic  
 114633 condition is specified for violating this rule, and no requirement for enforcement exists. The  
 114634 intent of the requirement is that if extended characters are used, the “guarantee” of portability  
 114635 implied by a standard is void.

#### 114636 PATH

114637 Many historical implementations of the Bourne shell do not interpret a trailing colon to  
 114638 represent the current working directory and are thus non-conforming. The C Shell and the  
 114639 KornShell conform to POSIX.1-200x on this point. The usual name of dot may also be used to  
 114640 refer to the current working directory.

114641 Many implementations historically have used a default value of */bin* and */usr/bin* for the *PATH*  
 114642 variable. POSIX.1-200x does not mandate this default path be identical to that retrieved from  
 114643 *getconf PATH* because it is likely that the standardized utilities may be provided in another  
 114644 directory separate from the directories used by some historical applications.

#### 114645 SHELL

114646 The *SHELL* variable names the preferred shell of the user; it is a guide to applications. There is  
 114647 no direct requirement that that shell conform to POSIX.1-200x; that decision should rest with the  
 114648 user. It is the intention of the standard developers that alternative shells be permitted, if the user  
 114649 chooses to develop or acquire one. An operating system that builds its shell into the “kernel” in  
 114650 such a manner that alternative shells would be impossible does not conform to the spirit of  
 114651 POSIX.1-200x.

#### 114652 TZ

114653 The quoted form of the timezone variable allows timezone names of the form UTC+1 (or any  
 114654 name that contains the character plus ('+'), the character minus ('-'), or digits), which may be  
 114655 appropriate for countries that do not have an official timezone name. It would be coded as  
 114656 <UTC+1>+1<UTC+2>, which would cause *std* to have a value of UTC+1 and *dst* a value of  
 114657 UTC+2, each with a length of 5 characters. This does not appear to conflict with any existing  
 114658 usage. The characters '<' and '>' were chosen for quoting because they are easier to parse  
 114659 visually than a quoting character that does not provide some sense of bracketing (and in a string  
 114660 like this, such bracketing is helpful). They were also chosen because they do not need special  
 114661 treatment when assigning to the *TZ* variable. Users are often confused by embedding quotes in a

114662 string. Because '`<`' and '`>`' are meaningful to the shell, the whole string would have to be  
 114663 quoted, but that is easily explained. (Parentheses would have presented the same problems.)  
 114664 Although the '`>`' symbol could have been permitted in the string by either escaping it or  
 114665 doubling it, it seemed of little value to require that. This could be provided as an extension if  
 114666 there was a need. Timezone names of this new form lead to a requirement that the value of  
 114667 `{_POSIX_TZNAME_MAX}` change from 3 to 6.

114668 Since the *TZ* environment variable is usually inherited by all applications started by a user after  
 114669 the value of the *TZ* environment variable is changed and since many applications run using the  
 114670 C or POSIX locale, using characters that are not in the portable character set in the *std* and *dst*  
 114671 fields could cause unexpected results.

114672 The format of the *TZ* environment variable is changed in Issue 6 to allow for the quoted form, as  
 114673 defined in earlier versions of the ISO POSIX-1 standard.

114674 IEEE Std 1003.1-2001/Cor 1-2002, item XBD/TC1/D6/7 is applied, adding the *ctime\_r()* and  
 114675 *localtime\_r()* functions to the list of functions that use the *TZ* environment variable.

## 114676 A.9 Regular Expressions

114677 Rather than repeating the description of REs for each utility supporting REs, the standard  
 114678 developers preferred a common, comprehensive description of regular expressions in one place.  
 114679 The most common behavior is described here, and exceptions or extensions to this are  
 114680 documented for the respective utilities, as appropriate.

114681 The BRE corresponds to the *ed* or historical *grep* type, and the ERE corresponds to the historical  
 114682 *egrep* type (now *grep -E*).

114683 The text is based on the *ed* description and substantially modified, primarily to aid developers  
 114684 and others in the understanding of the capabilities and limitations of REs. Much of this was  
 114685 influenced by internationalization requirements.

114686 It should be noted that the definitions in this section do not cover the *tr* utility; the *tr* syntax does  
 114687 not employ REs.

114688 The specification of REs is particularly important to internationalization because pattern  
 114689 matching operations are very basic operations in business and other operations. The syntax and  
 114690 rules of REs are intended to be as intuitive as possible to make them easy to understand and use.  
 114691 The historical rules and behavior do not provide that capability to non-English language users,  
 114692 and do not provide the necessary support for commonly used characters and language  
 114693 constructs. It was necessary to provide extensions to the historical RE syntax and rules to  
 114694 accommodate other languages.

114695 As they are limited to bracket expressions, the rationale for these modifications is in XBD [Section](#)  
 114696 [9.3.5](#) (on page 170).

### 114697 A.9.1 Regular Expression Definitions

114698 It is possible to determine what strings correspond to subexpressions by recursively applying  
 114699 the leftmost longest rule to each subexpression, but only with the proviso that the overall match  
 114700 is leftmost longest. For example, matching "`\(ac*\\)c*d[ac]*\1`" against *acdacaaa* matches  
 114701 *acdacaaa* (with `\1=a`); simply matching the longest match for "`\(ac*\\)`" would yield `\1=ac`, but  
 114702 the overall match would be smaller (*acdac*). Conceptually, the implementation must examine  
 114703 every possible match and among those that yield the leftmost longest total matches, pick the one  
 114704 that does the longest match for the leftmost subexpression, and so on. Note that this means that  
 114705 matching by subexpressions is context-dependent: a subexpression within a larger RE may

114706 match a different string from the one it would match as an independent RE, and two instances of  
 114707 the same subexpression within the same larger RE may match different lengths even in similar  
 114708 sequences of characters. For example, in the ERE "(a.\*b)(a.\*b)", the two identical  
 114709 subexpressions would match four and six characters, respectively, of *accbaccccb*.

114710 The definition of single character has been expanded to include also collating elements  
 114711 consisting of two or more characters; this expansion is applicable only when a bracket  
 114712 expression is included in the BRE or ERE. An example of such a collating element may be the  
 114713 Dutch *ij*, which collates as a 'y'. In some encodings, a ligature "i with j" exists as a character  
 114714 and would represent a single-character collating element. In another encoding, no such ligature  
 114715 exists, and the two-character sequence *ij* is defined as a multi-character collating element.  
 114716 Outside brackets, the *ij* is treated as a two-character RE and matches the same characters in a  
 114717 string. Historically, a bracket expression only matched a single character. The ISO POSIX-2: 1993  
 114718 standard required bracket expressions like "[<sup>^</sup>[:lower:]]" to match multi-character collating  
 114719 elements such as "*ij*". However, this requirement led to behavior that many users did not  
 114720 expect and that could not feasibly be mimicked in user code, and it was rarely if ever  
 114721 implemented correctly. The current standard leaves it unspecified whether a bracket expression  
 114722 matches a multi-character collating element, allowing both historical and ISO POSIX-2: 1993  
 114723 standard implementations to conform.

114724 Also, in the current standard, it is unspecified whether character class expressions like  
 114725 "[[:lower:]]" can include multi-character collating elements like "*ij*"; hence  
 114726 "[[:lower:]]" can match "*ij*", and "[<sup>^</sup>[:lower:]]" can fail to match "*ij*". Common  
 114727 practice is for a character class expression to match a collating element if it matches the collating  
 114728 element's first character.

## 114729 A.9.2 Regular Expression General Requirements

114730 The definition of which sequence is matched when several are possible is based on the leftmost-  
 114731 longest rule historically used by deterministic recognizers. This rule is easier to define and  
 114732 describe, and arguably more useful, than the first-match rule historically used by non-  
 114733 deterministic recognizers. It is thought that dependencies on the choice of rule are rare; carefully  
 114734 contrived examples are needed to demonstrate the difference.

114735 A formal expression of the leftmost-longest rule is:

114736 The search is performed as if all possible suffixes of the string were tested for a prefix  
 114737 matching the pattern; the longest suffix containing a matching prefix is chosen, and the  
 114738 longest possible matching prefix of the chosen suffix is identified as the matching  
 114739 sequence.

114740 Historically, most RE implementations only match lines, not strings. However, that is more an  
 114741 effect of the usage than of an inherent feature of REs themselves. Consequently, POSIX.1-200x  
 114742 does not regard <newline>s as special; they are ordinary characters, and both a period and a  
 114743 non-matching list can match them. Those utilities (like *grep*) that do not allow <newline>s to  
 114744 match are responsible for eliminating any <newline> from strings before matching against the  
 114745 RE. The *regcomp()* function, however, can provide support for such processing without violating  
 114746 the rules of this section.

114747 Some implementations of *egrep* have had very limited flexibility in handling complex EREs.  
 114748 POSIX.1-200x does not attempt to define the complexity of a BRE or ERE, but does place a lower  
 114749 limit on it—any RE must be handled, as long as it can be expressed in 256 bytes or less. (Of  
 114750 course, this does not place an upper limit on the implementation.) There are historical programs  
 114751 using a non-deterministic-recognizer implementation that should have no difficulty with this  
 114752 limit. It is possible that a good approach would be to attempt to use the faster, but more limited,  
 114753 deterministic recognizer for simple expressions and to fall back on the non-deterministic

114754 recognizer for those expressions requiring it. Non-deterministic implementations must be  
 114755 careful to observe the rules on which match is chosen; the longest match, not the first match,  
 114756 starting at a given character is used.

114757 The term “invalid” highlights a difference between this section and some others: POSIX.1-200x  
 114758 frequently avoids mandating of errors for syntax violations because they can be used by  
 114759 implementors to trigger extensions. However, the authors of the internationalization features of  
 114760 REs wanted to mandate errors for certain conditions to identify usage problems or non-portable  
 114761 constructs. These are identified within this rationale as appropriate. The remaining syntax  
 114762 violations have been left implicitly or explicitly undefined. For example, the BRE construct  
 114763 “\{1, 2, 3\}” does not comply with the grammar. A conforming application cannot rely on it  
 114764 producing an error nor matching the literal characters “\{1, 2, 3\}”.

114765 The term “undefined” was used in favor of “unspecified” because many of the situations are  
 114766 considered errors on some implementations, and the standard developers considered that  
 114767 consistency throughout the section was preferable to mixing undefined and unspecified.

### 114768 **A.9.3 Basic Regular Expressions**

114769 There is no additional rationale provided for this section.

#### 114770 *A.9.3.1 BREs Matching a Single Character or Collating Element*

114771 There is no additional rationale provided for this section.

#### 114772 *A.9.3.2 BRE Ordinary Characters*

114773 There is no additional rationale provided for this section.

#### 114774 *A.9.3.3 BRE Special Characters*

114775 There is no additional rationale provided for this section.

#### 114776 *A.9.3.4 Periods in BREs*

114777 There is no additional rationale provided for this section.

#### 114778 *A.9.3.5 RE Bracket Expression*

114779 Range expressions are, historically, an integral part of REs. However, the requirements of  
 114780 “natural language behavior” and portability do conflict. In the POSIX locale, ranges must be  
 114781 treated according to the collating sequence and include such characters that fall within the range  
 114782 based on that collating sequence, regardless of character values. In other locales, ranges have  
 114783 unspecified behavior.

114784 Some historical implementations allow range expressions where the ending range point of one  
 114785 range is also the starting point of the next (for instance, “[a-m-o]”). This behavior should not  
 114786 be permitted, but to avoid breaking historical implementations, it is now *undefined* whether it is  
 114787 a valid expression and how it should be interpreted.

114788 Current practice in *awk* and *lex* is to accept escape sequences in bracket expressions as per XBD  
 114789 [Table 5-1](#) (on page 108), while the normal ERE behavior is to regard such a sequence as  
 114790 consisting of two characters. Allowing the *awk/lex* behavior in EREs would change the normal  
 114791 behavior in an unacceptable way; it is expected that *awk* and *lex* will decode escape sequences in  
 114792 EREs before passing them to *regcomp()* or comparable routines. Each utility describes the escape  
 114793 sequences it accepts as an exception to the rules in this section; the list is not the same, for  
 114794 historical reasons.

114795 As noted previously, the new syntax and rules have been added to accommodate other  
114796 languages than English. The remainder of this section describes the rationale for these  
114797 modifications.

114798 In the POSIX locale, a regular expression that starts with a range expression matches a set of  
114799 strings that are contiguously sorted, but this is not necessarily true in other locales. For example,  
114800 a French locale might have the following behavior:

```
114801 $ ls
114802 alpha Alpha estimé ESTIMÉ été eurêka
114803 $ ls [a-e]*
114804 alpha Alpha estimé eurêka
```

114805 Such disagreements between matching and contiguous sorting are unavoidable because POSIX  
114806 sorting cannot be implemented in terms of a deterministic finite-state automaton (DFA), but  
114807 range expressions by design are implementable in terms of DFAs.

114808 Historical implementations used native character order to interpret range expressions. The  
114809 ISO POSIX-2:1993 standard instead required collating element order (CEO): the order that  
114810 collating elements were specified between the **order\_start** and **order\_end** keywords in the  
114811 *LC\_COLLATE* category of the current locale. CEO had some advantages in portability over the  
114812 native character order, but it also had some disadvantages:

- 114813 • CEO could not feasibly be mimicked in user code, leading to inconsistencies between  
114814 POSIX matchers and matchers in popular user programs like Emacs, *ksh*, and Perl.
- 114815 • CEO caused range expressions to match accented and capitalized letters contrary to many  
114816 users' expectations. For example, "[a-e]" typically matched both 'E' and 'á' but  
114817 neither 'A' nor 'é'.
- 114818 • CEO was not consistent across implementations. In practice, CEO was often less portable  
114819 than native character order. For example, it was common for the CEOs of two  
114820 implementation-supplied locales to disagree, even if both locales were named "da\_DK".

114821 Because of these problems, some implementations of regular expressions continued to use native  
114822 character order. Others used the collation sequence, which is more consistent with sorting than  
114823 either CEO or native order, but which departs further from the traditional POSIX semantics  
114824 because it generally requires "[a-e]" to match either 'A' or 'E' but not both. As a result of  
114825 this kind of implementation variation, programmers who wanted to write portable regular  
114826 expressions could not rely on the ISO POSIX-2:1993 standard guarantees in practice.

114827 While revising the standard, lengthy consideration was given to proposals to attack this problem  
114828 by adding an API for querying the CEO to allow user-mode matchers, but none of these  
114829 proposals had implementation experience and none achieved consensus. Leaving the standard  
114830 alone was also considered, but rejected due to the problems described above.

114831 The current standard leaves unspecified the behavior of a range expression outside the POSIX  
114832 locale. This makes it clearer that conforming applications should avoid range expressions  
114833 outside the POSIX locale, and it allows implementations and compatible user-mode matchers to  
114834 interpret range expressions using native order, CEO, collation sequence, or other, more  
114835 advanced techniques. The concerns which led to this change were raised in IEEE PASC  
114836 interpretation 1003.2 #43 and others, and related to ambiguities in the specification of how  
114837 multi-character collating elements should be handled in range expressions. These ambiguities  
114838 had led to multiple interpretations of the specification, in conflicting ways, which led to varying  
114839 implementations. As noted above, efforts were made to resolve the differences, but no solution  
114840 has been found that would be specific enough to allow for portable software while not  
114841 invalidating existing implementations.

114842 The standard developers recognize that collating elements are important, such elements being  
 114843 common in several European languages; for example, 'ch' or 'll' in traditional Spanish;  
 114844 'aa' in several Scandinavian languages. Existing internationalized implementations have  
 114845 processed, and continue to process, these elements in range expressions. Efforts are expected to  
 114846 continue in the future to find a way to define the behavior of these elements precisely and  
 114847 portably.

114848 The ISO POSIX-2:1993 standard required "[b-a]" to be an invalid expression in the POSIX  
 114849 locale, but this requirement has been relaxed in this version of the standard so that "[b-a]" can  
 114850 instead be treated as a valid expression that does not match any string.

#### 114851 A.9.3.6 BREs Matching Multiple Characters

114852 The limit of nine back-references to subexpressions in the RE is based on the use of a single-digit  
 114853 identifier; increasing this to multiple digits would break historical applications. This does not  
 114854 imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten  
 114855 subexpressions:

```
114856 \(\(\(ab\) *c\) *d\)\(ef\) *\(\(gh\)\{2\}\(ij\) *\(\(kl\) *\(\(mn\) *\(\(op\) *\(\(qr\) *
```

114857 The standard developers regarded the common historical behavior, which supported " $n^*$ ", but  
 114858 not " $n\{\min, \max\}$ ", " $\{...\}$ ", or " $\{...\}\{\min, \max\}$ ", as a non-intentional  
 114859 result of a specific implementation, and they supported both duplication and interval  
 114860 expressions following subexpressions and back-references.

114861 The changes to the processing of the back-reference expression remove an unspecified or  
 114862 ambiguous behavior in the Shell and Utilities volume of POSIX.1-200x, aligning it with the  
 114863 requirements specified for the *regcomp()* expression, and is the result of PASC Interpretation  
 114864 1003.2-92 #43 submitted for the ISO POSIX-2:1993 standard.

#### 114865 A.9.3.7 BRE Precedence

114866 There is no additional rationale provided for this section.

#### 114867 A.9.3.8 BRE Expression Anchoring

114868 Often, the dollar sign is viewed as matching the ending <newline> in text files. This is not  
 114869 strictly true; the <newline> is typically eliminated from the strings to be matched, and the dollar  
 114870 sign matches the terminating null character.

114871 The ability of '^', '\$', and '\*' to be non-special in certain circumstances may be confusing to  
 114872 some programmers, but this situation was changed only in a minor way from historical practice  
 114873 to avoid breaking many historical scripts. Some consideration was given to making the use of  
 114874 the anchoring characters undefined if not escaped and not at the beginning or end of strings.  
 114875 This would cause a number of historical BREs, such as " $2^10$ ", " $\$HOME$ ", and " $\$1.35$ ", that  
 114876 relied on the characters being treated literally, to become invalid.

114877 However, one relatively uncommon case was changed to allow an extension used on some  
 114878 implementations. Historically, the BREs " $\^foo$ " and " $\{^foo\}$ " did not match the same  
 114879 string, despite the general rule that subexpressions and entire BREs match the same strings. To  
 114880 increase consensus, POSIX.1-200x has allowed an extension on some implementations to treat  
 114881 these two cases in the same way by declaring that anchoring *may* occur at the beginning or end  
 114882 of a subexpression. Therefore, portable BREs that require a literal circumflex at the beginning or  
 114883 a dollar sign at the end of a subexpression must escape them. Note that a BRE such as  
 114884 " $a\{^bc\}$ " will either match " $a^bc$ " or nothing on different systems under the rules.

114885 ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped  
 114886 anchor character has never matched its literal counterpart outside a bracket expression. Some

114887 implementations treated "foo\$bar" as a valid expression that never matched anything; others  
114888 treated it as invalid. POSIX.1-200x mandates the former, valid unmatched behavior.

114889 Some implementations have extended the BRE syntax to add alternation. For example, the  
114890 subexpression "\ (foo\$ | bar\ )" would match either "foo" at the end of the string or "bar"  
114891 anywhere. The extension is triggered by the use of the undefined "\|" sequence. Because the  
114892 BRE is undefined for portable scripts, the extending system is free to make other assumptions,  
114893 such that the '\$' represents the end-of-line anchor in the middle of a subexpression. If it were  
114894 not for the extension, the '\$' would match a literal dollar sign under the rules.

#### 114895 A.9.4 Extended Regular Expressions

114896 As with BREs, the standard developers decided to make the interpretation of escaped ordinary  
114897 characters undefined.

114898 The right parenthesis is not listed as an ERE special character because it is only special in the  
114899 context of a preceding left parenthesis. If found without a preceding left parenthesis, the right  
114900 parenthesis has no special meaning.

114901 The interval expression, "{m,n}", has been added to EREs. Historically, the interval expression  
114902 has only been supported in some ERE implementations. The standard developers estimated that  
114903 the addition of interval expressions to EREs would not decrease consensus and would also make  
114904 BREs more of a subset of EREs than in many historical implementations.

114905 It was suggested that, in addition to interval expressions, back-references ('\n') should also be  
114906 added to EREs. This was rejected by the standard developers as likely to decrease consensus.

114907 In historical implementations, multiple duplication symbols are usually interpreted from left to  
114908 right and treated as additive. As an example, "a+b" matches zero or more instances of 'a'  
114909 followed by a 'b'. In POSIX.1-200x, multiple duplication symbols are undefined; that is, they  
114910 cannot be relied upon for conforming applications. One reason for this is to provide some scope  
114911 for future enhancements.

114912 The precedence of operations differs between EREs and those in *lex*; in *lex*, for historical reasons,  
114913 interval expressions have a lower precedence than concatenation.

##### 114914 A.9.4.1 EREs Matching a Single Character or Collating Element

114915 There is no additional rationale provided for this section.

##### 114916 A.9.4.2 ERE Ordinary Characters

114917 There is no additional rationale provided for this section.

##### 114918 A.9.4.3 ERE Special Characters

114919 There is no additional rationale provided for this section.

##### 114920 A.9.4.4 Periods in EREs

114921 There is no additional rationale provided for this section.



## 114922 A.9.4.5 ERE Bracket Expression

114923 There is no additional rationale provided for this section.

## 114924 A.9.4.6 EREs Matching Multiple Characters

114925 There is no additional rationale provided for this section.

## 114926 A.9.4.7 ERE Alternation

114927 There is no additional rationale provided for this section.

## 114928 A.9.4.8 ERE Precedence

114929 There is no additional rationale provided for this section.

## 114930 A.9.4.9 ERE Expression Anchoring

114931 There is no additional rationale provided for this section.

114932 **A.9.5 Regular Expression Grammar**

114933 The grammars are intended to represent the range of acceptable syntaxes available to  
 114934 conforming applications. There are instances in the text where undefined constructs are  
 114935 described; as explained previously, these allow implementation extensions. There is no intended  
 114936 requirement that an implementation extension must somehow fit into the grammars shown  
 114937 here.

114938 The BRE grammar does not permit L\_ANCHOR or R\_ANCHOR inside "\(" and "\)" (which  
 114939 implies that '^' and '\$' are ordinary characters). This reflects the semantic limits on the  
 114940 application, as noted in XBD Section 9.3.8 (on page 173). Implementations are permitted to  
 114941 extend the language to interpret '^' and '\$' as anchors in these locations, and as such,  
 114942 conforming applications cannot use unescaped '^' and '\$' in positions inside "\(" and "\)"  
 114943 that might be interpreted as anchors.

114944 The ERE grammar does not permit several constructs that XBD Section 9.4.2 (on page 174) and  
 114945 Section 9.4.3 (on page 174) specify as having undefined results:

- 114946 • ORD\_CHAR preceded by '\\'
- 114947 • ERE\_dupl\_symbol(s) appearing first in an ERE, or immediately following '|', '^', or '('
- 114948 • '{' not part of a valid ERE\_dupl\_symbol
- 114949 • '|' appearing first or last in an ERE, or immediately following '|' or '(', or  
 114950 immediately preceding ')'

114951 Implementations are permitted to extend the language to allow these. Conforming applications  
 114952 cannot use such constructs.

## 114953 A.9.5.1 BRE/ERE Grammar Lexical Conventions

114954 There is no additional rationale provided for this section.

114955 A.9.5.2 *RE and Bracket Expression Grammar*

114956 The removal of the *Back\_open\_paren Back\_close\_paren* option from the *nondupl\_RE* specification is  
 114957 the result of PASC Interpretation 1003.2-92 #43 submitted for the ISO POSIX-2:1993 standard.  
 114958 Although the grammar required support for null subexpressions, this section does not describe  
 114959 the meaning of, and historical practice did not support, this construct.

114960 A.9.5.3 *ERE Grammar*

114961 There is no additional rationale provided for this section.

114962 **A.10 Directory Structure and Devices**114963 **A.10.1 Directory Structure and Files**

114964 A description of the historical */usr/tmp* was omitted, removing any concept of differences in  
 114965 emphasis between the */* and */usr* directories. The descriptions of */bin*, */usr/bin*, */lib*, and */usr/lib*  
 114966 were omitted because they are not useful for applications. In an early draft, a distinction was  
 114967 made between system and application directory usage, but this was not found to be useful.

114968 The directories */* and */dev* are included because the notion of a hierarchical directory structure is  
 114969 key to other information presented elsewhere in POSIX.1-200x. In early drafts, it was argued that  
 114970 special devices and temporary files could conceivably be handled without a directory structure  
 114971 on some implementations. For example, the system could treat the characters *"/tmp"* as a  
 114972 special token that would store files using some non-POSIX file system structure. This notion was  
 114973 rejected by the standard developers, who required that all the files in this section be  
 114974 implemented via POSIX file systems.

114975 The */tmp* directory is retained in POSIX.1-200x to accommodate historical applications that  
 114976 assume its availability. Implementations are encouraged to provide suitable directory names in  
 114977 the environment variable *TMPDIR* and applications are encouraged to use the contents of  
 114978 *TMPDIR* for creating temporary files.

114979 The standard files */dev/null* and */dev/tty* are required to be both readable and writable to allow  
 114980 applications to have the intended historical access to these files.

114981 The standard file */dev/console* has been added for alignment with the Single UNIX  
 114982 Specification.

114983 **A.10.2 Output Devices and Terminal Types**

114984 IEEE Std 1003.1-2001/Cor 2-2004, item XBD/TC2/D6/17 is applied, making it clear that the  
 114985 requirements for documenting terminal support are in the system documentation.

## A.11 General Terminal Interface

If the implementation does not support this interface on any device types, it should behave as if it were being used on a device that is not a terminal device (in most cases *errno* will be set to [ENOTTY] on return from functions defined by this interface). This is based on the fact that many applications are written to run both interactively and in some non-interactive mode, and they adapt themselves at runtime. Requiring that they all be modified to test an environment variable to determine whether they should try to adapt is unnecessary. On a system that provides no general terminal interface, providing all the entry points as stubs that return [ENOTTY] (or an equivalent, as appropriate) has the same effect and requires no changes to the application.

Although the needs of both interface implementors and application developers were addressed throughout POSIX.1-200x, this section pays more attention to the needs of the latter. This is because, while many aspects of the programming interface can be hidden from the user by the application developer, the terminal interface is usually a large part of the user interface. Although to some extent the application developer can build missing features or work around inappropriate ones, the difficulties of doing that are greater in the terminal interface than elsewhere. For example, efficiency prohibits the average program from interpreting every character passing through it in order to simulate character erase, line kill, and so on. These functions should usually be done by the operating system, possibly at the interrupt level.

The *tc\**() functions were introduced as a way of avoiding the problems inherent in the traditional *ioctl*() function and in variants of it that were proposed. For example, *tcsetattr*() is specified in place of the use of the TCSETA *ioctl*() command function. This allows specification of all the arguments in a manner consistent with the ISO C standard unlike the varying third argument of *ioctl*(), which is sometimes a pointer (to any of many different types) and sometimes an *int*.

The advantages of this new method include:

- It allows strict type checking.
- The direction of transfer of control data is explicit.
- Portable capabilities are clearly identified.
- The need for a general interface routine is avoided.
- Size of the argument is well-defined (there is only one type).

The disadvantages include:

- No historical implementation used the new method.
- There are many small routines instead of one general-purpose one.
- The historical parallel with *fcntl*() is broken.

The issue of modem control was excluded from POSIX.1-200x on the grounds that:

- It was concerned with setting and control of hardware timers.
- The appropriate timers and settings vary widely internationally.
- Feedback from European computer manufacturers indicated that this facility was not consistent with European needs and that specification of such a facility was not a requirement for portability.

115027 **A.11.1 Interface Characteristics**115028 *A.11.1.1 Opening a Terminal Device File*

115029 There is no additional rationale provided for this section.

115030 *A.11.1.2 Process Groups*

115031 There is a potential race when the members of the foreground process group on a terminal leave  
 115032 that process group, either by exit or by changing process groups. After the last process exits the  
 115033 process group, but before the foreground process group ID of the terminal is changed (usually  
 115034 by a job control shell), it would be possible for a new process to be created with its process ID  
 115035 equal to the terminal's foreground process group ID. That process might then become the  
 115036 process group leader and accidentally be placed into the foreground on a terminal that was not  
 115037 necessarily its controlling terminal. As a result of this problem, the controlling terminal is  
 115038 defined to not have a foreground process group during this time.

115039 The cases where a controlling terminal has no foreground process group occur when all  
 115040 processes in the foreground process group either terminate and are waited for or join other  
 115041 process groups via *setpgid()* or *setsid()*. If the process group leader terminates, this is the first  
 115042 case described; if it leaves the process group via *setpgid()*, this is the second case described (a  
 115043 process group leader cannot successfully call *setsid()*). When one of those cases causes a  
 115044 controlling terminal to have no foreground process group, it has two visible effects on  
 115045 applications. The first is the value returned by *tcgetpgrp()*. The second (which occurs only in the  
 115046 case where the process group leader terminates) is the sending of signals in response to special  
 115047 input characters. The intent of POSIX.1-200x is that no process group be wrongly identified as  
 115048 the foreground process group by *tcgetpgrp()* or unintentionally receive signals because of  
 115049 placement into the foreground.

115050 In 4.3 BSD, the old process group ID continues to be used to identify the foreground process  
 115051 group and is returned by the function equivalent to *tcgetpgrp()*. In that implementation it is  
 115052 possible for a newly created process to be assigned the same value as a process ID and then form  
 115053 a new process group with the same value as a process group ID. The result is that the new  
 115054 process group would receive signals from this terminal for no apparent reason, and  
 115055 POSIX.1-200x precludes this by forbidding a process group from entering the foreground in this  
 115056 way. It would be more direct to place part of the requirement made by the last sentence under  
 115057 *fork()*, but there is no convenient way for that section to refer to the value that *tcgetpgrp()*  
 115058 returns, since in this case there is no process group and thus no process group ID.

115059 One possibility for a conforming implementation is to behave similarly to 4.3 BSD, but to  
 115060 prevent this reuse of the ID, probably in the implementation of *fork()*, as long as it is in use by  
 115061 the terminal.

115062 Another possibility is to recognize when the last process stops using the terminal's foreground  
 115063 process group ID, which is when the process group lifetime ends, and to change the terminal's  
 115064 foreground process group ID to a reserved value that is never used as a process ID or process  
 115065 group ID. (See the definition of *process group lifetime* in the definitions section.) The process ID  
 115066 can then be reserved until the terminal has another foreground process group.

115067 The 4.3 BSD implementation permits the leader (and only member) of the foreground process  
 115068 group to leave the process group by calling the equivalent of *setpgid()* and to later return,  
 115069 expecting to return to the foreground. There are no known application needs for this behavior,  
 115070 and POSIX.1-200x neither requires nor forbids it (except that it is forbidden for session leaders)  
 115071 by leaving it unspecified.

115072 A.11.1.3 *The Controlling Terminal*

115073 POSIX.1-200x does not specify a mechanism by which to allocate a controlling terminal. This is  
 115074 normally done by a system utility (such as *getty*) and is considered an administrative feature  
 115075 outside the scope of POSIX.1-200x.

115076 Historical implementations allocate controlling terminals on certain *open()* calls. Since *open()* is  
 115077 part of POSIX.1, its behavior had to be dealt with. The traditional behavior is not required  
 115078 because it is not very straightforward or flexible for either implementations or applications.  
 115079 However, because of its prevalence, it was not practical to disallow this behavior either. Thus, a  
 115080 mechanism was standardized to ensure portable, predictable behavior in *open()*.

115081 Some historical implementations deallocate a controlling terminal on the last system-wide close.  
 115082 This behavior is neither required nor prohibited. Even on implementations that do provide this  
 115083 behavior, applications generally cannot depend on it due to its system-wide nature.

115084 A.11.1.4 *Terminal Access Control*

115085 The access controls described in this section apply only to a process that is accessing its  
 115086 controlling terminal. A process accessing a terminal that is not its controlling terminal is  
 115087 effectively treated the same as a member of the foreground process group. While this may seem  
 115088 unintuitive, note that these controls are for the purpose of job control, not security, and job  
 115089 control relates only to the controlling terminal of a process. Normal file access permissions  
 115090 handle security.

115091 If the process calling *read()* or *write()* is in a background process group that is orphaned, it is not  
 115092 desirable to stop the process group, as it is no longer under the control of a job control shell that  
 115093 could put it into the foreground again. Accordingly, calls to *read()* or *write()* functions by such  
 115094 processes receive an immediate error return. This is different from 4.2 BSD, which kills orphaned  
 115095 processes that receive terminal stop signals.

115096 The foreground/background/orphaned process group check performed by the terminal driver  
 115097 must be repeatedly performed until the calling process moves into the foreground or until the  
 115098 process group of the calling process becomes orphaned. That is, when the terminal driver  
 115099 determines that the calling process is in the background and should receive a job control signal,  
 115100 it sends the appropriate signal (SIGTTIN or SIGTTOU) to every process in the process group of  
 115101 the calling process and then it allows the calling process to immediately receive the signal. The  
 115102 latter is typically performed by blocking the process so that the signal is immediately noticed.  
 115103 Note, however, that after the process finishes receiving the signal and control is returned to the  
 115104 driver, the terminal driver must re-execute the foreground/background/orphaned process  
 115105 group check. The process may still be in the background, either because it was continued in the  
 115106 background by a job control shell, or because it caught the signal and did nothing.

115107 The terminal driver repeatedly performs the foreground/background/orphaned process group  
 115108 checks whenever a process is about to access the terminal. In the case of *write()* or the control  
 115109 *tc\*()* functions, the check is performed at the entry of the function. In the case of *read()*, the  
 115110 check is performed not only at the entry of the function, but also after blocking the process to  
 115111 wait for input characters (if necessary). That is, once the driver has determined that the process  
 115112 calling the *read()* function is in the foreground, it attempts to retrieve characters from the input  
 115113 queue. If the queue is empty, it blocks the process waiting for characters. When characters are  
 115114 available and control is returned to the driver, the terminal driver must return to the repeated  
 115115 foreground/background/orphaned process group check again. The process may have moved  
 115116 from the foreground to the background while it was blocked waiting for input characters.

115117 *A.11.1.5 Input Processing and Reading Data*

115118 There is no additional rationale provided for this section.

115119 *A.11.1.6 Canonical Mode Input Processing*115120 The term “character” is intended here. ERASE should erase the last character, not the last byte.  
115121 In the case of multi-byte characters, these two may be different.115122 4.3 BSD has a WERASE character that erases the last “word” typed (but not any preceding  
115123 <blank>s or <tab>s). A word is defined as a sequence of non-<blank>s, with <tab>s counted as  
115124 <blank>s. Like ERASE, WERASE does not erase beyond the beginning of the line. This  
115125 WERASE feature has not been specified in POSIX.1 because it is difficult to define in the  
115126 international environment. It is only useful for languages where words are delimited by  
115127 <blank>s. In some ideographic languages, such as Japanese and Chinese, words are not  
115128 delimited at all. The WERASE character should presumably go back to the beginning of a  
115129 sentence in those cases; practically, this means it would not be used much for those languages.115130 It should be noted that there is a possible inherent deadlock if the application and  
115131 implementation conflict on the value of {MAX\_CANON}. With ICANON set (if IXOFF is  
115132 enabled) and more than {MAX\_CANON} characters transmitted without a <linefeed>,  
115133 transmission will be stopped, the <linefeed> (or <carriage-return> when ICRLF is set) will never  
115134 arrive, and the *read()* will never be satisfied.115135 An application should not set IXOFF if it is using canonical mode unless it knows that (even in  
115136 the face of a transmission error) the conditions described previously cannot be met or unless it  
115137 is prepared to deal with the possible deadlock in some other way, such as timeouts.115138 It should also be noted that this can be made to happen in non-canonical mode if the trigger  
115139 value for sending IXOFF is less than VMIN and VTIME is zero.115140 *A.11.1.7 Non-Canonical Mode Input Processing*

115141 Some points to note about MIN and TIME:

- 115142 1. The interactions of MIN and TIME are not symmetric. For example, when MIN>0 and
- 
- 115143 TIME=0, TIME has no effect. However, in the opposite case where MIN=0 and TIME>0,
- 
- 115144 both MIN and TIME play a role in that MIN is satisfied with the receipt of a single
- 
- 115145 character.
- 
- 115146 2. Also note that in case A (MIN>0, TIME>0), TIME represents an inter-character timer,
- 
- 115147 while in case C (MIN=0, TIME>0), TIME represents a read timer.

115148 These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where  
115149 MIN>0, exist to handle burst-mode activity (for example, file transfer programs) where a  
115150 program would like to process at least MIN characters at a time. In case A, the inter-character  
115151 timer is activated by a user as a safety measure; in case B, it is turned off.115152 Cases C and D exist to handle single-character timed transfers. These cases are readily adaptable  
115153 to screen-based applications that need to know if a character is present in the input queue before  
115154 refreshing the screen. In case C, the read is timed; in case D, it is not.115155 Another important note is that MIN is always just a minimum. It does not denote a record  
115156 length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20  
115157 characters are returned to the user. In the special case of MIN=0, this still applies: if more than  
115158 one character is available, they all will be returned immediately.

115159 *A.11.1.8 Writing Data and Output Processing*

115160 There is no additional rationale provided for this section.

115161 *A.11.1.9 Special Characters*

115162 There is no additional rationale provided for this section.

115163 *A.11.1.10 Modem Disconnect*

115164 There is no additional rationale provided for this section.

115165 *A.11.1.11 Closing a Terminal Device File*115166 POSIX.1-200x does not specify that a *close()* on a terminal device file include the equivalent of a  
115167 call to *tcflow(fd,TCOON)*.115168 An implementation that discards output at the time *close()* is called after reporting the return  
115169 value to the *write()* call that data was written does not conform with POSIX.1-200x. An  
115170 application has functions such as *tcdrain()*, *tcflush()*, and *tcflow()* available to obtain the detailed  
115171 behavior it requires with respect to flushing of output.115172 At the time of the last close on a terminal device, an application relinquishes any ability to exert  
115173 flow control via *tcflow()*.115174 **A.11.2 Parameters that Can be Set**115175 *A.11.2.1 The termios Structure*115176 This structure is part of an interface that, in general, retains the historic grouping of flags.  
115177 Although a more optimal structure for implementations may be possible, the degree of change  
115178 to applications would be significantly larger.115179 *A.11.2.2 Input Modes*115180 Some historical implementations treated a long break as multiple events, as many as one per  
115181 character time. The wording in POSIX.1 explicitly prohibits this.115182 Although the ISTRIP flag is normally superfluous with today's terminal hardware and software,  
115183 it is historically supported. Therefore, applications may be using ISTRIP, and there is no  
115184 technical problem with supporting this flag. Also, applications may wish to receive only 7-bit  
115185 input bytes and may not be connected directly to the hardware terminal device (for example,  
115186 when a connection traverses a network).115187 Also, there is no requirement in general that the terminal device ensures that high-order bits  
115188 beyond the specified character size are cleared. ISTRIP provides this function for 7-bit  
115189 characters, which are common.115190 In dealing with multi-byte characters, the consequences of a parity error in such a character, or  
115191 in an escape sequence affecting the current character set, are beyond the scope of POSIX.1 and  
115192 are best dealt with by the application processing the multi-byte characters.

115193 A.11.2.3 *Output Modes*

115194 POSIX.1 does not describe postprocessing of output to a terminal or detailed control of that from  
 115195 a conforming application. (That is, translation of <newline> to <carriage-return> followed by  
 115196 <linefeed> or <tab> processing.) There is nothing that a conforming application should do to its  
 115197 output for a terminal because that would require knowledge of the operation of the terminal. It  
 115198 is the responsibility of the operating system to provide postprocessing appropriate to the output  
 115199 device, whether it is a terminal or some other type of device.

115200 Extensions to POSIX.1 to control the type of postprocessing already exist and are expected to  
 115201 continue into the future. The control of these features is primarily to adjust the interface between  
 115202 the system and the terminal device so the output appears on the display correctly. This should  
 115203 be set up before use by any application.

115204 In general, both the input and output modes should not be set absolutely, but rather modified  
 115205 from the inherited state.

115206 A.11.2.4 *Control Modes*

115207 This section could be misread that the symbol "CSIZE" is a title in the **termios** *c\_flag* field.  
 115208 Although it does serve that function, it is also a required symbol, as a literal reading of POSIX.1  
 115209 (and the caveats about typography) would indicate.

115210 A.11.2.5 *Local Modes*

115211 Non-canonical mode is provided to allow fast bursts of input to be read efficiently while still  
 115212 allowing single-character input.

115213 The ECHONL function historically has been in many implementations. Since there seems to be  
 115214 no technical problem with supporting ECHONL, it is included in POSIX.1 to increase consensus.

115215 The alternate behavior possible when ECHOK or ECHOE are specified with ICANON is  
 115216 permitted as a compromise depending on what the actual terminal hardware can do. Erasing  
 115217 characters and lines is preferred, but is not always possible.

115218 A.11.2.6 *Special Control Characters*

115219 Permitting VMIN and VTIME to overlap with VEOF and VEOL was a compromise for historical  
 115220 implementations. Only when backwards-compatibility of object code is a serious concern to an  
 115221 implementor should an implementation continue this practice. Correct applications that work  
 115222 with the overlap (at the source level) should also work if it is not present, but not the reverse.

115223 **A.12 Utility Conventions**115224 **A.12.1 Utility Argument Syntax**

115225 The standard developers considered that recent trends toward diluting the SYNOPSIS sections  
 115226 of historical reference pages to the equivalent of:

115227 `command [options][operands]`

115228 were a disservice to the reader. Therefore, considerable effort was placed into rigorous  
 115229 definitions of all the command line arguments and their interrelationships. The relationships  
 115230 depicted in the synopses are normative parts of POSIX.1-200x; this information is sometimes  
 115231 repeated in textual form, but that is only for clarity within context.



115232 The use of “undefined” for conflicting argument usage and for repeated usage of the same  
 115233 option is meant to prevent conforming applications from using conflicting arguments or  
 115234 repeated options unless specifically allowed (as is the case with *ls*, which allows simultaneous,  
 115235 repeated use of the *-C*, *-l*, and *-1* options). Many historical implementations will tolerate this  
 115236 usage, choosing either the first or the last applicable argument. This tolerance can continue, but  
 115237 conforming applications cannot rely upon it. (Other implementations may choose to print usage  
 115238 messages instead.)

115239 The use of “undefined” for conflicting argument usage also allows an implementation to make  
 115240 reasonable extensions to utilities where the implementor considers mutually-exclusive options  
 115241 according to POSIX.1-200x to have a sensible meaning and result.

115242 POSIX.1-200x does not define the result of a command when an option-argument or operand is  
 115243 not followed by ellipses and the application specifies more than one of that option-argument or  
 115244 operand. This allows an implementation to define valid (although non-standard) behavior for  
 115245 the utility when more than one such option or operand is specified.

115246 The requirements for option-arguments are summarized as follows:

|                                      | SYNOPSIS Shows:                |                            |
|--------------------------------------|--------------------------------|----------------------------|
|                                      | <i>-a arg</i>                  | <i>-c[arg]</i>             |
| Conforming application uses:         | <i>-a arg</i>                  | <i>-carg</i> or <i>-c</i>  |
| System supports:                     | <i>-a arg</i> and <i>-aarg</i> | <i>-carg</i> and <i>-c</i> |
| Non-conforming applications may use: | <i>-aarg</i>                   | N/A                        |

115252 Earlier versions of this standard included obsolescent syntax which showed some options with  
 115253 (mandatory) adjacent option-arguments in the SYNOPSIS for some utilities. These have since  
 115254 been removed. For all options with mandatory option-arguments, the SYNOPSIS now shows  
 115255 <blank>s between the option and the option-argument; however, historical usage has not been  
 115256 consistent in this area; therefore, <blank>s are required to be used by conforming applications  
 115257 and to be handled by all implementations, but implementations are also required to handle an  
 115258 adjacent option-argument in order to preserve backwards-compatibility for old scripts. One of  
 115259 the justifications for selecting the multiple-argument method was that the single-argument case  
 115260 is inherently ambiguous when the option-argument can legitimately be a null string.

115261 POSIX.1-200x explicitly states that digits are permitted as operands and option-arguments. The  
 115262 lower and upper bounds for the values of the numbers used for operands and option-arguments  
 115263 were derived from the ISO C standard values for {LONG\_MIN} and {LONG\_MAX}. The  
 115264 requirement on the standard utilities is that numbers in the specified range do not cause a  
 115265 syntax error, although the specification of a number need not be semantically correct for a  
 115266 particular operand or option-argument of a utility. For example, the specification of:

115267 `dd obs=3000000000`

115268 would yield undefined behavior for the application and could be a syntax error because the  
 115269 number 3 000 000 000 is outside of the range  $-2\ 147\ 483\ 647$  to  $+2\ 147\ 483\ 647$ . On the other hand:

115270 `dd obs=2000000000`

115271 may cause some error, such as “blocksize too large”, rather than a syntax error.

### 115272 A.12.2 Utility Syntax Guidelines

115273 This section is based on the rules listed in the SVID. It was included for two reasons:

- 115274 1. The individual utility descriptions in XCU [Chapter 4](#) (on page 2343) needed a set of  
115275 common (although not universal) actions on which they could anchor their descriptions  
115276 of option and operand syntax. Most of the standard utilities actually do use these  
115277 guidelines, and many of their historical implementations use the *getopt()* function for  
115278 their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.
- 115279 2. Developers of conforming applications need suggested guidelines if the POSIX  
115280 community is to avoid the chaos of historical UNIX system command syntax.

115281 It is recommended that all *future* utilities and applications use these guidelines to enhance “user  
115282 portability”. The fact that some historical utilities could not be changed (to avoid breaking  
115283 historical applications) should not deter this future goal.

115284 The voluntary nature of the guidelines is highlighted by repeated uses of the word *should*  
115285 throughout. This usage should not be misinterpreted to imply that utilities that claim  
115286 conformance in their OPTIONS sections do not always conform.

115287 Guidelines 1 and 2 encourage utility writers to use only characters from the portable character  
115288 set because use of locale-specific characters may make the utility inaccessible from other locales.  
115289 Use of uppercase letters is discouraged due to problems associated with porting utilities to  
115290 systems that do not distinguish between uppercase and lowercase characters in filenames. Use  
115291 of non-alphanumeric characters is discouraged due to the number of utilities that treat non-  
115292 alphanumeric characters in “special” ways depending on context (such as the shell using  
115293 whitespace characters to delimit arguments, various quote characters for quoting, the dollar sign  
115294 to introduce variable expansion, etc.).

115295 In XCU [Section 2.9.1](#) (on page 2263), it is further stated that a command used in the Shell  
115296 Command Language cannot be named with a trailing colon.

115297 Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the  
115298 character set to allow compatibility with historical usage. Historical practice allows the use of  
115299 digits wherever practical, and there are no portability issues that would prohibit the use of  
115300 digits. In fact, from an internationalization viewpoint, digits (being non-language-dependent)  
115301 are preferable over letters (a `-2` is intuitively self-explanatory to any user, while in the `-f filename`  
115302 the letter ‘`f`’ is a mnemonic aid only to speakers of Latin-based languages where “filename”  
115303 happens to translate to a word that begins with ‘`f`’). Since Guideline 3 still retains the word  
115304 “single”, multi-digit options are not allowed. Instances of historical utilities that used them have  
115305 been marked obsolescent, with the numbers being changed from option names to option-  
115306 arguments.

115307 It was difficult to achieve a satisfactory solution to the problem of name space in option  
115308 characters. When the standard developers desired to extend the historical `cc` utility to accept  
115309 ISO C standard programs, they found that all of the portable alphabet was already in use by  
115310 various vendors. Thus, they had to devise a new name, `c89` (now superseded by `c99`), rather than  
115311 something like `cc -X`. There were suggestions that implementors be restricted to providing  
115312 extensions through various means (such as using a plus sign as the option delimiter or using  
115313 option characters outside the alphanumeric set) that would reserve all of the remaining  
115314 alphanumeric characters for future POSIX standards. These approaches were resisted because  
115315 they lacked the historical style of UNIX systems. Furthermore, if a vendor-provided option  
115316 should become commonly used in the industry, it would be a candidate for standardization. It  
115317 would be desirable to standardize such a feature using historical practice for the syntax (the  
115318 semantics can be standardized with any syntax). This would not be possible if the syntax was  
115319 one reserved for the vendor. However, since the standardization process may lead to minor

115320 changes in the semantics, it may prove to be better for a vendor to use a syntax that will not be  
115321 affected by standardization.

115322 Guideline 8 includes the concept of comma-separated lists in a single argument. It is up to the  
115323 utility to parse such a list itself because *getopt()* just returns the single string. This situation was  
115324 retained so that certain historical utilities would not violate the guidelines. Applications  
115325 preparing for international use should be aware of an occasional problem with comma-  
115326 separated lists: in some locales, the comma is used as the radix character. Thus, if an application  
115327 is preparing operands for a utility that expects a comma-separated list, it should avoid  
115328 generating non-integer values through one of the means that is influenced by setting the  
115329 *LC\_NUMERIC* variable (such as *awk*, *bc*, *printf*, or *printf()*).

115330 Unless explicitly stated otherwise in the utility description, Guideline 9 requires applications to  
115331 put options before operands, and requires utilities to accept any such usage without  
115332 misinterpreting operands as options. For example, if an implementation of the *printf* utility  
115333 supports a *-e* option as an extension, the command:

```
115334 printf %s -e
```

115335 must output the string "-e" without interpreting the *-e* as an option. Similarly, the command:

```
115336 ls myfile -l
```

115337 must interpret the *-l* argument as a second file operand, not as a *-l* option.

115338 Applications calling any utility with a first operand starting with '-' should usually specify *--*,  
115339 as indicated by Guideline 10, to mark the end of the options. This is true even if the SYNOPSIS  
115340 in the Shell and Utilities volume of POSIX.1-200x does not specify any options; implementations  
115341 may provide options as extensions to the Shell and Utilities volume of POSIX.1-200x. The  
115342 standard utilities that do not support Guideline 10 indicate that fact in the OPTIONS section of  
115343 the utility description.

115344 Guideline 7 allows any string to be an option-argument; an option-argument can begin with any  
115345 character, can be *-* or *--*, and can be an empty string. For example, the commands *pr -h -*, *pr -h*  
115346 *--*, *pr -h -d*, *pr -h +2*, and *pr -h ''* contain the option-arguments *-*, *--*, *-d*, *+2*, and an empty  
115347 string, respectively. Conversely, the command *pr -h -- -d* treats *-d* as an option, not as an  
115348 argument, because the *--* is an option-argument here, not a delimiter.

115349 Guideline 11 was modified to clarify that the order of different options should not matter  
115350 relative to one another. However, the order of repeated options that also have option-arguments  
115351 may be significant; therefore, such options are required to be interpreted in the order that they  
115352 are specified. The *make* utility is an instance of a historical utility that uses repeated options in  
115353 which the order is significant. Multiple files are specified by giving multiple instances of the *-f*  
115354 option; for example:

```
115355 make -f common_header -f specific_rules target
```

115356 Guideline 13 does not imply that all of the standard utilities automatically accept the operand  
115357 '-' to mean standard input or output, nor does it specify the actions of the utility upon  
115358 encountering multiple '-' operands. It simply says that, by default, '-' operands are not used  
115359 for other purposes in the file reading or writing (but not when using *stat()*, *unlink()*, *touch*, and  
115360 so on) utilities. In earlier versions of this standard, all information concerning actual treatment of  
115361 the '-' operand is found in the individual utility sections. Many implementations, however,  
115362 treated '-' as standard input or output and many applications depended on this behavior even  
115363 though it was not standard. This behavior is now implementation-defined. Portable applications  
115364 should not use '-' to mean standard input or output unless it is explicitly stated to do so in the  
115365 utility description and they should always use './-' if they intend to refer to a file named '-'  
115366 in the current working directory.

115367 Guideline 14 is intended to prohibit implementations that would treat the command `ls -l -d` as if  
115368 it were `ls -- -l -d` or `ls -l -- -d`.

115369 The standard permits implementations to have extensions that violate the Utility Syntax  
115370 Guidelines so long as when the utility is used in line with the forms defined by the standard it  
115371 follows the Utility Syntax Guidelines. Thus, `head-42file` and `ls--help` are permitted extensions.  
115372 The intent is to allow extensions so long as the standard form is accepted and follows the  
115373 guidelines.

115374 An area of concern was that as implementations mature, implementation-defined utilities and  
115375 implementation-defined utility options will result. The idea was expressed that there needed to  
115376 be a standard way, say an environment variable or some such mechanism, to identify  
115377 implementation-defined utilities separately from standard utilities that may have the same  
115378 name. It was decided that there already exist several ways of dealing with this situation and that  
115379 it is outside of the scope to attempt to standardize in the area of non-standard items. A method  
115380 that exists on some historical implementations is the use of the so-called `/local/bin` or  
115381 `/usr/local/bin` directory to separate local or additional copies or versions of utilities. Another  
115382 method that is also used is to isolate utilities into completely separate domains. Still another  
115383 method to ensure that the desired utility is being used is to request the utility by its full  
115384 pathname. There are many approaches to this situation; the examples given above serve to  
115385 illustrate that there is more than one.

## 115386 A.13 Headers

### 115387 A.13.1 Format of Entries

115388 Each header reference page has a common layout of sections describing the interface. This  
115389 layout is similar to the manual page or “man” page format shipped with most UNIX systems,  
115390 and each header has sections describing the SYNOPSIS and DESCRIPTION. These are the two  
115391 sections that relate to conformance.

115392 Additional sections are informative, and add considerable information for the application  
115393 developer. APPLICATION USAGE sections provide additional caveats, issues, and  
115394 recommendations to the developer. RATIONALE sections give additional information on the  
115395 decisions made in defining the interface.

115396 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
115397 the future, and often cautions the developer to architect the code to account for a change in this  
115398 area. Note that a future directions statement should not be taken as a commitment to adopt a  
115399 feature or interface in the future.

115400 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
115401 changed.

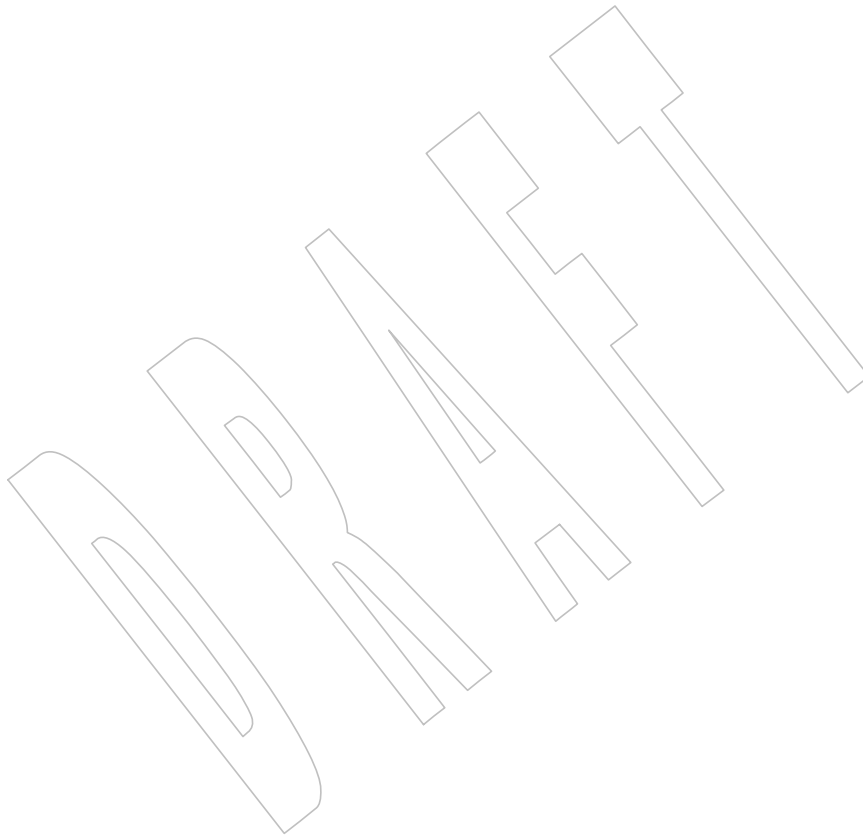
115402 Option labels and margin markings in the page can be useful in guiding the application  
115403 developer.

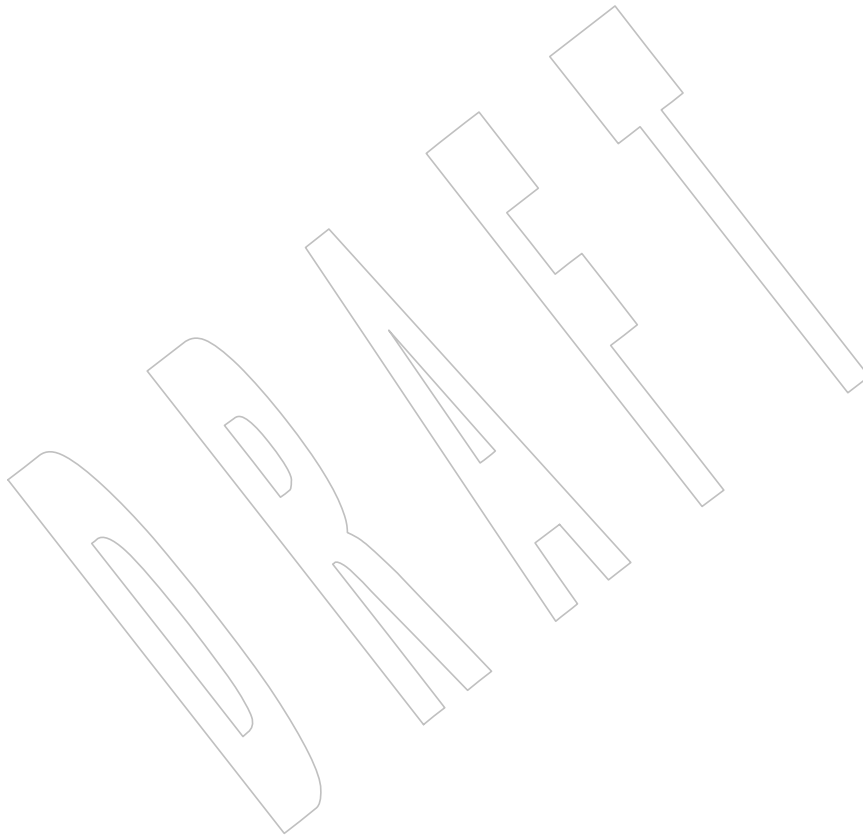
115404 **A.13.2 Removed Headers in Issue 7**

115405 The headers removed in Issue 7 (from the Issue 6 base document) are as follows:

115406 

| Removed Headers in Issue 7       |                                 |
|----------------------------------|---------------------------------|
| <code>&lt;sys/timeb.h&gt;</code> | <code>&lt;ucontext.h&gt;</code> |

115407



115408

**/** *Rationale (Informative)*

115409

**Part B:**

115410

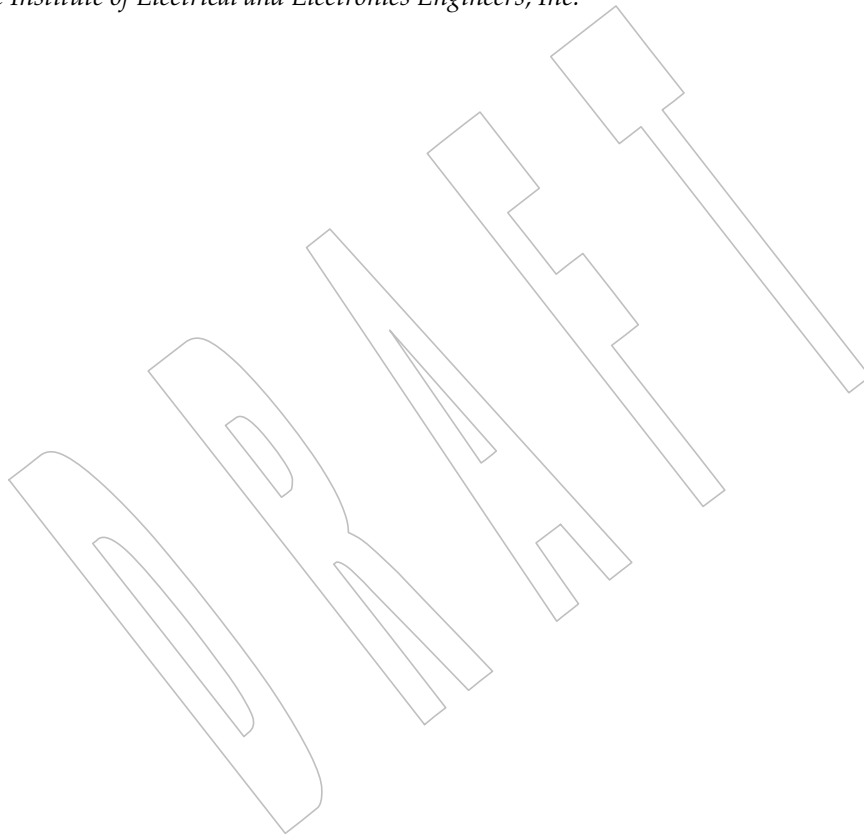
**System Interfaces**

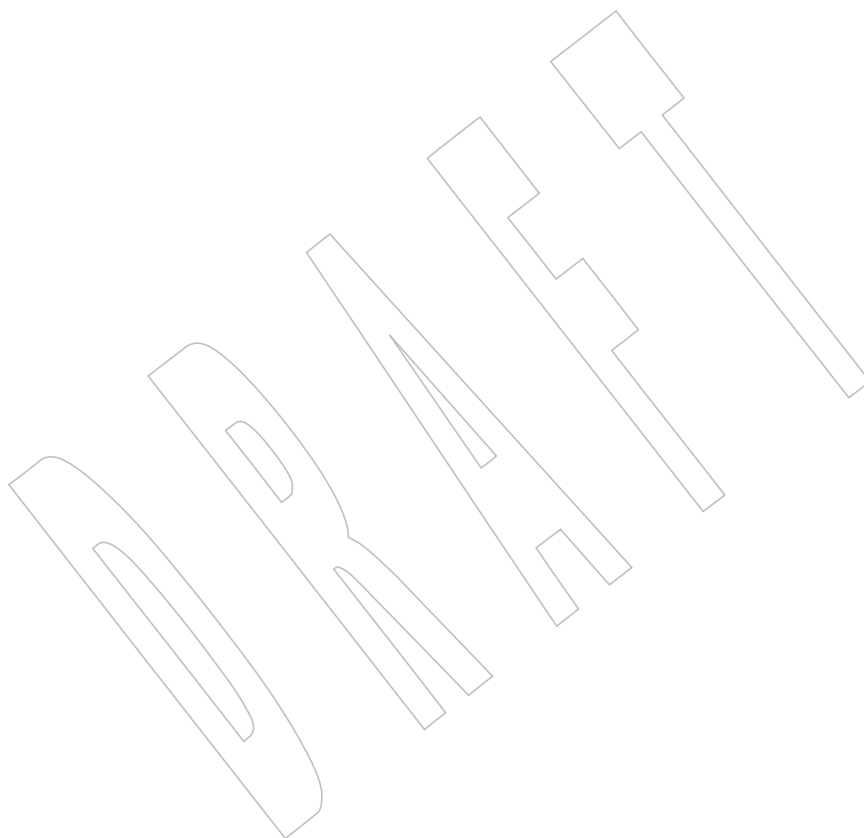
115411

*The Open Group*

115412

*The Institute of Electrical and Electronics Engineers, Inc.*







*Rationale for System Interfaces*

115413

115414

115415

**B.1 Introduction**

115416

115417

**B.1.1 Change History**

115418

The change history is provided as an informative section, to track changes from earlier versions of this standard.

115419

115420

The following sections describe changes made to the System Interfaces volume of POSIX.1-200x since Issue 6 of the base document. The CHANGE HISTORY section for each entry details the technical changes that have been made to that entry from Issue 5. Changes between earlier versions of the base document and Issue 5 are not included.

115421

115422

115423

115424

**Changes from Issue 6 to Issue 7 (POSIX.1-200x)**

115425

The following list summarizes the major changes that were made in the System Interfaces volume of POSIX.1-200x from Issue 6 to Issue 7:

115426

115427

- The Open Group Technical Standard, 2006, Extended API Set Part 1 is incorporated.

115428

- The Open Group Technical Standard, 2006, Extended API Set Part 2 is incorporated.

115429

- The Open Group Technical Standard, 2006, Extended API Set Part 3 is incorporated.

115430

- The Open Group Technical Standard, 2006, Extended API Set Part 4 is incorporated.

115431

- Existing functionality is aligned with ISO/IEC 9899:1999, Programming Languages — C, ISO/IEC 9899:1999/Cor.2:2004(E)

115432

115433

- Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses to ISO/IEC defect reports against ISO/IEC 9945 are applied.

115434

115435

- The Open Group corrigenda and resolutions are applied.

115436

- Features, marked legacy or obsolescent in the base document, have been considered for removal in this version.

115437

115438

- The options within the standard have been revised.

115439 **New Features in Issue 7**

115440 The functions first introduced in Issue 7 (over the Issue 6 base document) are as follows:

| New Functions in Issue 7 |                      |                                      |
|--------------------------|----------------------|--------------------------------------|
| 115441                   |                      |                                      |
| 115442                   | <i>alphasort()</i>   | <i>iswctype_l()</i>                  |
| 115443                   | <i>dirfd()</i>       | <i>iswdigit_l()</i>                  |
| 115444                   | <i>dprintf()</i>     | <i>iswgraph_l()</i>                  |
| 115445                   | <i>duplocale()</i>   | <i>iswlower_l()</i>                  |
| 115446                   | <i>faccessat()</i>   | <i>iswprint_l()</i>                  |
| 115447                   | <i>fchmodat()</i>    | <i>iswpunct_l()</i>                  |
| 115448                   | <i>fchownat()</i>    | <i>iswspace_l()</i>                  |
| 115449                   | <i>fdopendir()</i>   | <i>iswupper_l()</i>                  |
| 115450                   | <i>fexecve()</i>     | <i>iswxdigit_l()</i>                 |
| 115451                   | <i>fnmopen()</i>     | <i>isxdigit_l()</i>                  |
| 115452                   | <i>freelocale()</i>  | <i>linkat()</i>                      |
| 115453                   | <i>fstatat()</i>     | <i>mbsnrto wcs()</i>                 |
| 115454                   | <i>futimens()</i>    | <i>mkdirat()</i>                     |
| 115455                   | <i>getdelim()</i>    | <i>mkdtemp()</i>                     |
| 115456                   | <i>getline()</i>     | <i>mkffloat()</i>                    |
| 115457                   | <i>isalnum_l()</i>   | <i>mknodat()</i>                     |
| 115458                   | <i>isalpha_l()</i>   | <i>newlocale()</i>                   |
| 115459                   | <i>isblank_l()</i>   | <i>openat()</i>                      |
| 115460                   | <i>iscntrl_l()</i>   | <i>open_memstream()</i>              |
| 115461                   | <i>isdigit_l()</i>   | <i>psiginfo()</i>                    |
| 115462                   | <i>isgraph_l()</i>   | <i>psignal()</i>                     |
| 115463                   | <i>islower_l()</i>   | <i>pthread_mutexattr_getrobust()</i> |
| 115464                   | <i>isprint_l()</i>   | <i>pthread_mutexattr_setrobust()</i> |
| 115465                   | <i>ispunct_l()</i>   | <i>pthread_mutex_consistent()</i>    |
| 115466                   | <i>isspace_l()</i>   | <i>readlinkat()</i>                  |
| 115467                   | <i>isupper_l()</i>   | <i>renameat()</i>                    |
| 115468                   | <i>iswalnum_l()</i>  | <i>scandir()</i>                     |
| 115469                   | <i>iswalphal_l()</i> | <i>stpncpy()</i>                     |
| 115470                   | <i>iswblank_l()</i>  | <i>stpncpy()</i>                     |
| 115471                   | <i>iswcntrl_l()</i>  | <i>strcasecmp_l()</i>                |
|                          |                      | <i>strcoll_l()</i>                   |
|                          |                      | <i>strfmon_l()</i>                   |
|                          |                      | <i>strncasecmp_l()</i>               |
|                          |                      | <i>strndup()</i>                     |
|                          |                      | <i>strlen()</i>                      |
|                          |                      | <i>strsignal()</i>                   |
|                          |                      | <i>strxfrm_l()</i>                   |
|                          |                      | <i>symlinkat()</i>                   |
|                          |                      | <i>tolower_l()</i>                   |
|                          |                      | <i>toupper_l()</i>                   |
|                          |                      | <i>towctrans_l()</i>                 |
|                          |                      | <i>towlower()</i>                    |
|                          |                      | <i>towupper()</i>                    |
|                          |                      | <i>unlinkat()</i>                    |
|                          |                      | <i>uselocale()</i>                   |
|                          |                      | <i>utimensat()</i>                   |
|                          |                      | <i>wcpcpy()</i>                      |
|                          |                      | <i>wcpncpy()</i>                     |
|                          |                      | <i>wcscasecmp()</i>                  |
|                          |                      | <i>wcscasecmp_l()</i>                |
|                          |                      | <i>wcscoll_l()</i>                   |
|                          |                      | <i>wcsdup()</i>                      |
|                          |                      | <i>wcsncasecmp()</i>                 |
|                          |                      | <i>wcsncasecmp_l()</i>               |
|                          |                      | <i>wcsnlen()</i>                     |
|                          |                      | <i>wcsnrto mbs()</i>                 |
|                          |                      | <i>wcsxfrm_l()</i>                   |
|                          |                      | <i>wctrans_l()</i>                   |
|                          |                      | <i>wctype_l()</i>                    |

115472 **Newly Mandated Functions in Issue 7**115473 The functions that were previously part of an option group but are now mandatory in Issue 7 are  
115474 as follows:

115475

## Newly Mandated Functions in Issue 7

|        |                                         |                                      |                                     |
|--------|-----------------------------------------|--------------------------------------|-------------------------------------|
| 115476 | <i>aio_cancel()</i>                     | <i>pthread_atfork()</i>              | <i>pthread_rwlock_tryrdlock()</i>   |
| 115477 | <i>aio_error()</i>                      | <i>pthread_attr_destroy()</i>        | <i>pthread_rwlock_trywrlock()</i>   |
| 115478 | <i>aio_fsync()</i>                      | <i>pthread_attr_getdetachstate()</i> | <i>pthread_rwlock_unlock()</i>      |
| 115479 | <i>aio_read()</i>                       | <i>pthread_attr_getguardsize()</i>   | <i>pthread_rwlock_wrlock()</i>      |
| 115480 | <i>aio_return()</i>                     | <i>pthread_attr_getschedparam()</i>  | <i>pthread_rwlockattr_destroy()</i> |
| 115481 | <i>aio_suspend()</i>                    | <i>pthread_attr_init()</i>           | <i>pthread_rwlockattr_init()</i>    |
| 115482 | <i>aio_write()</i>                      | <i>pthread_attr_setdetachstate()</i> | <i>pthread_self()</i>               |
| 115483 | <i>asctime_r()</i>                      | <i>pthread_attr_setguardsize()</i>   | <i>pthread_setcancelstate()</i>     |
| 115484 | <i>catclose()</i>                       | <i>pthread_attr_setschedparam()</i>  | <i>pthread_setcanceltype()</i>      |
| 115485 | <i>catgets()</i>                        | <i>pthread_barrier_destroy()</i>     | <i>pthread_setspecific()</i>        |
| 115486 | <i>catopen()</i>                        | <i>pthread_barrier_init()</i>        | <i>pthread_spin_destroy()</i>       |
| 115487 | <i>clock_getres()</i>                   | <i>pthread_barrier_wait()</i>        | <i>pthread_spin_init()</i>          |
| 115488 | <i>clock_gettime()</i>                  | <i>pthread_barrierattr_destroy()</i> | <i>pthread_spin_lock()</i>          |
| 115489 | <i>clock_nanosleep()</i>                | <i>pthread_barrierattr_init()</i>    | <i>pthread_spin_trylock()</i>       |
| 115490 | <i>clock_settime()</i>                  | <i>pthread_cancel()</i>              | <i>pthread_spin_unlock()</i>        |
| 115491 | <i>ctime_r()</i>                        | <i>pthread_cleanup_pop()</i>         | <i>pthread_testcancel()</i>         |
| 115492 | <i>dlclose()</i>                        | <i>pthread_cleanup_push()</i>        | <i>putc_unlocked()</i>              |
| 115493 | <i>dlderror()</i>                       | <i>pthread_cond_broadcast()</i>      | <i>putchar_unlocked()</i>           |
| 115494 | <i>dlopen()</i>                         | <i>pthread_cond_destroy()</i>        | <i>pwrite()</i>                     |
| 115495 | <i>dlsym()</i>                          | <i>pthread_cond_init()</i>           | <i>rand_r()</i>                     |
| 115496 | <i>fehdir()</i>                         | <i>pthread_cond_signal()</i>         | <i>readdir_r()</i>                  |
| 115497 | <i>flockfile()</i>                      | <i>pthread_cond_timedwait()</i>      | <i>sem_close()</i>                  |
| 115498 | <i>fstatvfs()</i>                       | <i>pthread_cond_wait()</i>           | <i>sem_destroy()</i>                |
| 115499 | <i>ftrylockfile()</i>                   | <i>pthread_condattr_destroy()</i>    | <i>sem_getvalue()</i>               |
| 115500 | <i>funlockfile()</i>                    | <i>pthread_condattr_getclock()</i>   | <i>sem_init()</i>                   |
| 115501 | <i>getc_unlocked()</i>                  | <i>pthread_condattr_init()</i>       | <i>sem_open()</i>                   |
| 115502 | <i>getchar_unlocked()</i>               | <i>pthread_condattr_setclock()</i>   | <i>sem_post()</i>                   |
| 115503 | <i>getgrgid_r()</i>                     | <i>pthread_create()</i>              | <i>sem_timedwait()</i>              |
| 115504 | <i>getgrnam_r()</i>                     | <i>pthread_detach()</i>              | <i>sem_trywait()</i>                |
| 115505 | <i>getlogin_r()</i>                     | <i>pthread_equal()</i>               | <i>sem_unlink()</i>                 |
| 115506 | <i>getpgid()</i>                        | <i>pthread_exit()</i>                | <i>sem_wait()</i>                   |
| 115507 | <i>getpwnam_r()</i>                     | <i>pthread_getspecific()</i>         | <i>sigqueue()</i>                   |
| 115508 | <i>getpwuid_r()</i>                     | <i>pthread_join()</i>                | <i>sigqueue()</i>                   |
| 115509 | <i>getsid()</i>                         | <i>pthread_key_create()</i>          | <i>sigtimedwait()</i>               |
| 115510 | <i>getsubopt()</i>                      | <i>pthread_key_delete()</i>          | <i>sigwaitinfo()</i>                |
| 115511 | <i>gmtime_r()</i>                       | <i>pthread_mutex_destroy()</i>       | <i>statvfs()</i>                    |
| 115512 | <i>iconv()</i>                          | <i>pthread_mutex_init()</i>          | <i>strcasemp()</i>                  |
| 115513 | <i>iconv_close()</i>                    | <i>pthread_mutex_lock()</i>          | <i>strdup()</i>                     |
| 115514 | <i>iconv_open()</i>                     | <i>pthread_mutex_timedlock()</i>     | <i>strerror_r()</i>                 |
| 115515 | <i>lchown()</i>                         | <i>pthread_mutex_trylock()</i>       | <i>strfnon()</i>                    |
| 115516 | <i>lio_listio()</i>                     | <i>pthread_mutex_unlock()</i>        | <i>strncasemp()</i>                 |
| 115517 | <i>localtime_r()</i>                    | <i>pthread_mutexattr_destroy()</i>   | <i>strtok_r()</i>                   |
| 115518 | <i>mkstemp()</i>                        | <i>pthread_mutexattr_gettype()</i>   | <i>tcgetsid()</i>                   |
| 115519 | <i>mmap()</i>                           | <i>pthread_mutexattr_init()</i>      | <i>timer_create()</i>               |
| 115520 | <i>mprotect()</i>                       | <i>pthread_mutexattr_settype()</i>   | <i>timer_delete()</i>               |
| 115521 | <i>munmap()</i>                         | <i>pthread_once()</i>                | <i>timer_getoverrun()</i>           |
| 115522 | <i>nanosleep()</i>                      | <i>pthread_rwlock_destroy()</i>      | <i>timer_gettime()</i>              |
| 115523 | <i>nl_langinfo()</i>                    | <i>pthread_rwlock_init()</i>         | <i>timer_settime()</i>              |
| 115524 | <i>poll()</i>                           | <i>pthread_rwlock_rdlock()</i>       | <i>truncate()</i>                   |
| 115525 | <i>posix_trace_timedgetnext_event()</i> | <i>pthread_rwlock_timedrdlock()</i>  | <i>ttyname_r()</i>                  |
| 115526 | <i>pread()</i>                          | <i>pthread_rwlock_timedwrlock()</i>  | <i>waitid()</i>                     |

115527 **Obsolescent Functions in Issue 7**

115528 The base functions moved to obsolescent status in Issue 7 (from the Issue 6 base document) are  
 115529 as follows:

| 115530 <b>Obsolescent Base Functions in Issue 7</b> |                 |
|-----------------------------------------------------|-----------------|
| 115531 <i>asctime()</i>                             | <i>gets()</i>   |
| 115532 <i>asctime_r()</i>                           | <i>rand_r()</i> |
| 115533 <i>ctime()</i>                               | <i>tmpnam()</i> |
| 115534 <i>ctime_r()</i>                             |                 |

115535 The XSI functions moved to obsolescent status in Issue 7 (from the Issue 6 base document) are as  
 115536 follows:

| 115537 <b>Obsolescent XSI Functions in Issue 7</b> |                                 |                       |
|----------------------------------------------------|---------------------------------|-----------------------|
| 115538 <i>_longjmp()</i>                           | <i>pthread_getconcurrency()</i> | <i>sigset()</i>       |
| 115539 <i>_setjmp()</i>                            | <i>pthread_setconcurrency()</i> | <i>siginterrupt()</i> |
| 115540 <i>_tolower()</i>                           | <i>setitimer()</i>              | <i>tempnam()</i>      |
| 115541 <i>_toupper()</i>                           | <i>setpgrp()</i>                | <i>toascii()</i>      |
| 115542 <i>ftw()</i>                                | <i>sighold()</i>                | <i>ulimit()</i>       |
| 115543 <i>getitimer()</i>                          | <i>sigignore()</i>              | <i>utime()</i>        |
| 115544 <i>gettimeofday()</i>                       | <i>sigpause()</i>               |                       |
| 115545 <i>isascii()</i>                            | <i>sigrelse()</i>               |                       |

115546 **Removed Functions and Symbols in Issue 7**

115547 The functions and symbols removed in Issue 7 (from the Issue 6 base document) are as follows:

| 115548 <b>Removed Functions and Symbols in Issue 7</b> |                                    |                      |
|--------------------------------------------------------|------------------------------------|----------------------|
| 115549 <i>bcmp()</i>                                   | <i>gethostbyaddr()</i>             | <i>rindex()</i>      |
| 115550 <i>bcopy()</i>                                  | <i>gethostbyname()</i>             | <i>scalb()</i>       |
| 115551 <i>bsd_signal()</i>                             | <i>getwd()</i>                     | <i>setcontext()</i>  |
| 115552 <i>bzero()</i>                                  | <i>h_errno</i>                     | <i>swapcontext()</i> |
| 115553 <i>ecvt()</i>                                   | <i>index()</i>                     | <i>ualarm()</i>      |
| 115554 <i>fcvt()</i>                                   | <i>makecontext()</i>               | <i>usleep()</i>      |
| 115555 <i>ftime()</i>                                  | <i>mktemp()</i>                    | <i>vfork()</i>       |
| 115556 <i>gcvt()</i>                                   | <i>pthread_attr_getstackaddr()</i> | <i>wcs wcs()</i>     |
| 115557 <i>getcontext()</i>                             | <i>pthread_attr_setstackaddr()</i> |                      |

115558 **B.1.2 Relationship to Other Formal Standards**

115559 There is no additional rationale provided for this section.

115560 **B.1.3 Format of Entries**

115561 Each system interface reference page has a common layout of sections describing the interface.  
 115562 This layout is similar to the manual page or “man” page format shipped with most UNIX  
 115563 systems, and each header has sections describing the SYNOPSIS, DESCRIPTION, RETURN  
 115564 VALUE, and ERRORS. These are the four sections that relate to conformance.

115565 Additional sections are informative, and add considerable information for the application  
 115566 developer. EXAMPLES sections provide example usage. APPLICATION USAGE sections  
 115567 provide additional caveats, issues, and recommendations to the developer. RATIONALE  
 115568 sections give additional information on the decisions made in defining the interface.

115569 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
 115570 the future, and often cautions the developer to architect the code to account for a change in this  
 115571 area. Note that a future directions statement should not be taken as a commitment to adopt a  
 115572 feature or interface in the future.

115573 The CHANGE HISTORY section describes when the interface was introduced, and how it has  
 115574 changed.

115575 Option labels and margin markings in the page can be useful in guiding the application  
 115576 developer.

## 115577 B.2 General Information

### 115578 B.2.1 Use and Implementation of Functions

115579 The information concerning the use of functions was adapted from a description in the ISO C  
 115580 standard. Here is an example of how an application program can protect itself from functions  
 115581 that may or may not be macros, rather than true functions:

115582 The *atoi()* function may be used in any of several ways:

- 115583 • By use of its associated header (possibly generating a macro expansion):

```
115584 #include <stdlib.h>
115585 /* ... */
115586 i = atoi(str);
```

- 115587 • By use of its associated header (assuredly generating a true function call):

```
115588 #include <stdlib.h>
115589 #undef atoi
115590 /* ... */
115591 i = atoi(str);
```

115592 or:

```
115593 #include <stdlib.h>
115594 /* ... */
115595 i = (atoi) (str);
```

- 115596 • By explicit declaration:

```
115597 extern int atoi (const char *);
115598 /* ... */
115599 i = atoi(str);
```

- 115600 • By implicit declaration:

```
115601 /* ... */
115602 i = atoi(str);
```

115603 (Assuming no function prototype is in scope. This is not allowed by the ISO C standard for  
 115604 functions with variable arguments; furthermore, parameter type conversion “widening” is  
 115605 subject to different rules in this case.)

115606 Note that the ISO C standard reserves names starting with ‘\_’ for the compiler. Therefore, the  
 115607 compiler could, for example, implement an intrinsic, built-in function *\_asm\_builtin\_atoi()*, which  
 115608 it recognized and expanded into inline assembly code. Then, in **<stdlib.h>**, there could be the

115609 following:

```
115610 #define atoi(X) _asm_builtin_atoi(X)
```

115611 The user's "normal" call to `atoi()` would then be expanded inline, but the implementor would  
 115612 also be required to provide a callable function named `atoi()` for use when the application  
 115613 requires it; for example, if its address is to be stored in a function pointer variable.

## 115614 B.2.2 The Compilation Environment

### 115615 B.2.2.1 POSIX.1 Symbols

115616 This and the following section address the issue of "name space pollution". The ISO C standard  
 115617 requires that the name space beyond what it reserves not be altered except by explicit action of  
 115618 the application developer. This section defines the actions to add the POSIX.1 symbols for those  
 115619 headers where both the ISO C standard and POSIX.1 need to define symbols, and also where the  
 115620 XSI option extends the base standard.

115621 When headers are used to provide symbols, there is a potential for introducing symbols that the  
 115622 application developer cannot predict. Ideally, each header should only contain one set of  
 115623 symbols, but this is not practical for historical reasons. Thus, the concept of feature test macros is  
 115624 included. Two feature test macros are explicitly defined by POSIX.1-200x; it is expected that  
 115625 future versions may add to this.

115626 **Note:** Feature test macros allow an application to announce to the implementation its desire to have  
 115627 certain symbols and prototypes exposed. They should not be confused with the version test  
 115628 macros and constants for options in `<unistd.h>` which are the implementation's way of  
 115629 announcing functionality to the application.

115630 It is further intended that these feature test macros apply only to the headers specified by  
 115631 POSIX.1-200x. Implementations are expressly permitted to make visible symbols not specified  
 115632 by POSIX.1-200x, within both POSIX.1 and other headers, under the control of feature test  
 115633 macros that are not defined by POSIX.1-200x.

#### 115634 The `_POSIX_C_SOURCE` Feature Test Macro

115635 Since `_POSIX_SOURCE` specified by the POSIX.1-1990 standard did not have a value associated  
 115636 with it, the `_POSIX_C_SOURCE` macro replaces it, allowing an application to inform the system  
 115637 of the version of the standard to which it conforms. This symbol will allow implementations to  
 115638 support various versions of this standard simultaneously. For instance, when either  
 115639 `_POSIX_SOURCE` is defined or `_POSIX_C_SOURCE` is defined as 1, the system should make  
 115640 visible the same name space as permitted and required by the POSIX.1-1990 standard. When  
 115641 `_POSIX_C_SOURCE` is defined, the state of `_POSIX_SOURCE` is completely irrelevant.

115642 It is expected that C bindings to future POSIX standards will define new values for  
 115643 `_POSIX_C_SOURCE`, with each new value reserving the name space for that new standard, plus  
 115644 all earlier POSIX standards.

115645 **The \_XOPEN\_SOURCE Feature Test Macro**

115646 The feature test macro `_XOPEN_SOURCE` is provided as the announcement mechanism for the  
 115647 application that it requires functionality from the Single UNIX Specification. `_XOPEN_SOURCE`  
 115648 must be defined to the value 700 before the inclusion of any header to enable the functionality in  
 115649 the Single UNIX Specification. Its definition subsumes the use of `_POSIX_SOURCE` and  
 115650 `_POSIX_C_SOURCE`.

115651 An extract of code from a conforming application, that appears before any **#include** statements,  
 115652 is given below:

```
115653 #define _XOPEN_SOURCE 700 /* Single UNIX Specification, Version x */
115654 #include ...
```

115655 Note that the definition of `_XOPEN_SOURCE` with the value 700 makes the definition of  
 115656 `_POSIX_C_SOURCE` redundant and it can safely be omitted.

115657 **B.2.2.2 The Name Space**

115658 The reservation of identifiers is paraphrased from the ISO C standard. The text is included  
 115659 because it needs to be part of POSIX.1-200x, regardless of possible changes in future versions of  
 115660 the ISO C standard.

115661 These identifiers may be used by implementations, particularly for feature test macros.  
 115662 Implementations should not use feature test macro names that might be reasonably used by a  
 115663 standard.

115664 Including headers more than once is a reasonably common practice, and it should be carried  
 115665 forward from the ISO C standard. More significantly, having definitions in more than one  
 115666 header is explicitly permitted. Where the potential declaration is “benign” (the same definition  
 115667 twice) the declaration can be repeated, if that is permitted by the compiler. (This is usually true  
 115668 of macros, for example.) In those situations where a repetition is not benign (for example,  
 115669 **typedefs**), conditional compilation must be used. The situation actually occurs both within the  
 115670 ISO C standard and within POSIX.1: `time_t` should be in `<sys/types.h>`, and the ISO C standard  
 115671 mandates that it be in `<time.h>`.

115672 The area of name space pollution *versus* additions to structures is difficult because of the macro  
 115673 structure of C. The following discussion summarizes all the various problems with and  
 115674 objections to the issue.

115675 Note the phrase “user-defined macro”. Users are not permitted to define macro names (or any  
 115676 other name) beginning with “\_`[A-Z]`”. Thus, the conflict cannot occur for symbols reserved  
 115677 to the vendor’s name space, and the permission to add fields automatically applies, without  
 115678 qualification, to those symbols.

- 115679 1. Data structures (and unions) need to be defined in headers by implementations to meet  
 115680 certain requirements of POSIX.1 and the ISO C standard.
- 115681 2. The structures defined by POSIX.1 are typically minimal, and any practical  
 115682 implementation would wish to add fields to these structures either to hold additional  
 115683 related information or for backwards-compatibility (or both). Future standards (and *de*  
 115684 *facto* standards) would also wish to add to these structures. Issues of field alignment  
 115685 make it impractical (at least in the general case) to simply omit fields when they are not  
 115686 defined by the particular standard involved.

115687 The `dirent` structure is an example of such a minimal structure (although one could argue  
 115688 about whether the other fields need visible names). The `st_rdev` field of most  
 115689 implementations’ `stat` structure is a common example where extension is needed and

115690 where a conflict could occur.

115691 3. Fields in structures are in an independent name space, so the addition of such fields  
115692 presents no problem to the C language itself in that such names cannot interact with  
115693 identically named user symbols because access is qualified by the specific structure name.

115694 4. There is an exception to this: macro processing is done at a lexical level. Thus, symbols  
115695 added to a structure might be recognized as user-provided macro names at the location  
115696 where the structure is declared. This only can occur if the user-provided name is declared  
115697 as a macro before the header declaring the structure is included. The user's use of the  
115698 name after the declaration cannot interfere with the structure because the symbol is  
115699 hidden and only accessible through access to the structure. Presumably, the user would  
115700 not declare such a macro if there was an intention to use that field name.

115701 5. Macros from the same or a related header might use the additional fields in the structure,  
115702 and those field names might also collide with user macros. Although this is a less  
115703 frequent occurrence, since macros are expanded at the point of use, no constraint on the  
115704 order of use of names can apply.

115705 6. An "obvious" solution of using names in the reserved name space and then redefining  
115706 them as macros when they should be visible does not work because this has the effect of  
115707 exporting the symbol into the general name space. For example, given a (hypothetical)  
115708 system-provided header `<h.h>`, and two parts of a C program in `a.c` and `b.c`, in header  
115709 `<h.h>`:

```
115710 struct foo {
115711 int __i;
115712 }
115713 #ifdef _FEATURE_TEST
115714 #define i __i;
115715 #endif
```

115716 In file `a.c`:

```
115717 #include h.h
115718 extern int i;
115719 ...
```

115720 In file `b.c`:

```
115721 extern int i;
115722 ...
```

115723 The symbol that the user thinks of as `i` in both files has an external name of `__i` in `a.c`; the  
115724 same symbol `i` in `b.c` has an external name `i` (ignoring any hidden manipulations the  
115725 compiler might perform on the names). This would cause a mysterious name resolution  
115726 problem when `a.o` and `b.o` are linked.

115727 Simply avoiding definition then causes alignment problems in the structure.

115728 A structure of the form:

```
115729 struct foo {
115730 union {
115731 int __i;
115732 #ifdef _FEATURE_TEST
115733 int i;
115734 #endif
115735 } __ii;
```



115736

}

115737

does not work because the name of the logical field *i* is `__i.i`, and introduction of a macro to restore the logical name immediately reintroduces the problem discussed previously (although its manifestation might be more immediate because a syntax error would result if a recursive macro did not cause it to fail first).

115738

115739

115740

115741

7. A more workable solution would be to declare the structure:

115742

```
struct foo {
```

115743

```
#ifdef _FEATURE_TEST
```

115744

```
 int i;
```

115745

```
#else
```

115746

```
 int __i;
```

115747

```
#endif
```

115748

}

115749

However, if a macro (particularly one required by a standard) is to be defined that uses this field, two must be defined: one that uses *i*, the other that uses `__i`. If more than one additional field is used in a macro and they are conditional on distinct combinations of features, the complexity goes up as  $2^n$ .

115750

115751

115752

115753

All this leaves a difficult situation: vendors must provide very complex headers to deal with what is conceptually simple and safe—adding a field to a structure. It is the possibility of user-provided macros with the same name that makes this difficult.

115754

115755

115756

Several alternatives were proposed that involved constraining the user's access to part of the name space available to the user (as specified by the ISO C standard). In some cases, this was only until all the headers had been included. There were two proposals discussed that failed to achieve consensus:

115757

115758

115759

115760

1. Limiting it for the whole program.

115761

2. Restricting the use of identifiers containing only uppercase letters until after all system headers had been included. It was also pointed out that because macros might wish to access fields of a structure (and macro expansion occurs totally at point of use) restricting names in this way would not protect the macro expansion, and thus the solution was inadequate.

115762

115763

115764

115765

115766

It was finally decided that reservation of symbols would occur, but as constrained.

115767

The current wording also allows the addition of fields to a structure, but requires that user macros of the same name not interfere. This allows vendors to do one of the following:

115768

115769

- Not create the situation (do not extend the structures with user-accessible names or use the solution in (7) above)

115770

115771

- Extend their compilers to allow some way of adding names to structures and macros safely

115772

There are at least two ways that the compiler might be extended: add new preprocessor directives that turn off and on macro expansion for certain symbols (without changing the value of the macro) and a function or lexical operation that suppresses expansion of a word. The latter seems more flexible, particularly because it addresses the problem in macros as well as in declarations.

115773

115774

115775

115776

115777

The following seems to be a possible implementation extension to the C language that will do this: any token that during macro expansion is found to be preceded by three '#' symbols shall not be further expanded in exactly the same way as described for macros that expand to their own name as in Section 3.8.3.4 of the ISO C standard. A vendor may also wish to implement this as an operation that is lexically a function, which might be implemented as:

115778

115779

115780

115781

115782 `#define __safe_name(x) ###x`

115783 Using a function notation would insulate vendors from changes in standards until such a  
115784 functionality is standardized (if ever). Standardization of such a function would be valuable  
115785 because it would then permit third parties to take advantage of it portably in software they may  
115786 supply.

115787 The symbols that are “explicitly permitted, but not required by POSIX.1-200x” include those  
115788 classified below. (That is, the symbols classified below might, but are not required to, be present  
115789 when `_POSIX_C_SOURCE` is defined to have the value `200xxL`.)

- 115790 • Symbols in `<limits.h>` and `<unistd.h>` that are defined to indicate support for options or  
115791 limits that are constant at compile-time
- 115792 • Symbols in the name space reserved for the implementation by the ISO C standard
- 115793 • Symbols in a name space reserved for a particular type of extension (for example, type  
115794 names ending with `_t` in `<sys/types.h>`)
- 115795 • Additional members of structures or unions whose names do not reduce the name space  
115796 reserved for applications

115797 Since both implementations and future versions of this standard and other POSIX standards  
115798 may use symbols in the reserved spaces described in these tables, there is a potential for name  
115799 space clashes. To avoid future name space clashes when adding symbols, implementations  
115800 should not use the `posix_`, `POSIX_`, or `_POSIX_` prefixes.

115801 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/2 is applied, deleting the entries `POSIX_`,  
115802 `_POSIX_`, and `posix_` from the column of allowed name space prefixes for use by an  
115803 implementation in the first table. The presence of these prefixes was contradicting later text  
115804 which states that: “The prefixes `posix_`, `POSIX_`, and `_POSIX_` are reserved for use by XCU  
115805 [Chapter 2](#) (on page 2245) and other POSIX standards. Implementations may add symbols to the  
115806 headers shown in the following table, provided the identifiers ... do not use the reserved  
115807 prefixes `posix_`, `POSIX_`, or `_POSIX_`.”

115808 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/3 is applied, correcting the reserved  
115809 macro prefix from: “`PRI[a-z]`, `SCN[a-z]`” to: “`PRI[Xa-z]`, `SCN[Xa-z]`” in the second table. The  
115810 change was needed since the ISO C standard allows implementations to define macros of the  
115811 form `PRI` or `SCN` followed by any lowercase letter or ‘`x`’ in `<inttypes.h>`. (The  
115812 ISO/IEC 9899:1999 standard, Subclause 7.26.4.)

115813 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/4 is applied, adding a new section listing  
115814 reserved names for the `<stdint.h>` header. This change is for alignment with the ISO C standard.

115815 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/2 is applied, making it clear that  
115816 implementations are permitted to have symbols with the prefix `_POSIX_` visible in any header.

115817 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/3 is applied, updating the table of  
115818 allowed macro prefixes to include the prefix `FP_[A-Z]` for `<math.h>`. This text is added for  
115819 consistency with the `<math.h>` reference page in the Base Definitions volume of POSIX.1-200x  
115820 which permits additional implementation-defined floating-point classifications.

115821 Austin Group Interpretation 1003.1-2001 #048 is applied, reserving `SEEK_` in the name space.

### 115822 B.2.3 Error Numbers

115823 It was the consensus of the standard developers that to allow the conformance document to state  
115824 that an error occurs and under what conditions, but to disallow a statement that it never occurs,  
115825 does not make sense. It could be implied by the current wording that this is allowed, but to  
115826 reduce the possibility of future interpretation requests, it is better to make an explicit statement.

115827 The ISO C standard requires that *errno* be an assignable lvalue. Originally, the definition in  
115828 POSIX.1 was stricter than that in the ISO C standard, **extern int *errno***, in order to support  
115829 historical usage. In a multi-threaded environment, implementing *errno* as a global variable  
115830 results in non-deterministic results when accessed. It is required, however, that *errno* work as a  
115831 per-thread error reporting mechanism. In order to do this, a separate *errno* value has to be  
115832 maintained for each thread. The following section discusses the various alternative solutions  
115833 that were considered.

115834 In order to avoid this problem altogether for new functions, these functions avoid using *errno*  
115835 and, instead, return the error number directly as the function return value; a return value of zero  
115836 indicates that no error was detected.

115837 For any function that can return errors, the function return value is not used for any purpose  
115838 other than for reporting errors. Even when the output of the function is scalar, it is passed  
115839 through a function argument. While it might have been possible to allow some scalar outputs to  
115840 be coded as negative function return values and mixed in with positive error status returns, this  
115841 was rejected—using the return value for a mixed purpose was judged to be of limited use and  
115842 error prone.

115843 Checking the value of *errno* alone is not sufficient to determine the existence or type of an error,  
115844 since it is not required that a successful function call clear *errno*. The variable *errno* should only  
115845 be examined when the return value of a function indicates that the value of *errno* is meaningful.  
115846 In that case, the function is required to set the variable to something other than zero.

115847 The variable *errno* is never set to zero by any function call; to do so would contradict the ISO C  
115848 standard.

115849 POSIX.1 requires (in the ERRORS sections of function descriptions) certain error values to be set  
115850 in certain conditions because many existing applications depend on them. Some error numbers,  
115851 such as [EFAULT], are entirely implementation-defined and are noted as such in their  
115852 description in the ERRORS section. This section otherwise allows wide latitude to the  
115853 implementation in handling error reporting.

115854 Some of the ERRORS sections in POSIX.1-200x have two subsections. The first:

115855 “The function shall fail if:”

115856 could be called the “mandatory” section.

115857 The second:

115858 “The function may fail if:”

115859 could be informally known as the “optional” section.

115860 Attempting to infer the quality of an implementation based on whether it detects optional error  
115861 conditions is not useful.

115862 Following each one-word symbolic name for an error, there is a description of the error. The  
115863 rationale for some of the symbolic names follows:

|        |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 115864 | [ECANCELED]    | This spelling was chosen as being more common.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 115865 | [EFAULT]       | Most historical implementations do not catch an error and set <i>errno</i> when an invalid address is given to the functions <i>wait()</i> , <i>time()</i> , or <i>times()</i> . Some implementations cannot reliably detect an invalid address. And most systems that detect invalid addresses will do so only for a system call, not for a library routine.                                                                                                                                                                                                                                                                                                                                                                                         |
| 115866 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115867 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115868 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115869 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115870 | [EFTYPE]       | This error code was proposed in earlier proposals as “Inappropriate operation for file type”, meaning that the operation requested is not appropriate for the file specified in the function call. This code was proposed, although the same idea was covered by [ENOTTY], because the connotations of the name would be misleading. It was pointed out that the <i>fcntl()</i> function uses the error code [EINVAL] for this notion, and hence all instances of [EFTYPE] were changed to this code.                                                                                                                                                                                                                                                 |
| 115871 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115872 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115873 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115874 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115875 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115876 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115877 | [EINTR]        | POSIX.1 prohibits conforming implementations from restarting interrupted system calls of conforming applications unless the SA_RESTART flag is in effect for the signal. However, it does not require that [EINTR] be returned when another legitimate value may be substituted; for example, a partial transfer count when <i>read()</i> or <i>write()</i> are interrupted. This is only given when the signal-catching function returns normally as opposed to returns by mechanisms like <i>longjmp()</i> or <i>siglongjmp()</i> .                                                                                                                                                                                                                 |
| 115878 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115879 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115880 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115881 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115882 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115883 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115884 | [ELOOP]        | In specifying conditions under which implementations would generate this error, the following goals were considered: <ul style="list-style-type: none"> <li>• To ensure that actual loops are detected, including loops that result from symbolic links across distributed file systems.</li> <li>• To ensure that during pathname resolution an application can rely on the ability to follow at least {SYMLOOP_MAX} symbolic links in the absence of a loop.</li> <li>• To allow implementations to provide the capability of traversing more than {SYMLOOP_MAX} symbolic links in the absence of a loop.</li> <li>• To allow implementations to detect loops and generate the error prior to encountering {SYMLOOP_MAX} symbolic links.</li> </ul> |
| 115885 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115886 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115887 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115888 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115889 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115890 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115891 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115892 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115893 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115894 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115895 | [ENAMETOOLONG] | When a symbolic link is encountered during pathname resolution, the contents of that symbolic link are used to create a new pathname. The standard developers intended to allow, but not require, that implementations enforce the restriction of {PATH_MAX} on the result of this pathname substitution.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 115896 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115897 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115898 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115899 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115900 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115901 | [ENOMEM]       | The term “main memory” is not used in POSIX.1 because it is implementation-defined.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 115902 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115903 | [ENOTSUP]      | This error code is to be used when an implementation chooses to implement the required functionality of POSIX.1-200x but does not support optional facilities defined by POSIX.1-200x. The return of [ENOSYS] is to be taken to indicate that the function of the interface is not supported at all; the function will always fail with this error code.                                                                                                                                                                                                                                                                                                                                                                                              |
| 115904 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115905 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115906 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115907 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115908 | [ENOTTY]       | The symbolic name for this error is derived from a time when device control was done by <i>ioctl()</i> and that operation was only permitted on a terminal interface. The term “TTY” is derived from “teletypewriter”, the devices to                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 115909 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 115910 |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

- 115911 which this error originally applied.
- 115912 [EOVERFLOW] Most of the uses of this error code are related to large file support. Typically,  
115913 these cases occur on systems which support multiple programming  
115914 environments with different sizes for `off_t`, but they may also occur in  
115915 connection with remote file systems.
- 115916 In addition, when different programming environments have different widths  
115917 for types such as `int` and `uid_t`, several functions may encounter a condition  
115918 where a value in a particular environment is too wide to be represented. In  
115919 that case, this error should be raised. For example, suppose the currently  
115920 running process has 64-bit `int`, and file descriptor 9 223 372 036 854 775 807 is  
115921 open and does not have the `close-on-exec` flag set. If the process then uses  
115922 `execl()` to `exec` a file compiled in a programming environment with 32-bit `int`,  
115923 the call to `execl()` can fail with `errno` set to [EOVERFLOW]. A similar failure  
115924 can occur with `execl()` if any of the user IDs or any of the group IDs to be  
115925 assigned to the new process image are out of range for the executed file's  
115926 programming environment.
- 115927 Note, however, that this condition cannot occur for functions that are  
115928 explicitly described as always being successful, such as `getpid()`.
- 115929 [EPIPE] This condition normally generates the signal SIGPIPE; the error is returned if  
115930 the signal does not terminate the process.
- 115931 [EROFS] In historical implementations, attempting to `unlink()` or `rmdir()` a mount point  
115932 would generate an [EBUSY] error. An implementation could be envisioned  
115933 where such an operation could be performed without error. In this case, if  
115934 either the directory entry or the actual data structures reside on a read-only file  
115935 system, [EROFS] is the appropriate error to generate. (For example, changing  
115936 the link count of a file on a read-only file system could not be done, as is  
115937 required by `unlink()`, and thus an error should be reported.)
- 115938 Three error numbers, [EDOM], [EILSEQ], and [ERANGE], were added to this section primarily  
115939 for consistency with the ISO C standard.

#### 115940 Alternative Solutions for Per-Thread `errno`

115941 The usual implementation of `errno` as a single global variable does not work in a multi-threaded  
115942 environment. In such an environment, a thread may make a POSIX.1 call and get a `-1` error  
115943 return, but before that thread can check the value of `errno`, another thread might have made a  
115944 second POSIX.1 call that also set `errno`. This behavior is unacceptable in robust programs. There  
115945 were a number of alternatives that were considered for handling the `errno` problem:

- 115946 • Implement `errno` as a per-thread integer variable.
- 115947 • Implement `errno` as a service that can access the per-thread error number.
- 115948 • Change all POSIX.1 calls to accept an extra status argument and avoid setting `errno`.
- 115949 • Change all POSIX.1 calls to raise a language exception.

115950 The first option offers the highest level of compatibility with existing practice but requires  
115951 special support in the linker, compiler, and/or virtual memory system to support the new  
115952 concept of thread private variables. When compared with current practice, the third and fourth  
115953 options are much cleaner, more efficient, and encourage a more robust programming style, but  
115954 they require new versions of all of the POSIX.1 functions that might detect an error. The second  
115955 option offers compatibility with existing code that uses the `<errno.h>` header to define the  
115956 symbol `errno`. In this option, `errno` may be a macro defined:

```

115957 #define errno (*__errno())
115958 extern int *__errno();

```

115959 This option may be implemented as a per-thread variable whereby an *errno* field is allocated in  
 115960 the user space object representing a thread, and whereby the function `__errno()` makes a system  
 115961 call to determine the location of its user space object and returns the address of the *errno* field of  
 115962 that object. Another implementation, one that avoids calling the kernel, involves allocating  
 115963 stacks in chunks. The stack allocator keeps a side table indexed by chunk number containing a  
 115964 pointer to the thread object that uses that chunk. The `__errno()` function then looks at the stack  
 115965 pointer, determines the chunk number, and uses that as an index into the chunk table to find its  
 115966 thread object and thus its private value of *errno*. On most architectures, this can be done in four  
 115967 to five instructions. Some compilers may wish to implement `__errno()` inline to improve  
 115968 performance.

### 115969 Disallowing Return of the [EINTR] Error Code

115970 Many blocking interfaces defined by POSIX.1-200x may return [EINTR] if interrupted during  
 115971 their execution by a signal handler. Blocking interfaces introduced under the threads  
 115972 functionality do not have this property. Instead, they require that the interface appear to be  
 115973 atomic with respect to interruption. In particular, clients of blocking interfaces need not handle  
 115974 any possible [EINTR] return as a special case since it will never occur. If it is necessary to restart  
 115975 operations or complete incomplete operations following the execution of a signal handler, this is  
 115976 handled by the implementation, rather than by the application.

115977 Requiring applications to handle [EINTR] errors on blocking interfaces has been shown to be a  
 115978 frequent source of often unreproducible bugs, and it adds no compelling value to the available  
 115979 functionality. Thus, blocking interfaces introduced for use by multi-threaded programs do not  
 115980 use this paradigm. In particular, in none of the functions `flockfile()`, `pthread_cond_timedwait()`,  
 115981 `pthread_cond_wait()`, `pthread_join()`, `pthread_mutex_lock()`, and `sigwait()` did providing [EINTR]  
 115982 returns add value, or even particularly make sense. Thus, these functions do not provide for an  
 115983 [EINTR] return, even when interrupted by a signal handler. The same arguments can be applied  
 115984 to `sem_wait()`, `sem_trywait()`, `sigwaitinfo()`, and `sigtimedwait()`, but implementations are  
 115985 permitted to return [EINTR] error codes for these functions for compatibility with earlier  
 115986 versions of this standard. Applications cannot rely on calls to these functions returning [EINTR]  
 115987 error codes when signals are delivered to the calling thread, but they should allow for the  
 115988 possibility.

115989 Austin Group Interpretation 1003.1-2001 #050 is applied, allowing [ENOTSUP] and  
 115990 [EOPNOTSUPP] to be the same values.

#### 115991 B.2.3.1 Additional Error Numbers

115992 The ISO C standard defines the name space for implementations to add additional error  
 115993 numbers.

### 115994 B.2.4 Signal Concepts

115995 Historical implementations of signals, using the `signal()` function, have shortcomings that make  
 115996 them unreliable for many application uses. Because of this, a new signal mechanism, based very  
 115997 closely on the one of 4.2 BSD and 4.3 BSD, was added to POSIX.1.

115998 **Signal Names**

115999 The restriction on the actual type used for **sigset\_t** is intended to guarantee that these objects can  
 116000 always be assigned, have their address taken, and be passed as parameters by value. It is not  
 116001 intended that this type be a structure including pointers to other data structures, as that could  
 116002 impact the portability of applications performing such operations. A reasonable implementation  
 116003 could be a structure containing an array of some integer type.

116004 The signals described in POSIX.1-200x must have unique values so that they may be named as  
 116005 parameters of **case** statements in the body of a C-language **switch** clause. However,  
 116006 implementation-defined signals may have values that overlap with each other or with signals  
 116007 specified in POSIX.1-200x. An example of this is SIGABRT, which traditionally overlaps some  
 116008 other signal, such as SIGIOT.

116009 SIGKILL, SIGTERM, SIGUSR1, and SIGUSR2 are ordinarily generated only through the explicit  
 116010 use of the *kill()* function, although some implementations generate SIGKILL under  
 116011 extraordinary circumstances. SIGTERM is traditionally the default signal sent by the *kill*  
 116012 command.

116013 The signals SIGBUS, SIGEMT, SIGIOT, SIGTRAP, and SIGSYS were omitted from POSIX.1  
 116014 because their behavior is implementation-defined and could not be adequately categorized.  
 116015 Conforming implementations may deliver these signals, but must document the circumstances  
 116016 under which they are delivered and note any restrictions concerning their delivery. The signals  
 116017 SIGFPE, SIGILL, and SIGSEGV are similar in that they also generally result only from  
 116018 programming errors. They were included in POSIX.1 because they do indicate three relatively  
 116019 well-categorized conditions. They are all defined by the ISO C standard and thus would have to  
 116020 be defined by any system with an ISO C standard binding, even if not explicitly included in  
 116021 POSIX.1.

116022 There is very little that a Conforming POSIX.1 Application can do by catching, ignoring, or  
 116023 masking any of the signals SIGILL, SIGTRAP, SIGIOT, SIGEMT, SIGBUS, SIGSEGV, SIGSYS, or  
 116024 SIGFPE. They will generally be generated by the system only in cases of programming errors.  
 116025 While it may be desirable for some robust code (for example, a library routine) to be able to  
 116026 detect and recover from programming errors in other code, these signals are not nearly sufficient  
 116027 for that purpose. One portable use that does exist for these signals is that a command interpreter  
 116028 can recognize them as the cause of termination of a process (with *wait()*) and print an  
 116029 appropriate message. The mnemonic tags for these signals are derived from their PDP-11 origin.

116030 The signals SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, and SIGCONT are provided for job control  
 116031 and are unchanged from 4.2 BSD. The signal SIGCHLD is also typically used by job control  
 116032 shells to detect children that have terminated or, as in 4.2 BSD, stopped.

116033 Some implementations, including System V, have a signal named SIGCLD, which is similar to  
 116034 SIGCHLD in 4.2 BSD. POSIX.1 permits implementations to have a single signal with both  
 116035 names. POSIX.1 carefully specifies ways in which conforming applications can avoid the  
 116036 semantic differences between the two different implementations. The name SIGCHLD was  
 116037 chosen for POSIX.1 because most current application usages of it can remain unchanged in  
 116038 conforming applications. SIGCLD in System V has more cases of semantics that POSIX.1 does  
 116039 not specify, and thus applications using it are more likely to require changes in addition to the  
 116040 name change.

116041 The signals SIGUSR1 and SIGUSR2 are commonly used by applications for notification of  
 116042 exceptional behavior and are described as “reserved as application-defined” so that such use is  
 116043 not prohibited. Implementations should not generate SIGUSR1 or SIGUSR2, except when  
 116044 explicitly requested by *kill()*. It is recommended that libraries not use these two signals, as such  
 116045 use in libraries could interfere with their use by applications calling the libraries. If such use is  
 116046 unavoidable, it should be documented. It is prudent for non-portable libraries to use non-

116047 standard signals to avoid conflicts with use of standard signals by portable libraries.

116048 There is no portable way for an application to catch or ignore non-standard signals. Some  
 116049 implementations define the range of signal numbers, so applications can install signal-catching  
 116050 functions for all of them. Unfortunately, implementation-defined signals often cause problems  
 116051 when caught or ignored by applications that do not understand the reason for the signal. While  
 116052 the desire exists for an application to be more robust by handling all possible signals (even those  
 116053 only generated by *kill()*), no existing mechanism was found to be sufficiently portable to include  
 116054 in POSIX.1. The value of such a mechanism, if included, would be diminished given that  
 116055 SIGKILL would still not be catchable.

116056 A number of new signal numbers are reserved for applications because the two user signals  
 116057 defined by POSIX.1 are insufficient for many realtime applications. A range of signal numbers is  
 116058 specified, rather than an enumeration of additional reserved signal names, because different  
 116059 applications and application profiles will require a different number of application signals. It is  
 116060 not desirable to burden all application domains and therefore all implementations with the  
 116061 maximum number of signals required by all possible applications. Note that in this context,  
 116062 signal numbers are essentially different signal priorities.

116063 The relatively small number of required additional signals, `{_POSIX_RTSIG_MAX}`, was chosen  
 116064 so as not to require an unreasonably large signal mask/set. While this number of signals  
 116065 defined in POSIX.1 will fit in a single 32-bit word signal mask, it is recognized that most existing  
 116066 implementations define many more signals than are specified in POSIX.1 and, in fact, many  
 116067 implementations have already exceeded 32 signals (including the “null signal”). Support of  
 116068 `{_POSIX_RTSIG_MAX}` additional signals may push some implementation over the single 32-bit  
 116069 word line, but is unlikely to push any implementations that are already over that line beyond  
 116070 the 64-signal line.

#### 116071 B.2.4.1 *Signal Generation and Delivery*

116072 The terms defined in this section are not used consistently in documentation of historical  
 116073 systems. Each signal can be considered to have a lifetime beginning with generation and ending  
 116074 with delivery or acceptance. The POSIX.1 definition of “delivery” does not exclude ignored  
 116075 signals; this is considered a more consistent definition. This revised text in several parts of  
 116076 POSIX.1-200x clarifies the distinct semantics of asynchronous signal delivery and synchronous  
 116077 signal acceptance. The previous wording attempted to categorize both under the term  
 116078 “delivery”, which led to conflicts over whether the effects of asynchronous signal delivery  
 116079 applied to synchronous signal acceptance.

116080 Signals generated for a process are delivered to only one thread. Thus, if more than one thread  
 116081 is eligible to receive a signal, one has to be chosen. The choice of threads is left entirely up to the  
 116082 implementation both to allow the widest possible range of conforming implementations and to  
 116083 give implementations the freedom to deliver the signal to the “easiest possible” thread should  
 116084 there be differences in ease of delivery between different threads.

116085 Note that should multiple delivery among cooperating threads be required by an application,  
 116086 this can be trivially constructed out of the provided single-delivery semantics. The construction  
 116087 of a *sigwait\_multiple()* function that accomplishes this goal is presented with the rationale for  
 116088 *sigwaitinfo()*.

116089 Implementations should deliver unblocked signals as soon after they are generated as possible.  
 116090 However, it is difficult for POSIX.1 to make specific requirements about this, beyond those in  
 116091 *kill()* and *sigprocmask()*. Even on systems with prompt delivery, scheduling of higher priority  
 116092 processes is always likely to cause delays.

116093 In general, the interval between the generation and delivery of unblocked signals cannot be  
 116094 detected by an application. Thus, references to pending signals generally apply to blocked,



116095 pending signals. An implementation registers a signal as pending on the process when no  
116096 thread has the signal unblocked and there are no threads blocked in a *sigwait()* function for that  
116097 signal. Thereafter, the implementation delivers the signal to the first thread that unblocks the  
116098 signal or calls a *sigwait()* function on a signal set containing this signal rather than choosing the  
116099 recipient thread at the time the signal is sent.

116100 In the 4.3 BSD system, signals that are blocked and set to SIG\_IGN are discarded immediately  
116101 upon generation. For a signal that is ignored as its default action, if the action is SIG\_DFL and  
116102 the signal is blocked, a generated signal remains pending. In the 4.1 BSD system and in  
116103 System V Release 3 (two other implementations that support a somewhat similar signal  
116104 mechanism), all ignored blocked signals remain pending if generated. Because it is not normally  
116105 useful for an application to simultaneously ignore and block the same signal, it was unnecessary  
116106 for POSIX.1 to specify behavior that would invalidate any of the historical implementations.

116107 There is one case in some historical implementations where an unblocked, pending signal does  
116108 not remain pending until it is delivered. In the System V implementation of *signal()*, pending  
116109 signals are discarded when the action is set to SIG\_DFL or a signal-catching routine (as well as  
116110 to SIG\_IGN). Except in the case of setting SIGCHLD to SIG\_DFL, implementations that do this  
116111 do not conform completely to POSIX.1. Some earlier proposals for POSIX.1 explicitly stated this,  
116112 but these statements were redundant due to the requirement that functions defined by POSIX.1  
116113 not change attributes of processes defined by POSIX.1 except as explicitly stated.

116114 POSIX.1 specifically states that the order in which multiple, simultaneously pending signals are  
116115 delivered is unspecified. This order has not been explicitly specified in historical  
116116 implementations, but has remained quite consistent and been known to those familiar with the  
116117 implementations. Thus, there have been cases where applications (usually system utilities) have  
116118 been written with explicit or implicit dependencies on this order. Implementors and others  
116119 porting existing applications may need to be aware of such dependencies.

116120 When there are multiple pending signals that are not blocked, implementations should arrange  
116121 for the delivery of all signals at once, if possible. Some implementations stack calls to all pending  
116122 signal-catching routines, making it appear that each signal-catcher was interrupted by the next  
116123 signal. In this case, the implementation should ensure that this stacking of signals does not  
116124 violate the semantics of the signal masks established by *sigaction()*. Other implementations  
116125 process at most one signal when the operating system is entered, with remaining signals saved  
116126 for later delivery. Although this practice is widespread, this behavior is neither standardized  
116127 nor endorsed. In either case, implementations should attempt to deliver signals associated with  
116128 the current state of the process (for example, SIGFPE) before other signals, if possible.

116129 In 4.2 BSD and 4.3 BSD, it is not permissible to ignore or explicitly block SIGCONT, because if  
116130 blocking or ignoring this signal prevented it from continuing a stopped process, such a process  
116131 could never be continued (only killed by SIGKILL). However, 4.2 BSD and 4.3 BSD do block  
116132 SIGCONT during execution of its signal-catching function when it is caught, creating exactly  
116133 this problem. A proposal was considered to disallow catching SIGCONT in addition to ignoring  
116134 and blocking it, but this limitation led to objections. The consensus was to require that  
116135 SIGCONT always continue a stopped process when generated. This removed the need to  
116136 disallow ignoring or explicit blocking of the signal; note that SIG\_IGN and SIG\_DFL are  
116137 equivalent for SIGCONT.

## 116138 B.2.4.2 Realtime Signal Generation and Delivery

116139 The realtime signals functionality is required in this version of the standard for the following  
 116140 reasons:

- 116141 • The **sigevent** structure is used by other POSIX.1 functions that result in asynchronous  
 116142 event notifications to specify the notification mechanism to use and other information  
 116143 needed by the notification mechanism. POSIX.1-200x defines only three symbolic values  
 116144 for the notification mechanism:
  - 116145 — SIGEV\_NONE is used to indicate that no notification is required when the event  
 116146 occurs. This is useful for applications that use asynchronous I/O with polling for  
 116147 completion.
  - 116148 — SIGEV\_SIGNAL indicates that a signal is generated when the event occurs.
  - 116149 — SIGEV\_THREAD provides for “callback functions” for asynchronous notifications  
 116150 done by a function call within the context of a new thread. This provides a multi-  
 116151 threaded process with a more natural means of notification than signals.

116152 The primary difficulty with previous notification approaches has been to specify the  
 116153 environment of the notification routine.

- 116154 — One approach is to limit the notification routine to call only functions permitted in a  
 116155 signal handler. While the list of permissible functions is clearly stated, this is overly  
 116156 restrictive.
- 116157 — A second approach is to define a new list of functions or classes of functions that are  
 116158 explicitly permitted or not permitted. This would give a programmer more lists to  
 116159 deal with, which would be awkward.
- 116160 — The third approach is to define completely the environment for execution of the  
 116161 notification function. A clear definition of an execution environment for notification  
 116162 is provided by executing the notification function in the environment of a newly  
 116163 created thread.

116164 Implementations may support additional notification mechanisms by defining new values  
 116165 for *sigev\_notify*.

116166 For a notification type of SIGEV\_SIGNAL, the other members of the **sigevent** structure  
 116167 defined by POSIX.1-200x specify the realtime signal—that is, the signal number and  
 116168 application-defined value that differentiates between occurrences of signals with the same  
 116169 number—that will be generated when the event occurs. The structure is defined in  
 116170 **<signal.h>**, even though the structure is not directly used by any of the signal functions,  
 116171 because it is part of the signals interface used by the POSIX.1b “client functions”. When the  
 116172 client functions include **<signal.h>** to define the signal names, the **sigevent** structure will  
 116173 also be defined.

116174 An application-defined value passed to the signal handler is used to differentiate between  
 116175 different “events” instead of requiring that the application use different signal numbers for  
 116176 several reasons:

- 116177 — Realtime applications potentially handle a very large number of different events.  
 116178 Requiring that implementations support a correspondingly large number of distinct  
 116179 signal numbers will adversely impact the performance of signal delivery because the  
 116180 signal masks to be manipulated on entry and exit to the handlers will become large.
- 116181 — Event notifications are prioritized by signal number (the rationale for this is  
 116182 explained in the following paragraphs) and the use of different signal numbers to  
 116183 differentiate between the different event notifications overloads the signal number

116184 more than has already been done. It also requires that the application developer  
116185 make arbitrary assignments of priority to events that are logically of equal priority.

116186 A union is defined for the application-defined value so that either an integer constant or a  
116187 pointer can be portably passed to the signal-catching function. On some architectures a  
116188 pointer cannot be cast to an **int** and *vice versa*.

116189 Use of a structure here with an explicit notification type discriminant rather than explicit  
116190 parameters to realtime functions, or embedded in other realtime structures, provides for  
116191 future extensions to POSIX.1-200x. Additional, perhaps more efficient, notification  
116192 mechanisms can be supported for existing realtime function interfaces, such as timers and  
116193 asynchronous I/O, by extending the **sigevent** structure appropriately. The existing  
116194 realtime function interfaces will not have to be modified to use any such new notification  
116195 mechanism. The revised text concerning the SIGEV\_SIGNAL value makes consistent the  
116196 semantics of the members of the **sigevent** structure, particularly in the definitions of  
116197 *lio\_listio()* and *aio\_fsync()*. For uniformity, other revisions cause this specification to be  
116198 referred to rather than inaccurately duplicated in the descriptions of functions and  
116199 structures using the **sigevent** structure. The revised wording does not relax the  
116200 requirement that the signal number be in the range SIGRTMIN to SIGRTMAX to guarantee  
116201 queuing and passing of the application value, since that requirement is still implied by the  
116202 signal names.

116203 • POSIX.1-200x is intentionally vague on whether “non-realtime” signal-generating  
116204 mechanisms can result in a **siginfo\_t** being supplied to the handler on delivery. In one  
116205 existing implementation, a **siginfo\_t** is posted on signal generation, even though the  
116206 implementation does not support queuing of multiple occurrences of a signal. It is not the  
116207 intent of POSIX.1-200x to preclude this, independent of the mandate to define signals that  
116208 do support queuing. Any interpretation that appears to preclude this is a mistake in the  
116209 reading or writing of the standard.

116210 • Signals handled by realtime signal handlers might be generated by functions or conditions  
116211 that do not allow the specification of an application-defined value and do not queue.  
116212 POSIX.1-200x specifies the *si\_code* member of the **siginfo\_t** structure used in existing  
116213 practice and defines additional codes so that applications can detect whether an  
116214 application-defined value is present or not. The code SI\_USER for *kill()*-generated signals  
116215 is adopted from existing practice.

116216 • The *sigaction()* *sa\_flags* value SA\_SIGINFO tells the implementation that the signal-  
116217 catching function expects two additional arguments. When the flag is not set, a single  
116218 argument, the signal number, is passed as specified by POSIX.1-200x. Although  
116219 POSIX.1-200x does not explicitly allow the *info* argument to the handler function to be  
116220 NULL, this is existing practice. This provides for compatibility with programs whose  
116221 signal-catching functions are not prepared to accept the additional arguments.  
116222 POSIX.1-200x is explicitly unspecified as to whether signals actually queue when  
116223 SA\_SIGINFO is not set for a signal, as there appear to be no benefits to applications in  
116224 specifying one behavior or another. One existing implementation queues a **siginfo\_t** on  
116225 each signal generation, unless the signal is already pending, in which case the  
116226 implementation discards the new **siginfo\_t**; that is, the queue length is never greater than  
116227 one. This implementation only examines SA\_SIGINFO on signal delivery, discarding the  
116228 queued **siginfo\_t** if its delivery was not requested.

116229 POSIX.1-200x specifies several new values for the *si\_code* member of the **siginfo\_t**  
116230 structure. In existing practice, a *si\_code* value of less than or equal to zero indicates that the  
116231 signal was generated by a process via the *kill()* function. In existing practice, values of  
116232 *si\_code* that provide additional information for implementation-generated signals, such as  
116233 SIGFPE or SIGSEGV, are all positive. Thus, if implementations define the new constants

116234 specified in POSIX.1-200x to be negative numbers, programs written to use existing  
 116235 practice will not break. POSIX.1-200x chose not to attempt to specify existing practice  
 116236 values of *si\_code* other than SI\_USER both because it was deemed beyond the scope of  
 116237 POSIX.1-200x and because many of the values in existing practice appear to be platform  
 116238 and implementation-defined. But, POSIX.1-200x does specify that if an implementation—  
 116239 for example, one that does not have existing practice in this area—chooses to define  
 116240 additional values for *si\_code*, these values have to be different from the values of the  
 116241 symbols specified by POSIX.1-200x. This will allow conforming applications to  
 116242 differentiate between signals generated by one of the POSIX.1b asynchronous events and  
 116243 those generated by other implementation events in a manner compatible with existing  
 116244 practice.

116245 The unique values of *si\_code* for the POSIX.1b asynchronous events have implications for  
 116246 implementations of, for example, asynchronous I/O or message passing in user space  
 116247 library code. Such an implementation will be required to provide a hidden interface to the  
 116248 signal generation mechanism that allows the library to specify the standard values of  
 116249 *si\_code*.

116250 Existing practice also defines additional members of **siginfo\_t**, such as the process ID and  
 116251 user ID of the sending process for *kill()*-generated signals. These members were deemed  
 116252 not necessary to meet the requirements of realtime applications and are not specified by  
 116253 POSIX.1-200x. Neither are they precluded.

116254 The third argument to the signal-catching function, *context*, is left undefined by  
 116255 POSIX.1-200x, but is specified in the interface because it matches existing practice for the  
 116256 SA\_SIGINFO flag. It was considered undesirable to require a separate implementation for  
 116257 SA\_SIGINFO for POSIX conformance on implementations that already support the two  
 116258 additional parameters.

116259 • The requirement to deliver lower numbered signals in the range SIGRTMIN to SIGRTMAX  
 116260 first, when multiple unblocked signals are pending, results from several considerations:

116261 — A method is required to prioritize event notifications. The signal number was chosen  
 116262 instead of, for instance, associating a separate priority with each request, because an  
 116263 implementation has to check pending signals at various points and select one for  
 116264 delivery when more than one is pending. Specifying a selection order is the minimal  
 116265 additional semantic that will achieve prioritized delivery. If a separate priority were  
 116266 to be associated with queued signals, it would be necessary for an implementation to  
 116267 search all non-empty, non-blocked signal queues and select from among them the  
 116268 pending signal with the highest priority. This would significantly increase the cost of  
 116269 and decrease the determinism of signal delivery.

116270 — Given the specified selection of the lowest numeric unblocked pending signal,  
 116271 preemptive priority signal delivery can be achieved using signal numbers and signal  
 116272 masks by ensuring that the *sa\_mask* for each signal number blocks all signals with a  
 116273 higher numeric value.

116274 For realtime applications that want to use only the newly defined realtime signal  
 116275 numbers without interference from the standard signals, this can be achieved by  
 116276 blocking all of the standard signals in the thread signal mask and in the *sa\_mask*  
 116277 installed by the signal action for the realtime signal handlers.

116278 POSIX.1-200x explicitly leaves unspecified the ordering of signals outside of the range of  
 116279 realtime signals and the ordering of signals within this range with respect to those outside  
 116280 the range. It was believed that this would unduly constrain implementations or standards  
 116281 in the future definition of new signals.

## 116282 B.2.4.3 Signal Actions

116283 Early proposals mentioned SIGCONT as a second exception to the rule that signals are not  
 116284 delivered to stopped processes until continued. Because POSIX.1-200x now specifies that  
 116285 SIGCONT causes the stopped process to continue when it is generated, delivery of SIGCONT is  
 116286 not prevented because a process is stopped, even without an explicit exception to this rule.

116287 Ignoring a signal by setting the action to SIG\_IGN (or SIG\_DFL for signals whose default action  
 116288 is to ignore) is not the same as installing a signal-catching function that simply returns. Invoking  
 116289 such a function will interrupt certain system functions that block processes (for example, *wait()*,  
 116290 *sigsuspend()*, *pause()*, *read()*, *write()*) while ignoring a signal has no such effect on the process.

116291 Historical implementations discard pending signals when the action is set to SIG\_IGN.  
 116292 However, they do not always do the same when the action is set to SIG\_DFL and the default  
 116293 action is to ignore the signal. POSIX.1-200x requires this for the sake of consistency and also for  
 116294 completeness, since the only signal this applies to is SIGCHLD, and POSIX.1-200x disallows  
 116295 setting its action to SIG\_IGN.

116296 Some implementations (System V, for example) assign different semantics for SIGCLD  
 116297 depending on whether the action is set to SIG\_IGN or SIG\_DFL. Since POSIX.1 requires that the  
 116298 default action for SIGCHLD be to ignore the signal, applications should always set the action to  
 116299 SIG\_DFL in order to avoid SIGCHLD.

116300 Whether or not an implementation allows SIG\_IGN as a SIGCHLD disposition to be inherited  
 116301 across a call to one of the *exec* family of functions or *posix\_spawn()* is explicitly left as  
 116302 unspecified. This change was made as a result of IEEE PASC Interpretation 1003.1 #132, and  
 116303 permits the implementation to decide between the following alternatives:

- 116304 • Unconditionally leave SIGCHLD set to SIG\_IGN, in which case the implementation would  
 116305 not allow applications that assume inheritance of SIG\_DFL to conform to POSIX.1-200x  
 116306 without change. The implementation would, however, retain an ability to control  
 116307 applications that create child processes but never call on the *wait* family of functions,  
 116308 potentially filling up the process table.
- 116309 • Unconditionally reset SIGCHLD to SIG\_DFL, in which case the implementation would  
 116310 allow applications that assume inheritance of SIG\_DFL to conform. The implementation  
 116311 would, however, lose an ability to control applications that spawn child processes but  
 116312 never reap them.
- 116313 • Provide some mechanism, not specified in POSIX.1-200x, to control inherited SIGCHLD  
 116314 dispositions.

116315 Some implementations (System V, for example) will deliver a SIGCLD signal immediately when  
 116316 a process establishes a signal-catching function for SIGCLD when that process has a child that  
 116317 has already terminated. Other implementations, such as 4.3 BSD, do not generate a new  
 116318 SIGCHLD signal in this way. In general, a process should not attempt to alter the signal action  
 116319 for the SIGCHLD signal while it has any outstanding children. However, it is not always  
 116320 possible for a process to avoid this; for example, shells sometimes start up processes in pipelines  
 116321 with other processes from the pipeline as children. Processes that cannot ensure that they have  
 116322 no children when altering the signal action for SIGCHLD thus need to be prepared for, but not  
 116323 depend on, generation of an immediate SIGCHLD signal.

116324 The default action of the stop signals (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU) is to stop a  
 116325 process that is executing. If a stop signal is delivered to a process that is already stopped, it has  
 116326 no effect. In fact, if a stop signal is generated for a stopped process whose signal mask blocks the  
 116327 signal, the signal will never be delivered to the process since the process must receive a  
 116328 SIGCONT, which discards all pending stop signals, in order to continue executing.

116329 The SIGCONT signal continues a stopped process even if SIGCONT is blocked (or ignored).  
 116330 However, if a signal-catching routine has been established for SIGCONT, it will not be entered  
 116331 until SIGCONT is unblocked.

116332 If a process in an orphaned process group stops, it is no longer under the control of a job control  
 116333 shell and hence would not normally ever be continued. Because of this, orphaned processes that  
 116334 receive terminal-related stop signals (SIGTSTP, SIGTTIN, SIGTTOU, but not SIGSTOP) must not  
 116335 be allowed to stop. The goal is to prevent stopped processes from languishing forever. (As  
 116336 SIGSTOP is sent only via *kill()*, it is assumed that the process or user sending a SIGSTOP can  
 116337 send a SIGCONT when desired.) Instead, the system must discard the stop signal. As an  
 116338 extension, it may also deliver another signal in its place. 4.3 BSD sends a SIGKILL, which is  
 116339 overly effective because SIGKILL is not catchable. Another possible choice is SIGHUP. 4.3 BSD  
 116340 also does this for orphaned processes (processes whose parent has terminated) rather than for  
 116341 members of orphaned process groups; this is less desirable because job control shells manage  
 116342 process groups. POSIX.1 also prevents SIGTTIN and SIGTTOU signals from being generated for  
 116343 processes in orphaned process groups as a direct result of activity on a terminal, preventing  
 116344 infinite loops when *read()* and *write()* calls generate signals that are discarded; see [Section](#)  
 116345 [A.11.1.4](#) (on page 3381). A similar restriction on the generation of SIGTSTP was considered, but  
 116346 that would be unnecessary and more difficult to implement due to its asynchronous nature.

116347 Although POSIX.1 requires that signal-catching functions be called with only one argument,  
 116348 there is nothing to prevent conforming implementations from extending POSIX.1 to pass  
 116349 additional arguments, as long as Strictly Conforming POSIX.1 Applications continue to compile  
 116350 and execute correctly. Most historical implementations do, in fact, pass additional, signal-  
 116351 specific arguments to certain signal-catching routines.

116352 There was a proposal to change the declared type of the signal handler to:

```
116353 void func (int sig, ...);
```

116354 The usage of ellipses ("...") is ISO C standard syntax to indicate a variable number of  
 116355 arguments. Its use was intended to allow the implementation to pass additional information to  
 116356 the signal handler in a standard manner.

116357 Unfortunately, this construct would require all signal handlers to be defined with this syntax  
 116358 because the ISO C standard allows implementations to use a different parameter passing  
 116359 mechanism for variable parameter lists than for non-variable parameter lists. Thus, all existing  
 116360 signal handlers in all existing applications would have to be changed to use the variable syntax  
 116361 in order to be standard and portable. This is in conflict with the goal of Minimal Changes to  
 116362 Existing Application Code.

116363 When terminating a process from a signal-catching function, processes should be aware of any  
 116364 interpretation that their parent may make of the status returned by *wait()*, *waitid()*, or *waitpid()*.  
 116365 In particular, a signal-catching function should not call *exit(0)* or *\_exit(0)* unless it wants to  
 116366 indicate successful termination. A non-zero argument to *exit()* or *\_exit()* can be used to indicate  
 116367 unsuccessful termination. Alternatively, the process can use *kill()* to send itself a fatal signal  
 116368 (first ensuring that the signal is set to the default action and not blocked). See also the  
 116369 RATIONALE section of the *\_exit()* function.

116370 The behavior of *unsafe* functions, as defined by this section, is undefined when they are invoked  
 116371 from signal-catching functions in certain circumstances. The behavior of reentrant functions, as  
 116372 defined by this section, is as specified by POSIX.1, regardless of invocation from a signal-  
 116373 catching function. This is the only intended meaning of the statement that reentrant functions  
 116374 may be used in signal-catching functions without restriction. Applications must still consider all  
 116375 effects of such functions on such things as data structures, files, and process state. In particular,  
 116376 application developers need to consider the restrictions on interactions when interrupting  
 116377 *sleep()* (see *sleep()*) and interactions among multiple handles for a file description. The fact that

116378 any specific function is listed as reentrant does not necessarily mean that invocation of that  
116379 function from a signal-catching function is recommended.

116380 In order to prevent errors arising from interrupting non-reentrant function calls, applications  
116381 should protect calls to these functions either by blocking the appropriate signals or through the  
116382 use of some programmatic semaphore. POSIX.1 does not address the more general problem of  
116383 synchronizing access to shared data structures. Note in particular that even the “safe” functions  
116384 may modify the global variable *errno*; the signal-catching function may want to save and restore  
116385 its value. The same principles apply to the reentrancy of application routines and asynchronous  
116386 data access.

116387 Note that *longjmp()* and *siglongjmp()* are not in the list of reentrant functions. This is because the  
116388 code executing after *longjmp()* or *siglongjmp()* can call any unsafe functions with the same  
116389 danger as calling those unsafe functions directly from the signal handler. Applications that use  
116390 *longjmp()* or *siglongjmp()* out of signal handlers require rigorous protection in order to be  
116391 portable. Many of the other functions that are excluded from the list are traditionally  
116392 implemented using either the C language *malloc()* or *free()* functions or the ISO C standard I/O  
116393 library, both of which traditionally use data structures in a non-reentrant manner. Because any  
116394 combination of different functions using a common data structure can cause reentrancy  
116395 problems, POSIX.1 does not define the behavior when any unsafe function is called in a signal  
116396 handler that interrupts any unsafe function.

116397 The only realtime extension to signal actions is the addition of the additional parameters to the  
116398 signal-catching function. This extension has been explained and motivated in the previous  
116399 section. In making this extension, though, developers of POSIX.1b ran into issues relating to  
116400 function prototypes. In response to input from the POSIX.1 standard developers, members were  
116401 added to the **sigaction** structure to specify function prototypes for the newer signal-catching  
116402 function specified by POSIX.1b. These members follow changes that are being made to POSIX.1.  
116403 Note that POSIX.1-200x explicitly states that these fields may overlap so that a union can be  
116404 defined. This enabled existing implementations of POSIX.1 to maintain binary-compatibility  
116405 when these extensions were added.

116406 The **siginfo\_t** structure was adopted for passing the application-defined value to match existing  
116407 practice, but the existing practice has no provision for an application-defined value, so this was  
116408 added. Note that POSIX normally reserves the “\_t” type designation for opaque types. The  
116409 **siginfo\_t** structure breaks with this convention to follow existing practice and thus promote  
116410 portability. Standardization of the existing practice for the other members of this structure may  
116411 be addressed in the future.

116412 Although it is not explicitly visible to applications, there are additional semantics for signal  
116413 actions implied by queued signals and their interaction with other POSIX.1b realtime functions.  
116414 Specifically:

- 116415 • It is not necessary to queue signals whose action is SIG\_IGN.
- 116416 • For implementations that support POSIX.1b timers, some interaction with the timer  
116417 functions at signal delivery is implied to manage the timer overrun count.

116418 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/5 is applied, reordering the RTS shaded  
116419 text under the third and fourth paragraphs of the SIG\_DFL description. This corrects an earlier  
116420 editorial error in this section.

116421 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/6 is applied, adding the *abort()* function  
116422 to the list of async-cancel-safe functions.

116423 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/4 is applied, adding the *socketmark()*  
116424 function to the list of functions that shall be either reentrant or non-interruptible by signals and  
116425 shall be async-signal-safe.

116426 **B.2.4.4** *Signal Effects on Other Functions*

116427 The most common behavior of an interrupted function after a signal-catching function returns is  
 116428 for the interrupted function to give an [EINTR] error unless the SA\_RESTART flag is in effect for  
 116429 the signal. However, there are a number of specific exceptions, including *sleep()* and certain  
 116430 situations with *read()* and *write()*.

116431 The historical implementations of many functions defined by POSIX.1-200x are not interruptible,  
 116432 but delay delivery of signals generated during their execution until after they complete. This is  
 116433 never a problem for functions that are guaranteed to complete in a short (imperceptible to a  
 116434 human) period of time. It is normally those functions that can suspend a process indefinitely or  
 116435 for long periods of time (for example, *wait()*, *pause()*, *sigsuspend()*, *sleep()*, or *read()/write()* on a  
 116436 slow device like a terminal) that are interruptible. This permits applications to respond to  
 116437 interactive signals or to set timeouts on calls to most such functions with *alarm()*. Therefore,  
 116438 implementations should generally make such functions (including ones defined as extensions)  
 116439 interruptible.

116440 Functions not mentioned explicitly as interruptible may be so on some implementations,  
 116441 possibly as an extension where the function gives an [EINTR] error. There are several functions  
 116442 (for example, *getpid()*, *getuid()*) that are specified as never returning an error, which can thus  
 116443 never be extended in this way.

116444 If a signal-catching function returns while the SA\_RESTART flag is in effect, an interrupted  
 116445 function is restarted at the point it was interrupted. Conforming applications cannot make  
 116446 assumptions about the internal behavior of interrupted functions, even if the functions are  
 116447 async-signal-safe. For example, suppose the *read()* function is interrupted with SA\_RESTART in  
 116448 effect, the signal-catching function closes the file descriptor being read from and returns, and the  
 116449 *read()* function is then restarted; in this case the application cannot assume that the *read()*  
 116450 function will give an [EBADF] error, since *read()* might have checked the file descriptor for  
 116451 validity before being interrupted.

116452 **B.2.5** **Standard I/O Streams**116453 **B.2.5.1** *Interaction of File Descriptors and Standard I/O Streams*

116454 There is no additional rationale provided for this section.

116455 **B.2.5.2** *Stream Orientation and Encoding Rules*

116456 There is no additional rationale provided for this section.

116457 **B.2.6** **STREAMS**

116458 STREAMS are included into POSIX.1-200x as part of the alignment with the Single UNIX  
 116459 Specification, but marked as an option in recognition that not all systems may wish to  
 116460 implement the facility. The option within POSIX.1-200x is denoted by the XSR margin marker.  
 116461 The standard developers made this option independent of the XSI option. In this version of the  
 116462 standard this option is marked obsolescent.

116463 STREAMS are a method of implementing network services and other character-based  
 116464 input/output mechanisms, with the STREAM being a full-duplex connection between a process  
 116465 and a device. STREAMS provides direct access to protocol modules, and optional protocol  
 116466 modules can be interposed between the process-end of the STREAM and the device-driver at the  
 116467 device-end of the STREAM. Pipes can be implemented using the STREAMS mechanism, so they  
 116468 can provide process-to-process as well as process-to-device communications.



116469 This section introduces STREAMS I/O, the message types used to control them, an overview of  
116470 the priority mechanism, and the interfaces used to access them.

#### 116471 B.2.6.1 Accessing STREAMS

116472 There is no additional rationale provided for this section.

### 116473 B.2.7 XSI Interprocess Communication

116474 There are two forms of IPC supported as options in POSIX.1-200x. The traditional System V IPC  
116475 routines derived from the SVID—that is, the *msg\**(), *sem\**(), and *shm\**() interfaces—are  
116476 mandatory on XSI-conformant systems. Thus, all XSI-conformant systems provide the same  
116477 mechanisms for manipulating messages, shared memory, and semaphores.

116478 In addition, the POSIX Realtime Extension provides an alternate set of routines for those systems  
116479 supporting the appropriate options.

116480 The application developer is presented with a choice: the System V interfaces or the POSIX  
116481 interfaces (loosely derived from the Berkeley interfaces). The XSI profile prefers the System V  
116482 interfaces, but the POSIX interfaces may be more suitable for realtime or other performance-  
116483 sensitive applications.

#### 116484 B.2.7.1 IPC General Information

116485 General information that is shared by all three mechanisms is described in this section. The  
116486 common permissions mechanism is briefly introduced, describing the mode bits, and how they  
116487 are used to determine whether or not a process has access to read or write/alter the appropriate  
116488 instance of one of the IPC mechanisms. All other relevant information is contained in the  
116489 reference pages themselves.

116490 The semaphore type of IPC allows processes to communicate through the exchange of  
116491 semaphore values. A semaphore is a positive integer. Since many applications require the use of  
116492 more than one semaphore, XSI-conformant systems have the ability to create sets or arrays of  
116493 semaphores.

116494 Calls to support semaphores include:

116495 *semctl*(), *semget*(), *semop*()

116496 Semaphore sets are created by using the *semget*() function.

116497 The message type of IPC allows processes to communicate through the exchange of data stored  
116498 in buffers. This data is transmitted between processes in discrete portions known as messages.

116499 Calls to support message queues include:

116500 *msgctl*(), *msgget*(), *msgrcv*(), *msgsnd*()

116501 The shared memory type of IPC allows two or more processes to share memory and  
116502 consequently the data contained therein. This is done by allowing processes to set up access to a  
116503 common memory address space. This sharing of memory provides a fast means of exchange of  
116504 data between processes.

116505 Calls to support shared memory include:

116506 *shmctl*(), *shmdt*(), *shmget*()

116507 The *ftok*() interface is also provided.

## B.2.8 Realtime

### Advisory Information

POSIX.1b contains an Informative Annex with proposed interfaces for “realtime files”. These interfaces could determine groups of the exact parameters required to do “direct I/O” or “extents”. These interfaces were objected to by a significant portion of the balloting group as too complex. A conforming application had little chance of correctly navigating the large parameter space to match its desires to the system. In addition, they only applied to a new type of file (realtime files) and they told the implementation exactly what to do as opposed to advising the implementation on application behavior and letting it optimize for the system the (portable) application was running on. For example, it was not clear how a system that had a disk array should set its parameters.

There seemed to be several overall goals:

- Optimizing sequential access
- Optimizing caching behavior
- Optimizing I/O data transfer
- Preallocation

The advisory interfaces, *posix\_fadvise()* and *posix\_madvise()*, satisfy the first two goals. The `POSIX_FADV_SEQUENTIAL` and `POSIX_MADV_SEQUENTIAL` advice tells the implementation to expect serial access. Typically the system will prefetch the next several serial accesses in order to overlap I/O. It may also free previously accessed serial data if memory is tight. If the application is not doing serial access it can use `POSIX_FADV_WILLNEED` and `POSIX_MADV_WILLNEED` to accomplish I/O overlap, as required. When the application advises `POSIX_FADV_RANDOM` or `POSIX_MADV_RANDOM` behavior, the implementation usually tries to fetch a minimum amount of data with each request and it does not expect much locality. `POSIX_FADV_DONTNEED` and `POSIX_MADV_DONTNEED` allow the system to free up caching resources as the data will not be required in the near future.

`POSIX_FADV_NOREUSE` tells the system that caching the specified data is not optimal. For file I/O, the transfer should go directly to the user buffer instead of being cached internally by the implementation. To portably perform direct disk I/O on all systems, the application must perform its I/O transfers according to the following rules:

1. The user buffer should be aligned according to the `{POSIX_REC_XFER_ALIGN} pathconf()` variable.
2. The number of bytes transferred in an I/O operation should be a multiple of the `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
3. The offset into the file at the start of an I/O operation should be a multiple of the `{POSIX_ALLOC_SIZE_MIN} pathconf()` variable.
4. The application should ensure that all threads which open a given file specify `POSIX_FADV_NOREUSE` to be sure that there is no unexpected interaction between threads using buffered I/O and threads using direct I/O to the same file.

In some cases, a user buffer must be properly aligned in order to be transferred directly to/from the device. The `{POSIX_REC_XFER_ALIGN} pathconf()` variable tells the application the proper alignment.

The preallocation goal is met by the space control function, *posix\_fallocate()*. The application can use *posix\_fallocate()* to guarantee no `[ENOSPC]` errors and to improve performance by prepaying any overhead required for block allocation.

116553 Implementations may use information conveyed by a previous *posix\_fadvise()* call to influence  
 116554 the manner in which allocation is performed. For example, if an application did the following  
 116555 calls:

```
116556 fd = open("file");
116557 posix_fadvise(fd, offset, len, POSIX_FADV_SEQUENTIAL);
116558 posix_fallocate(fd, len, size);
```

116559 an implementation might allocate the file contiguously on disk.

116560 Finally, the *pathconf()* variables {POSIX\_REC\_MIN\_XFER\_SIZE},  
 116561 {POSIX\_REC\_MAX\_XFER\_SIZE}, and {POSIX\_REC\_INCR\_XFER\_SIZE} tell the application a  
 116562 range of transfer sizes that are recommended for best I/O performance.

116563 Where bounded response time is required, the vendor can supply the appropriate settings of the  
 116564 advisories to achieve a guaranteed performance level.

116565 The interfaces meet the goals while allowing applications using regular files to take advantage  
 116566 of performance optimizations. The interfaces tell the implementation expected application  
 116567 behavior which the implementation can use to optimize performance on a particular system  
 116568 with a particular dynamic load.

116569 The *posix\_memalign()* function was added to allow for the allocation of specifically aligned  
 116570 buffers; for example, for {POSIX\_REC\_XFER\_ALIGN}.

116571 The working group also considered the alternative of adding a function which would return an  
 116572 aligned pointer to memory within a user-supplied buffer. This was not considered to be the best  
 116573 method, because it potentially wastes large amounts of memory when buffers need to be aligned  
 116574 on large alignment boundaries.

## 116575 Message Passing

116576 This section provides the rationale for the definition of the message passing interface in  
 116577 POSIX.1-200x. This is presented in terms of the objectives, models, and requirements imposed  
 116578 upon this interface.

- 116579 • Objectives

116580 Many applications, including both realtime and database applications, require a means of  
 116581 passing arbitrary amounts of data between cooperating processes comprising the overall  
 116582 application on one or more processors. Many conventional interfaces for interprocess  
 116583 communication are insufficient for realtime applications in that efficient and deterministic  
 116584 data passing methods cannot be implemented. This has prompted the definition of  
 116585 message passing interfaces providing these facilities:

- 116586 — Open a message queue.
- 116587 — Send a message to a message queue.
- 116588 — Receive a message from a queue, either synchronously or asynchronously.
- 116589 — Alter message queue attributes for flow and resource control.

116590 It is assumed that an application may consist of multiple cooperating processes and that  
 116591 these processes may wish to communicate and coordinate their activities. The message  
 116592 passing facility described in POSIX.1-200x allows processes to communicate through  
 116593 system-wide queues. These message queues are accessed through names that may be  
 116594 pathnames. A message queue can be opened for use by multiple sending and/or multiple  
 116595 receiving processes.

- 116596
- Background on Embedded Applications
- 116597 Interprocess communication utilizing message passing is a key facility for the construction  
116598 of deterministic, high-performance realtime applications. The facility is present in all  
116599 realtime systems and is the framework upon which the application is constructed. The  
116600 performance of the facility is usually a direct indication of the performance of the resulting  
116601 application.
- 116602 Realtime applications, especially for embedded systems, are typically designed around the  
116603 performance constraints imposed by the message passing mechanisms. Applications for  
116604 embedded systems are typically very tightly constrained. Application developers expect to  
116605 design and control the entire system. In order to minimize system costs, the writer will  
116606 attempt to use all resources to their utmost and minimize the requirement to add  
116607 additional memory or processors.
- 116608 The embedded applications usually share address spaces and only a simple message  
116609 passing mechanism is required. The application can readily access common data incurring  
116610 only mutual-exclusion overheads. The models desired are the simplest possible with the  
116611 application building higher-level facilities only when needed.
- Requirements
- 116612 The following requirements determined the features of the message passing facilities  
116613 defined in POSIX.1-200x:  
116614
- Naming of Message Queues
- 116615 The mechanism for gaining access to a message queue is a pathname evaluated in a  
116616 context that is allowed to be a file system name space, or it can be independent of  
116617 any file system. This is a specific attempt to allow implementations based on either  
116618 method in order to address both embedded systems and to also allow  
116619 implementation in larger systems.  
116620
- 116621 The interface of `mq_open()` is defined to allow but not require the access control and  
116622 name conflicts resulting from utilizing a file system for name resolution. All required  
116623 behavior is specified for the access control case. Yet a conforming implementation,  
116624 such as an embedded system kernel, may define that there are no distinctions  
116625 between users and may define that all processes have all access privileges.
- Embedded System Naming
- 116626 Embedded systems need to be able to utilize independent name spaces for accessing  
116627 the various system objects. They typically do not have a file system, precluding its  
116628 utilization as a common name resolution mechanism. The modularity of an  
116629 embedded system limits the connections between separate mechanisms that can be  
116630 allowed.  
116631
- 116632 Embedded systems typically do not have any access protection. Since the system  
116633 does not support the mixing of applications from different areas, and usually does  
116634 not even have the concept of an authorization entity, access control is not useful.
- Large System Naming
- 116635 On systems with more functionality, the name resolution must support the ability to  
116636 use the file system as the name resolution mechanism/object storage medium and to  
116637 have control over access to the objects. Utilizing the pathname space can result in  
116638 further errors when the names conflict with other objects.  
116639

## 116640 — Fixed Size of Messages

116641 The interfaces impose a fixed upper bound on the size of messages that can be sent to  
 116642 a specific message queue. The size is set on an individual queue basis and cannot be  
 116643 changed dynamically.

116644 The purpose of the fixed size is to increase the ability of the system to optimize the  
 116645 implementation of *mq\_send()* and *mq\_receive()*. With fixed sizes of messages and  
 116646 fixed numbers of messages, specific message blocks can be pre-allocated. This  
 116647 eliminates a significant amount of checking for errors and boundary conditions.  
 116648 Additionally, an implementation can optimize data copying to maximize  
 116649 performance. Finally, with a restricted range of message sizes, an implementation is  
 116650 better able to provide deterministic operations.

## 116651 — Prioritization of Messages

116652 Message prioritization allows the application to determine the order in which  
 116653 messages are received. Prioritization of messages is a key facility that is provided by  
 116654 most realtime kernels and is heavily utilized by the applications. The major purpose  
 116655 of having priorities in message queues is to avoid priority inversions in the message  
 116656 system, where a high-priority message is delayed behind one or more lower-priority  
 116657 messages. This allows the applications to be designed so that they do not need to be  
 116658 interrupted in order to change the flow of control when exceptional conditions occur.  
 116659 The prioritization does add additional overhead to the message operations in those  
 116660 cases it is actually used but a clever implementation can optimize for the FIFO case to  
 116661 make that more efficient.

## 116662 — Asynchronous Notification

116663 The interface supports the ability to have a task asynchronously notified of the  
 116664 availability of a message on the queue. The purpose of this facility is to allow the task  
 116665 to perform other functions and yet still be notified that a message has become  
 116666 available on the queue.

116667 To understand the requirement for this function, it is useful to understand two  
 116668 models of application design: a single task performing multiple functions and  
 116669 multiple tasks performing a single function. Each of these models has advantages.

116670 Asynchronous notification is required to build the model of a single task performing  
 116671 multiple operations. This model typically results from either the expectation that  
 116672 interruption is less expensive than utilizing a separate task or from the growth of the  
 116673 application to include additional functions.

116674 **Semaphores**

116675 Semaphores are a high-performance process synchronization mechanism. Semaphores are  
 116676 named by null-terminated strings of characters.

116677 A semaphore is created using the *sem\_init()* function or the *sem\_open()* function with the  
 116678 *O\_CREAT* flag set in *oflag*.

116679 To use a semaphore, a process has to first initialize the semaphore or inherit an open descriptor  
 116680 for the semaphore via *fork()*.

116681 A semaphore preserves its state when the last reference is closed. For example, if a semaphore  
 116682 has a value of 13 when the last reference is closed, it will have a value of 13 when it is next  
 116683 opened.

116684 When a semaphore is created, an initial state for the semaphore has to be provided. This value is

116685 a non-negative integer. Negative values are not possible since they indicate the presence of  
 116686 blocked processes. The persistence of any of these objects across a system crash or a system  
 116687 reboot is undefined. Conforming applications must not depend on any sort of persistence across  
 116688 a system reboot or a system crash.

116689 • Models and Requirements

116690 A realtime system requires synchronization and communication between the processes  
 116691 comprising the overall application. An efficient and reliable synchronization mechanism  
 116692 has to be provided in a realtime system that will allow more than one schedulable process  
 116693 mutually-exclusive access to the same resource. This synchronization mechanism has to  
 116694 allow for the optimal implementation of synchronization or systems implementors will  
 116695 define other, more cost-effective methods.

116696 At issue are the methods whereby multiple processes (tasks) can be designed and  
 116697 implemented to work together in order to perform a single function. This requires  
 116698 interprocess communication and synchronization. A semaphore mechanism is the lowest  
 116699 level of synchronization that can be provided by an operating system.

116700 A semaphore is defined as an object that has an integral value and a set of blocked  
 116701 processes associated with it. If the value is positive or zero, then the set of blocked  
 116702 processes is empty; otherwise, the size of the set is equal to the absolute value of the  
 116703 semaphore value. The value of the semaphore can be incremented or decremented by any  
 116704 process with access to the semaphore and must be done as an indivisible operation. When  
 116705 a semaphore value is less than or equal to zero, any process that attempts to lock it again  
 116706 will block or be informed that it is not possible to perform the operation.

116707 A semaphore may be used to guard access to any resource accessible by more than one  
 116708 schedulable task in the system. It is a global entity and not associated with any particular  
 116709 process. As such, a method of obtaining access to the semaphore has to be provided by the  
 116710 operating system. A process that wants access to a critical resource (section) has to wait on  
 116711 the semaphore that guards that resource. When the semaphore is locked on behalf of a  
 116712 process, it knows that it can utilize the resource without interference by any other  
 116713 cooperating process in the system. When the process finishes its operation on the resource,  
 116714 leaving it in a well-defined state, it posts the semaphore, indicating that some other  
 116715 process may now obtain the resource associated with that semaphore.

116716 In this section, mutexes and condition variables are specified as the synchronization  
 116717 mechanisms between threads.

116718 These primitives are typically used for synchronizing threads that share memory in a  
 116719 single process. However, this section provides an option allowing the use of these  
 116720 synchronization interfaces and objects between processes that share memory, regardless of  
 116721 the method for sharing memory.

116722 Much experience with semaphores shows that there are two distinct uses of  
 116723 synchronization: locking, which is typically of short duration; and waiting, which is  
 116724 typically of long or unbounded duration. These distinct usages map directly onto mutexes  
 116725 and condition variables, respectively.

116726 Semaphores are provided in POSIX.1-200x primarily to provide a means of  
 116727 synchronization for processes; these processes may or may not share memory. Mutexes  
 116728 and condition variables are specified as synchronization mechanisms between threads;  
 116729 these threads always share (some) memory. Both are synchronization paradigms that have  
 116730 been in widespread use for a number of years. Each set of primitives is particularly well  
 116731 matched to certain problems.

116732 With respect to binary semaphores, experience has shown that condition variables and

116733 mutexes are easier to use for many synchronization problems than binary semaphores. The  
 116734 primary reason for this is the explicit appearance of a Boolean predicate that specifies  
 116735 when the condition wait is satisfied. This Boolean predicate terminates a loop, including  
 116736 the call to *pthread\_cond\_wait()*. As a result, extra wakeups are benign since the predicate  
 116737 governs whether the thread will actually proceed past the condition wait. With stateful  
 116738 primitives, such as binary semaphores, the wakeup in itself typically means that the wait is  
 116739 satisfied. The burden of ensuring correctness for such waits is thus placed on *all* signalers  
 116740 of the semaphore rather than on an *explicitly coded* Boolean predicate located at the  
 116741 condition wait. Experience has shown that the latter creates a major improvement in safety  
 116742 and ease-of-use.

116743 Counting semaphores are well matched to dealing with producer/consumer problems,  
 116744 including those that might exist between threads of different processes, or between a signal  
 116745 handler and a thread. In the former case, there may be little or no memory shared by the  
 116746 processes; in the latter case, one is not communicating between co-equal threads, but  
 116747 between a thread and an interrupt-like entity. It is for these reasons that POSIX.1-200x  
 116748 allows semaphores to be used by threads.

116749 Mutexes and condition variables have been effectively used with and without priority  
 116750 inheritance, priority ceiling, and other attributes to synchronize threads that share  
 116751 memory. The efficiency of their implementation is comparable to or better than that of  
 116752 other synchronization primitives that are sometimes harder to use (for example, binary  
 116753 semaphores). Furthermore, there is at least one known implementation of Ada tasking that  
 116754 uses these primitives. Mutexes and condition variables together constitute an appropriate,  
 116755 sufficient, and complete set of inter-thread synchronization primitives.

116756 Efficient multi-threaded applications require high-performance synchronization  
 116757 primitives. Considerations of efficiency and generality require a small set of primitives  
 116758 upon which more sophisticated synchronization functions can be built.

116759 • Standardization Issues

116760 It is possible to implement very high-performance semaphores using test-and-set  
 116761 instructions on shared memory locations. The library routines that implement such a high-  
 116762 performance interface have to properly ensure that a *sem\_wait()* or *sem\_trywait()*  
 116763 operation that cannot be performed will issue a blocking semaphore system call or properly report  
 116764 the condition to the application. The same interface to the application program would be  
 116765 provided by a high-performance implementation.

116766 **B.2.8.1 Realtime Signals**

116767 **Realtime Signals Extension**

116768 This portion of the rationale presents models, requirements, and standardization issues relevant  
 116769 to the Realtime Signals Extension. This extension provides the capability required to support  
 116770 reliable, deterministic, asynchronous notification of events. While a new mechanism,  
 116771 unencumbered by the historical usage and semantics of POSIX.1 signals, might allow for a more  
 116772 efficient implementation, the application requirements for event notification can be met with a  
 116773 small number of extensions to signals. Therefore, a minimal set of extensions to signals to  
 116774 support the application requirements is specified.

116775 The realtime signal extensions specified in this section are used by other realtime functions  
 116776 requiring asynchronous notification:

116777 • Models

116778 The model supported is one of multiple cooperating processes, each of which handles  
 116779 multiple asynchronous external events. Events represent occurrences that are generated as

116780 the result of some activity in the system. Examples of occurrences that can constitute an  
116781 event include:

- 116782 — Completion of an asynchronous I/O request
- 116783 — Expiration of a POSIX.1b timer
- 116784 — Arrival of an interprocess message
- 116785 — Generation of a user-defined event

116786 Processing of these events may occur synchronously via polling for event notifications or  
116787 asynchronously via a software interrupt mechanism. Existing practice for this model is  
116788 well established for traditional proprietary realtime operating systems, realtime  
116789 executives, and realtime extended POSIX-like systems.

116790 A contrasting model is that of “cooperating sequential processes” where each process  
116791 handles a single priority of events via polling. Each process blocks while waiting for  
116792 events, and each process depends on the preemptive, priority-based process scheduling  
116793 mechanism to arbitrate between events of different priority that need to be processed  
116794 concurrently. Existing practice for this model is also well established for small realtime  
116795 executives that typically execute in an unprotected physical address space, but it is just  
116796 emerging in the context of a fuller function operating system with multiple virtual address  
116797 spaces.

116798 It could be argued that the cooperating sequential process model, and the facilities  
116799 supported by the POSIX Threads Extension obviate a software interrupt model. But, even  
116800 with the cooperating sequential process model, the need has been recognized for a  
116801 software interrupt model to handle exceptional conditions and process aborting, so the  
116802 mechanism must be supported in any case. Furthermore, it is not the purview of  
116803 POSIX.1-200x to attempt to convince realtime practitioners that their current application  
116804 models based on software interrupts are “broken” and should be replaced by the  
116805 cooperating sequential process model. Rather, it is the charter of POSIX.1-200x to provide  
116806 standard extensions to mechanisms that support existing realtime practice.

#### 116807 • Requirements

116808 This section discusses the following realtime application requirements for asynchronous  
116809 event notification:

- 116810 — Reliable delivery of asynchronous event notification

116811 The events notification mechanism guarantees delivery of an event notification.  
116812 Asynchronous operations (such as asynchronous I/O and timers) that complete  
116813 significantly after they are invoked have to guarantee that delivery of the event  
116814 notification can occur at the time of completion.

- 116815 — Prioritized handling of asynchronous event notifications

116816 The events notification mechanism supports the assigning of a user function as an  
116817 event notification handler. Furthermore, the mechanism supports the preemption of  
116818 an event handler function by a higher priority event notification and supports the  
116819 selection of the highest priority pending event notification when multiple  
116820 notifications (of different priority) are pending simultaneously.

116821 The model here is based on hardware interrupts. Asynchronous event handling  
116822 allows the application to ensure that time-critical events are immediately processed  
116823 when delivered, without the indeterminism of being at a random location within a  
116824 polling loop. Use of handler priority allows the specification of how handlers are  
116825 interrupted by other higher priority handlers.



- 116826 — Differentiation between multiple occurrences of event notifications of the same type
- 116827 The events notification mechanism passes an application-defined value to the event  
116828 handler function. This value can be used for a variety of purposes, such as enabling  
116829 the application to identify which of several possible events of the same type (for  
116830 example, timer expirations) has occurred.
- 116831 — Polled reception of asynchronous event notifications
- 116832 The events notification mechanism supports blocking and non-blocking polls for  
116833 asynchronous event notification.
- 116834 The polled mode of operation is often preferred over the interrupt mode by those  
116835 practitioners accustomed to this model. Providing support for this model facilitates  
116836 the porting of applications based on this model to POSIX.1b conforming systems.
- 116837 — Deterministic response to asynchronous event notifications
- 116838 The events notification mechanism does not preclude implementations that provide  
116839 deterministic event dispatch latency and minimizes the number of system calls  
116840 needed to use the event facilities during realtime processing.
- 116841 • Rationale for Extension
- 116842 POSIX.1 signals have many of the characteristics necessary to support the asynchronous  
116843 handling of event notifications, and the Realtime Signals Extension addresses the  
116844 following deficiencies in the POSIX.1 signal mechanism:
- 116845 — Signals do not support reliable delivery of event notification. Subsequent  
116846 occurrences of a pending signal are not guaranteed to be delivered.
- 116847 — Signals do not support prioritized delivery of event notifications. The order of signal  
116848 delivery when multiple unblocked signals are pending is undefined.
- 116849 — Signals do not support the differentiation between multiple signals of the same type.

### 116850 B.2.8.2 Asynchronous I/O

116851 Many applications need to interact with the I/O subsystem in an asynchronous manner. The  
116852 asynchronous I/O mechanism provides the ability to overlap application processing and I/O  
116853 operations initiated by the application. The asynchronous I/O mechanism allows a single  
116854 process to perform I/O simultaneously to a single file multiple times or to multiple files  
116855 multiple times.

#### 116856 Overview

116857 Asynchronous I/O operations proceed in logical parallel with the processing done by the  
116858 application after the asynchronous I/O has been initiated. Other than this difference,  
116859 asynchronous I/O behaves similarly to normal I/O using *read()*, *write()*, *lseek()*, and *fsync()*.  
116860 The effect of issuing an asynchronous I/O request is as if a separate thread of execution were to  
116861 perform atomically the implied *lseek()* operation, if any, and then the requested I/O operation  
116862 (either *read()*, *write()*, or *fsync()*). There is no seek implied with a call to *aio\_fsync()*. Concurrent  
116863 asynchronous operations and synchronous operations applied to the same file update the file as  
116864 if the I/O operations had proceeded serially.

116865 When asynchronous I/O completes, a signal can be delivered to the application to indicate the  
116866 completion of the I/O. This signal can be used to indicate that buffers and control blocks used  
116867 for asynchronous I/O can be reused. Signal delivery is not required for an asynchronous  
116868 operation and may be turned off on a per-operation basis by the application. Signals may also be  
116869 synchronously polled using *aio\_suspend()*, *sigtimedwait()*, or *sigwaitinfo()*.

116870 Normal I/O has a return value and an error status associated with it. Asynchronous I/O  
 116871 returns a value and an error status when the operation is first submitted, but that only relates to  
 116872 whether the operation was successfully queued up for servicing. The I/O operation itself also  
 116873 has a return status and an error value. To allow the application to retrieve the return status and  
 116874 the error value, functions are provided that, given the address of an asynchronous I/O control  
 116875 block, yield the return and error status associated with the operation. Until an asynchronous I/O  
 116876 operation is done, its error status is [EINPROGRESS]. Thus, an application can poll for  
 116877 completion of an asynchronous I/O operation by waiting for the error status to become equal to  
 116878 a value other than [EINPROGRESS]. The return status of an asynchronous I/O operation is  
 116879 undefined so long as the error status is equal to [EINPROGRESS].

116880 Storage for asynchronous operation return and error status may be limited. Submission of  
 116881 asynchronous I/O operations may fail if this storage is exceeded. When an application retrieves  
 116882 the return status of a given asynchronous operation, therefore, any system-maintained storage  
 116883 used for this status and the error status may be reclaimed for use by other asynchronous  
 116884 operations.

116885 Asynchronous I/O can be performed on file descriptors that have been enabled for POSIX.1b  
 116886 synchronized I/O. In this case, the I/O operation still occurs asynchronously, as defined herein;  
 116887 however, the asynchronous operation I/O in this case is not completed until the I/O has reached  
 116888 either the state of synchronized I/O data integrity completion or synchronized I/O file integrity  
 116889 completion, depending on the sort of synchronized I/O that is enabled on the file descriptor.

## 116890 Models

116891 Three models illustrate the use of asynchronous I/O: a journalization model, a data acquisition  
 116892 model, and a model of the use of asynchronous I/O in supercomputing applications.

- 116893 • Journalization Model

116894 Many realtime applications perform low-priority journalizing functions. Journalizing  
 116895 requires that logging records be queued for output without blocking the initiating process.

- 116896 • Data Acquisition Model

116897 A data acquisition process may also serve as a model. The process has two or more  
 116898 channels delivering intermittent data that must be read within a certain time. The process  
 116899 issues one asynchronous read on each channel. When one of the channels needs data  
 116900 collection, the process reads the data and posts it through an asynchronous write to  
 116901 secondary memory for future processing.

- 116902 • Supercomputing Model

116903 The supercomputing community has used asynchronous I/O much like that specified in  
 116904 POSIX.1 for many years. This community requires the ability to perform multiple I/O  
 116905 operations to multiple devices with a minimal number of entries to “the system”; each  
 116906 entry to “the system” provokes a major delay in operations when compared to the normal  
 116907 progress made by the application. This existing practice motivated the use of combined  
 116908 *lseek()* and *read()* or *write()* calls, as well as the *lio\_listio()* call. Another common practice is  
 116909 to disable signal notification for I/O completion, and simply poll for I/O completion at  
 116910 some interval by which the I/O should be completed. Likewise, interfaces like *aio\_cancel()*  
 116911 have been in successful commercial use for many years. Note also that an underlying  
 116912 implementation of asynchronous I/O will require the ability, at least internally, to cancel  
 116913 outstanding asynchronous I/O, at least when the process exits. (Consider an asynchronous  
 116914 read from a terminal, when the process intends to exit immediately.)

116915 **Requirements**

116916 Asynchronous input and output for realtime implementations have these requirements:

- 116917 • The ability to queue multiple asynchronous read and write operations to a single open
- 116918 instance. Both sequential and random access should be supported.
- 116919 • The ability to queue asynchronous read and write operations to multiple open instances.
- 116920 • The ability to obtain completion status information by polling and/or asynchronous event
- 116921 notification.
- 116922 • Asynchronous event notification on asynchronous I/O completion is optional.
- 116923 • It has to be possible for the application to associate the event with the *aiocbp* for the
- 116924 operation that generated the event.
- 116925 • The ability to cancel queued requests.
- 116926 • The ability to wait upon asynchronous I/O completion in conjunction with other types of
- 116927 events.
- 116928 • The ability to accept an *aio\_read()* and an *aio\_cancel()* for a device that accepts a *read()*, and
- 116929 the ability to accept an *aio\_write()* and an *aio\_cancel()* for a device that accepts a *write()*.
- 116930 This does not imply that the operation is asynchronous.

116931 **Standardization Issues**

116932 The following issues are addressed by the standardization of asynchronous I/O:

## 116933 • Rationale for New Interface

116934 Non-blocking I/O does not satisfy the needs of either realtime or high-performance  
 116935 computing models; these models require that a process overlap program execution and  
 116936 I/O processing. Realtime applications will often make use of direct I/O to or from the  
 116937 address space of the process, or require synchronized (unbuffered) I/O; they also require  
 116938 the ability to overlap this I/O with other computation. In addition, asynchronous I/O  
 116939 allows an application to keep a device busy at all times, possibly achieving greater  
 116940 throughput. Supercomputing and database architectures will often have specialized  
 116941 hardware that can provide true asynchrony underlying the logical asynchrony provided  
 116942 by this interface. In addition, asynchronous I/O should be supported by all types of files  
 116943 and devices in the same manner.

## 116944 • Effect of Buffering

116945 If asynchronous I/O is performed on a file that is buffered prior to being actually written  
 116946 to the device, it is possible that asynchronous I/O will offer no performance advantage  
 116947 over normal I/O; the cycles *stolen* to perform the asynchronous I/O will be taken away  
 116948 from the running process and the I/O will occur at interrupt time. This potential lack of  
 116949 gain in performance in no way obviates the need for asynchronous I/O by realtime  
 116950 applications, which very often will use specialized hardware support, multiple processors,  
 116951 and/or unbuffered, synchronized I/O.

116952 **B.2.8.3 Memory Management**

116953 All memory management and shared memory definitions are located in the `<sys/mman.h>`  
 116954 header. This is for alignment with historical practice.

116955 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/7 is applied, correcting the shading and  
 116956 margin markers in the introduction to Section 2.8.3.1.

116957 **Memory Locking Functions**

116958 This portion of the rationale presents models, requirements, and standardization issues relevant  
116959 to process memory locking.

## 116960 • Models

116961 Realtime systems that conform to POSIX.1-200x are expected (and desired) to be supported  
116962 on systems with demand-paged virtual memory management, non-paged swapping  
116963 memory management, and physical memory systems with no memory management  
116964 hardware. The general case, however, is the demand-paged, virtual memory system with  
116965 each POSIX process running in a virtual address space. Note that this includes  
116966 architectures where each process resides in its own virtual address space and architectures  
116967 where the address space of each process is only a portion of a larger global virtual address  
116968 space.

116969 The concept of memory locking is introduced to eliminate the indeterminacy introduced  
116970 by paging and swapping, and to support an upper bound on the time required to access  
116971 the memory mapped into the address space of a process. Ideally, this upper bound will be  
116972 the same as the time required for the processor to access “main memory”, including any  
116973 address translation and cache miss overheads. But some implementations—primarily on  
116974 mainframes—will not actually force locked pages to be loaded and held resident in main  
116975 memory. Rather, they will handle locked pages so that accesses to these pages will meet the  
116976 performance metrics for locked process memory in the implementation. Also, although it  
116977 is not, for example, the intention that this interface, as specified, be used to lock process  
116978 memory into “cache”, it is conceivable that an implementation could support a large static  
116979 RAM memory and define this as “main memory” and use a large[r] dynamic RAM as  
116980 “backing store”. These interfaces could then be interpreted as supporting the locking of  
116981 process memory into the static RAM. Support for multiple levels of backing store would  
116982 require extensions to these interfaces.

116983 Implementations may also use memory locking to guarantee a fixed translation between  
116984 virtual and physical addresses where such is beneficial to improving determinacy for  
116985 direct-to/from-process input/output. POSIX.1-200x does not guarantee to the application  
116986 that the virtual-to-physical address translations, if such exist, are fixed, because such  
116987 behavior would not be implementable on all architectures on which implementations of  
116988 POSIX.1-200x are expected. But POSIX.1-200x does mandate that an implementation  
116989 define, for the benefit of potential users, whether or not locking guarantees fixed  
116990 translations.

116991 Memory locking is defined with respect to the address space of a process. Only the pages  
116992 mapped into the address space of a process may be locked by the process, and when the  
116993 pages are no longer mapped into the address space—for whatever reason—the locks  
116994 established with respect to that address space are removed. Shared memory areas warrant  
116995 special mention, as they may be mapped into more than one address space or mapped  
116996 more than once into the address space of a process; locks may be established on pages  
116997 within these areas with respect to several of these mappings. In such a case, the lock state  
116998 of the underlying physical pages is the logical OR of the lock state with respect to each of  
116999 the mappings. Only when all such locks have been removed are the shared pages  
117000 considered unlocked.

117001 In recognition of the page granularity of Memory Management Units (MMU), and in order  
117002 to support locking of ranges of address space, memory locking is defined in terms of  
117003 “page” granularity. That is, for the interfaces that support an address and size specification  
117004 for the region to be locked, the address must be on a page boundary, and all pages mapped  
117005 by the specified range are locked, if valid. This means that the length is implicitly rounded

117006 up to a multiple of the page size. The page size is implementation-defined and is available  
117007 to applications as a compile-time symbolic constant or at runtime via *sysconf()*.

117008 A “real memory” POSIX.1b implementation that has no MMU could elect not to support  
117009 these interfaces, returning [ENOSYS]. But an application could easily interpret this as  
117010 meaning that the implementation would unconditionally page or swap the application  
117011 when such is not the case. It is the intention of POSIX.1-200x that such a system could  
117012 define these interfaces as “NO-OPs”, returning success without actually performing any  
117013 function except for mandated argument checking.

117014 • Requirements

117015 For realtime applications, memory locking is generally considered to be required as part of  
117016 application initialization. This locking is performed after an application has been loaded  
117017 (that is, *exec'd*) and the program remains locked for its entire lifetime. But to support  
117018 applications that undergo major mode changes where, in one mode, locking is required,  
117019 but in another it is not, the specified interfaces allow repeated locking and unlocking of  
117020 memory within the lifetime of a process.

117021 When a realtime application locks its address space, it should not be necessary for the  
117022 application to then “touch” all of the pages in the address space to guarantee that they are  
117023 resident or else suffer potential paging delays the first time the page is referenced. Thus,  
117024 POSIX.1-200x requires that the pages locked by the specified interfaces be resident when  
117025 the locking functions return successfully.

117026 Many architectures support system-managed stacks that grow automatically when the  
117027 current extent of the stack is exceeded. A realtime application has a requirement to be able  
117028 to “preallocate” sufficient stack space and lock it down so that it will not suffer page faults  
117029 to grow the stack during critical realtime operation. There was no consensus on a portable  
117030 way to specify how much stack space is needed, so POSIX.1-200x supports no specific  
117031 interface for preallocating stack space. But an application can portably lock down a specific  
117032 amount of stack space by specifying *MCL\_FUTURE* in a call to *mlockall()* and then calling  
117033 a dummy function that declares an automatic array of the desired size.

117034 Memory locking for realtime applications is also generally considered to be an “all or  
117035 nothing” proposition. That is, the entire process, or none, is locked down. But, for  
117036 applications that have well-defined sections that need to be locked and others that do not,  
117037 POSIX.1-200x supports an optional set of interfaces to lock or unlock a range of process  
117038 addresses. Reasons for locking down a specific range include:

117039 — An asynchronous event handler function that must respond to external events in a  
117040 deterministic manner such that page faults cannot be tolerated

117041 — An input/output “buffer” area that is the target for direct-to-process I/O, and the  
117042 overhead of implicit locking and unlocking for each I/O call cannot be tolerated

117043 Finally, locking is generally viewed as an “application-wide” function. That is, the  
117044 application is globally aware of which regions are locked and which are not over time. This  
117045 is in contrast to a function that is used temporarily within a “third party” library routine  
117046 whose function is unknown to the application, and therefore must have no “side effects”.  
117047 The specified interfaces, therefore, do not support “lock stacking” or “lock nesting” within  
117048 a process. But, for pages that are shared between processes or mapped more than once  
117049 into a process address space, “lock stacking” is essentially mandated by the requirement  
117050 that unlocking of pages that are mapped by more than one process or more than once by  
117051 the same process does not affect locks established on the other mappings.

117052 There was some support for “lock stacking” so that locking could be transparently used in  
117053 functions or opaque modules. But the consensus was not to burden all implementations

117054 with lock stacking (and reference counting), and an implementation option was proposed.  
 117055 There were strong objections to the option because applications would have to support  
 117056 both options in order to remain portable. The consensus was to eliminate lock stacking  
 117057 altogether, primarily through overwhelming support for the System V “m[un]lock[all]”  
 117058 interface on which POSIX.1-200x is now based.

117059 Locks are not inherited across *fork()*s because some implementations implement *fork()* by  
 117060 creating new address spaces for the child. In such an implementation, requiring locks to be  
 117061 inherited would lead to new situations in which a fork would fail due to the inability of  
 117062 the system to lock sufficient memory to lock both the parent and the child. The consensus  
 117063 was that there was no benefit to such inheritance. Note that this does not mean that locks  
 117064 are removed when, for instance, a thread is created in the same address space.

117065 Similarly, locks are not inherited across *exec* because some implementations implement *exec*  
 117066 by unmapping all of the pages in the address space (which, by definition, removes the  
 117067 locks on these pages), and maps in pages of the *exec*'d image. In such an implementation,  
 117068 requiring locks to be inherited would lead to new situations in which *exec* would fail.  
 117069 Reporting this failure would be very cumbersome to detect in time to report to the calling  
 117070 process, and no appropriate mechanism exists for informing the *exec*'d process of its status.

117071 It was determined that, if the newly loaded application required locking, it was the  
 117072 responsibility of that application to establish the locks. This is also in keeping with the  
 117073 general view that it is the responsibility of the application to be aware of all locks that are  
 117074 established.

117075 There was one request to allow (not mandate) locks to be inherited across *fork()*, and a  
 117076 request for a flag, *MCL\_INHERIT*, that would specify inheritance of memory locks across  
 117077 *execs*. Given the difficulties raised by this and the general lack of support for the feature in  
 117078 POSIX.1-200x, it was not added. POSIX.1-200x does not preclude an implementation from  
 117079 providing this feature for administrative purposes, such as a “run” command that will  
 117080 lock down and execute a specified application. Additionally, the rationale for the objection  
 117081 equated *fork()* with creating a thread in the address space. POSIX.1-200x does not mandate  
 117082 releasing locks when creating additional threads in an existing process.

117083 • Standardization Issues

117084 One goal of POSIX.1-200x is to define a set of primitives that provide the necessary  
 117085 functionality for realtime applications, with consideration for the needs of other  
 117086 application domains where such were identified, which is based to the extent possible on  
 117087 existing industry practice.

117088 The Memory Locking option is required by many realtime applications to tune  
 117089 performance. Such a facility is accomplished by placing constraints on the virtual memory  
 117090 system to limit paging of time of the process or of critical sections of the process. This  
 117091 facility should not be used by most non-realtime applications.

117092 Optional features provided in POSIX.1-200x allow applications to lock selected address  
 117093 ranges with the caveat that the process is responsible for being aware of the page  
 117094 granularity of locking and the unnested nature of the locks.

117095

**Mapped Files Functions**

117096

117097

117098

117099

117100

117101

117102

The memory mapped files functionality provides a mechanism that allows a process to access files by directly incorporating file data into its address space. Once a file is “mapped” into a process address space, the data can be manipulated by instructions as memory. The use of mapped files can significantly reduce I/O data movement since file data does not have to be copied into process data buffers as in *read()* and *write()*. If more than one process maps a file, its contents are shared among them. This provides a low overhead mechanism by which processes can synchronize and communicate.

117103

- Historical Perspective

117104

117105

117106

117107

Realtime applications have historically been implemented using a collection of cooperating processes or tasks. In early systems, these processes ran on bare hardware (that is, without an operating system) with no memory relocation or protection. The application paradigms that arose from this environment involve the sharing of data between the processes.

117108

117109

117110

117111

When realtime systems were implemented on top of vendor-supplied operating systems, the paradigm or performance benefits of direct access to data by multiple processes was still deemed necessary. As a result, operating systems that claim to support realtime applications must support the shared memory paradigm.

117112

117113

117114

117115

117116

117117

117118

117119

Additionally, a number of realtime systems provide the ability to map specific sections of the physical address space into the address space of a process. This ability is required if an application is to obtain direct access to memory locations that have specific properties (for example, refresh buffers or display devices, dual ported memory locations, DMA target locations). The use of this ability is common enough to warrant some degree of standardization of its interface. This ability overlaps the general paradigm of shared memory in that, in both instances, common global objects are made addressable by individual processes or tasks.

117120

117121

117122

Finally, a number of systems also provide the ability to map process addresses to files. This provides both a general means of sharing persistent objects, and using files in a manner that optimizes memory and swapping space usage.

117123

117124

117125

117126

117127

117128

Simple shared memory is clearly a special case of the more general file mapping capability. In addition, there is relatively widespread agreement and implementation of the file mapping interface. In these systems, many different types of objects can be mapped (for example, files, memory, devices, and so on) using the same mapping interfaces. This approach both minimizes interface proliferation and maximizes the generality of programs using the mapping interfaces.

117129

- Memory Mapped Files Usage

117130

117131

117132

117133

117134

117135

117136

117137

117138

A memory object can be concurrently mapped into the address space of one or more processes. The *mmap()* and *munmap()* functions allow a process to manipulate their address space by mapping portions of memory objects into it and removing them from it. When multiple processes map the same memory object, they can share access to the underlying data. Implementations may restrict the size and alignment of mappings to be on *page-size* boundaries. The page size, in bytes, is the value of the system-configurable variable {PAGESIZE}, typically accessed by calling *sysconf()* with a *name* argument of *\_SC\_PAGESIZE*. If an implementation has no restrictions on size or alignment, it may specify a 1-byte page size.

117139

117140

117141

To map memory, a process first opens a memory object. The *ftruncate()* function can be used to contract or extend the size of the memory object even when the object is currently mapped. If the memory object is extended, the contents of the extended areas are zeros.

117142 After opening a memory object, the application maps the object into its address space  
 117143 using the *mmap()* function call. Once a mapping has been established, it remains mapped  
 117144 until unmapped with *munmap()*, even if the memory object is closed. The *mprotect()*  
 117145 function can be used to change the memory protections initially established by *mmap()*.

117146 A *close()* of the file descriptor, while invalidating the file descriptor itself, does not unmap  
 117147 any mappings established for the memory object. The address space, including all mapped  
 117148 regions, is inherited on *fork()*. The entire address space is unmapped on process  
 117149 termination or by successful calls to any of the *exec* family of functions.

117150 The *msync()* function is used to force mapped file data to permanent storage.

117151 • Effects on Other Functions

117152 With memory mapped files, the operation of the *open()*, *creat()*, and *unlink()* functions are  
 117153 a natural result of using the file system name space to map the global names for memory  
 117154 objects.

117155 The *ftruncate()* function can be used to set the length of a sharable memory object.

117156 The meaning of *stat()* fields other than the size and protection information is undefined on  
 117157 implementations where memory objects are not implemented using regular files. When  
 117158 regular files are used, the times reflect when the implementation updated the file image of  
 117159 the data, not when a process updated the data in memory.

117160 The operations of *fdopen()*, *write()*, *read()*, and *lseek()* were made unspecified for objects  
 117161 opened with *shm\_open()*, so that implementations that did not implement memory objects  
 117162 as regular files would not have to support the operation of these functions on shared  
 117163 memory objects.

117164 The behavior of memory objects with respect to *close()*, *dup()*, *dup2()*, *open()*, *close()*,  
 117165 *fork()*, *\_exit()*, and the *exec* family of functions is the same as the behavior of the existing  
 117166 practice of the *mmap()* function.

117167 A memory object can still be referenced after a close. That is, any mappings made to the  
 117168 file are still in effect, and reads and writes that are made to those mappings are still valid  
 117169 and are shared with other processes that have the same mapping. Likewise, the memory  
 117170 object can still be used if any references remain after its name(s) have been deleted. Any  
 117171 references that remain after a close must not appear to the application as file descriptors.

117172 This is existing practice for *mmap()* and *close()*. In addition, there are already mappings  
 117173 present (text, data, stack) that do not have open file descriptors. The text mapping in  
 117174 particular is considered a reference to the file containing the text. The desire was to treat all  
 117175 mappings by the process uniformly. Also, many modern implementations use *mmap()* to  
 117176 implement shared libraries, and it would not be desirable to keep file descriptors for each  
 117177 of the many libraries an application can use. It was felt there were many other existing  
 117178 programs that used this behavior to free a file descriptor, and thus POSIX.1-200x could not  
 117179 forbid it and still claim to be using existing practice.

117180 For implementations that implement memory objects using memory only, memory objects  
 117181 will retain the memory allocated to the file after the last close and will use that same  
 117182 memory on the next open. Note that closing the memory object is not the same as deleting  
 117183 the name, since the memory object is still defined in the memory object name space.

117184 The locks of *fcntl()* do not block any read or write operation, including read or write access  
 117185 to shared memory or mapped files. In addition, implementations that only support shared  
 117186 memory objects should not be required to implement record locks. The reference to *fcntl()*  
 117187 is added to make this point explicitly. The other *fcntl()* commands are useful with shared  
 117188 memory objects.



117189 The size of pages that mapping hardware may be able to support may be a configurable  
 117190 value, or it may change based on hardware implementations. The addition of the  
 117191 `_SC_PAGESIZE` parameter to the `sysconf()` function is provided for determining the  
 117192 mapping page size at runtime.

### 117193 Shared Memory Functions

117194 Implementations may support the Shared Memory Objects option independently of memory  
 117195 mapped files. Shared memory objects are named regions of storage that may be independent of  
 117196 the file system and can be mapped into the address space of one or more processes to allow  
 117197 them to share the associated memory.

- 117198 • Requirements

117199 Shared memory is used to share data among several processes, each potentially running at  
 117200 different priority levels, responding to different inputs, or performing separate tasks.  
 117201 Shared memory is not just simply providing common access to data, it is providing the  
 117202 fastest possible communication between the processes. With one memory write operation,  
 117203 a process can pass information to as many processes as have the memory region mapped.

117204 As a result, shared memory provides a mechanism that can be used for all other  
 117205 interprocess communication facilities. It may also be used by an application for  
 117206 implementing more sophisticated mechanisms than semaphores and message queues.

117207 The need for a shared memory interface is obvious for virtual memory systems, where the  
 117208 operating system is directly preventing processes from accessing each other's data.  
 117209 However, in unprotected systems, such as those found in some embedded controllers, a  
 117210 shared memory interface is needed to provide a portable mechanism to allocate a region of  
 117211 memory to be shared and then to communicate the address of that region to other  
 117212 processes.

117213 This, then, provides the minimum functionality that a shared memory interface must have  
 117214 in order to support realtime applications: to allocate and name an object to be mapped into  
 117215 memory for potential sharing (`open()` or `shm_open()`), and to make the memory object  
 117216 available within the address space of a process (`mmap()`). To complete the interface, a  
 117217 mechanism to release the claim of a process on a shared memory object (`munmap()`) is also  
 117218 needed, as well as a mechanism for deleting the name of a sharable object that was  
 117219 previously created (`unlink()` or `shm_unlink()`).

117220 After a mapping has been established, an implementation should not have to provide  
 117221 services to maintain that mapping. All memory writes into that area will appear  
 117222 immediately in the memory mapping of that region by any other processes.

117223 Thus, requirements include:

- 117224 — Support creation of sharable memory objects and the mapping of these objects into  
 117225 the address space of a process.
- 117226 — Sharable memory objects should be accessed by global names accessible from all  
 117227 processes.
- 117228 — Support the mapping of specific sections of physical address space (such as a  
 117229 memory mapped device) into the address space of a process. This should not be  
 117230 done by the process specifying the actual address, but again by an implementation-  
 117231 defined global name (such as a special device name) dedicated to this purpose.
- 117232 — Support the mapping of discrete portions of these memory objects.

- 117233 — Support for minimum hardware configurations that contain no physical media on
- 117234 which to store shared memory contents permanently.
- 117235 — The ability to preallocate the entire shared memory region so that minimum
- 117236 hardware configurations without virtual memory support can guarantee contiguous
- 117237 space.
- 117238 — The maximizing of performance by not requiring functionality that would require
- 117239 implementation interaction above creating the shared memory area and returning
- 117240 the mapping.

117241 Note that the above requirements do not preclude:

- 117242 — The sharable memory object from being implemented using actual files on an actual
- 117243 file system.
- 117244 — The global name that is accessible from all processes being restricted to a file system
- 117245 area that is dedicated to handling shared memory.
- 117246 — An implementation not providing implementation-defined global names for the
- 117247 purpose of physical address mapping.

#### 117248 • Shared Memory Objects Usage

117249 If the Shared Memory Objects option is supported, a shared memory object may be  
 117250 created, or opened if it already exists, with the *shm\_open()* function. If the shared memory  
 117251 object is created, it has a length of zero. The *ftruncate()* function can be used to set the size  
 117252 of the shared memory object after creation. The *shm\_unlink()* function removes the name  
 117253 for a shared memory object created by *shm\_open()*.

#### 117254 • Shared Memory Overview

117255 The shared memory facility defined by POSIX.1-200x usually results in memory locations  
 117256 being added to the address space of the process. The implementation returns the address  
 117257 of the new space to the application by means of a pointer. This works well in languages  
 117258 like C. However, in languages without pointer types it will not work. In the bindings for  
 117259 such a language, either a special COMMON section will need to be defined (which is  
 117260 unlikely), or the binding will have to allow existing structures to be mapped. The  
 117261 implementation will likely have to place restrictions on the size and alignment of such  
 117262 structures or will have to map a suitable region of the address space of the process into the  
 117263 memory object, and thus into other processes. These are issues for that particular language  
 117264 binding. For POSIX.1-200x, however, the practice will not be forbidden, merely undefined.

117265 Two potentially different name spaces are used for naming objects that may be mapped  
 117266 into process address spaces. When using memory mapped files, files may be accessed via  
 117267 *open()*. When the Shared Memory Objects option is supported, sharable memory objects  
 117268 that might not be files may be accessed via the *shm\_open()* function. These operations are  
 117269 not mutually-exclusive.

117270 Some implementations supporting the Shared Memory Objects option may choose to  
 117271 implement the shared memory object name space as part of the file system name space.  
 117272 There are several reasons for this:

- 117273 — It allows applications to prevent name conflicts by use of the directory structure.
- 117274 — It uses an existing mechanism for accessing global objects and prevents the creation
- 117275 of a new mechanism for naming global objects.

117276 In such implementations, memory objects can be implemented using regular files, if that is  
 117277 what the implementation chooses. The *shm\_open()* function can be implemented as an

117278 *open()* call in a fixed directory followed by a call to *fcntl()* to set *FD\_CLOEXEC*. The  
117279 *shm\_unlink()* function can be implemented as an *unlink()* call.

117280 On the other hand, it is also expected that small embedded systems that support the  
117281 Shared Memory Objects option may wish to implement shared memory without having  
117282 any file systems present. In this case, the implementations may choose to use a simple  
117283 string valued name space for shared memory regions. The *shm\_open()* function permits  
117284 either type of implementation.

117285 Some implementations have hardware that supports protection of mapped data from  
117286 certain classes of access and some do not. Systems that supply this functionality support  
117287 the memory protection functionality.

117288 Some implementations restrict size, alignment, and protections to be on *page-size*  
117289 boundaries. If an implementation has no restrictions on size or alignment, it may specify a  
117290 1-byte page size. Applications on implementations that do support larger pages must be  
117291 cognizant of the page size since this is the alignment and protection boundary.

117292 Simple embedded implementations may have a 1-byte page size and only support the  
117293 Shared Memory Objects option. This provides simple shared memory between processes  
117294 without requiring mapping hardware.

117295 POSIX.1-200x specifically allows a memory object to remain referenced after a close  
117296 because that is existing practice for the *mmap()* function.

### 117297 **Typed Memory Functions**

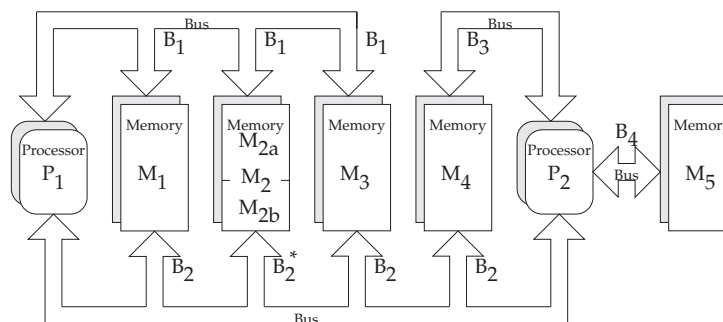
117298 Implementations may support the Typed Memory Objects option without supporting either the  
117299 Shared Memory option or memory mapped files. Types memory objects are pools of specialized  
117300 storage, different from the main memory resource normally used by a processor to hold code  
117301 and data, that can be mapped into the address space of one or more processes.

- 117302 • **Model**

117303 Realtime systems conforming to one of the POSIX.13 realtime profiles are expected (and  
117304 desired) to be supported on systems with more than one type or pool of memory (for  
117305 example, SRAM, DRAM, ROM, EPROM, EEPROM), where each type or pool of memory  
117306 may be accessible by one or more processors via one or more buses (ports). Memory  
117307 mapped files, shared memory objects, and the language-specific storage allocation  
117308 operators (*malloc()* for the ISO C standard, *new* for ISO Ada) fail to provide application  
117309 program interfaces versatile enough to allow applications to control their utilization of  
117310 such diverse memory resources. The typed memory interfaces *posix\_typed\_mem\_open()*,  
117311 *posix\_mem\_offset()*, *posix\_typed\_mem\_get\_info()*, *mmap()*, and *munmap()* defined herein  
117312 support the model of typed memory described below.

117313 For purposes of this model, a system comprises several processors (for example,  $P_1$  and  
117314  $P_2$ ), several physical memory pools (for example,  $M_1$ ,  $M_2$ ,  $M_{2a}$ ,  $M_{2b}$ ,  $M_3$ ,  $M_4$ , and  $M_5$ ), and  
117315 several buses or “ports” (for example,  $B_1$ ,  $B_2$ ,  $B_3$ , and  $B_4$ ) interconnecting the various  
117316 processors and memory pools in some system-specific way. Notice that some memory  
117317 pools may be contained in others (for example,  $M_{2a}$  and  $M_{2b}$  are contained in  $M_2$ ).

117318 **Figure B-1** (on page 3436) shows an example of such a model. In a system like this, an  
117319 application should be able to perform the following operations:



\* All addresses in pool M<sub>2</sub> (comprising pools M<sub>2a</sub> and M<sub>2b</sub>) accessible via port B<sub>1</sub>.  
 Addresses in pool M<sub>2b</sub> are also accessible via port B<sub>2</sub>.  
 Addresses in pool M<sub>2a</sub> are *not* accessible via port B<sub>2</sub>.

117320

**Figure B-1** Example of a System with Typed Memory

117321

— Typed Memory Allocation

117322

117323

117324

117325

117326

117327

An application should be able to allocate memory dynamically from the desired pool using the desired bus, and map it into the address space of a process. For example, processor P<sub>1</sub> can allocate some portion of memory pool M<sub>1</sub> through port B<sub>1</sub>, treating all unmapped subareas of M<sub>1</sub> as a heap-storage resource from which memory may be allocated. This portion of memory is mapped into address space of the process, and subsequently deallocated when unmapped from all processes.

117328

— Using the Same Storage Region from Different Busses

117329

117330

117331

117332

117333

An application process with a mapped region of storage that is accessed from one bus should be able to map that same storage area at another address (subject to page size restrictions detailed in *mmap()*), to allow it to be accessed from another bus. For example, processor P<sub>1</sub> may wish to access the same region of memory pool M<sub>2b</sub> both through ports B<sub>1</sub> and B<sub>2</sub>.

117334

— Sharing Typed Memory Regions

117335

117336

117337

117338

117339

117340

117341

117342

117343

117344

117345

117346

117347

Several application processes running on the same or different processors may wish to share a particular region of a typed memory pool. Each process or processor may wish to access this region through different buses. For example, processor P<sub>1</sub> may want to share a region of memory pool M<sub>4</sub> with processor P<sub>2</sub>, and they may be required to use buses B<sub>2</sub> and B<sub>3</sub>, respectively, to minimize bus contention. A problem arises here when a process allocates and maps a portion of fragmented memory and then wants to share this region of memory with another process, either in the same processor or different processors. The solution adopted is to allow the first process to find out the memory map (offsets and lengths) of all the different fragments of memory that were mapped into its address space, by repeatedly calling *posix\_mem\_offset()*. Then, this process can pass the offsets and lengths obtained to the second process, which can then map the same memory fragments into its address space.

117348

— Contiguous Allocation

117349

117350

The problem of finding the memory map of the different fragments of the memory pool that were mapped into logically contiguous addresses of a given process can be

117351 solved by requesting contiguous allocation. For example, a process in  $P_1$  can allocate  
 117352 10 Kbytes of physically contiguous memory from  $M_3-B_1$ , and obtain the offset (within  
 117353 pool  $M_3$ ) of this block of memory. Then, it can pass this offset (and the length) to a  
 117354 process in  $P_2$  using some interprocess communication mechanism. The second  
 117355 process can map the same block of memory by using the offset transferred and  
 117356 specifying  $M_3-B_2$ .

117357 — Unallocated Mapping

117358 Any subarea of a memory pool that is mapped to a process, either as the result of an  
 117359 allocation request or an explicit mapping, is normally unavailable for allocation.  
 117360 Special processes such as debuggers, however, may need to map large areas of a  
 117361 typed memory pool, yet leave those areas available for allocation.

117362 Typed memory allocation and mapping has to coexist with storage allocation operators  
 117363 like *malloc()*, but systems are free to choose how to implement this coexistence. For  
 117364 example, it may be system configuration-dependent if all available system memory is  
 117365 made part of one of the typed memory pools or if some part will be restricted to  
 117366 conventional allocation operators. Equally system configuration-dependent may be the  
 117367 availability of operators like *malloc()* to allocate storage from certain typed memory pools.  
 117368 It is not excluded to configure a system such that a given named pool,  $P_1$ , is in turn split  
 117369 into non-overlapping named subpools. For example,  $M_1-B_1$ ,  $M_2-B_1$ , and  $M_3-B_1$  could also be  
 117370 accessed as one common pool  $M_{123}-B_1$ . A call to *malloc()* on  $P_1$  could work on such a larger  
 117371 pool while full optimization of memory usage by  $P_1$  would require typed memory  
 117372 allocation at the subpool level.

117373 • Existing Practice

117374 OS-9 provides for the naming (numbering) and prioritization of memory types by a system  
 117375 administrator. It then provides APIs to request memory allocation of typed (colored)  
 117376 memory by number, and to generate a bus address from a mapped memory address  
 117377 (translate). When requesting colored memory, the user can specify type 0 to signify  
 117378 allocation from the first available type in priority order.

117379 HP-RT presents interfaces to map different kinds of storage regions that are visible through  
 117380 a VME bus, although it does not provide allocation operations. It also provides functions  
 117381 to perform address translation between VME addresses and virtual addresses. It represents  
 117382 a VME-bus unique solution to the general problem.

117383 The PSOS approach is similar (that is, based on a pre-established mapping of bus address  
 117384 ranges to specific memories) with a concept of segments and regions (regions dynamically  
 117385 allocated from a heap which is a special segment). Therefore, PSOS does not fully address  
 117386 the general allocation problem either. PSOS does not have a “process”-based model, but  
 117387 more of a “thread”-only-based model of multi-tasking. So mapping to a process address  
 117388 space is not an issue.

117389 QNX uses the System V approach of opening specially named devices (shared memory  
 117390 segments) and using *mmap()* to then gain access from the process. They do not address  
 117391 allocation directly, but once typed shared memory can be mapped, an “allocation  
 117392 manager” process could be written to handle requests for allocation.

117393 The System V approach also included allocation, implemented by opening yet other  
 117394 special “devices” which allocate, rather than appearing as a whole memory object.

117395 The Orkid realtime kernel interface definition has operations to manage memory “regions”  
 117396 and “pools”, which are areas of memory that may reflect the differing physical nature of  
 117397 the memory. Operations to allocate memory from these regions and pools are also  
 117398 provided.

- 117399 • Requirements
- 117400 Existing practice in SVID-derived UNIX systems relies on functionality similar to *mmap()*  
117401 and its related interfaces to achieve mapping and allocation of typed memory. However,  
117402 the issue of sharing typed memory (allocated or mapped) and the complication of multiple  
117403 ports are not addressed in any consistent way by existing UNIX system practice. Part of  
117404 this functionality is existing practice in specialized realtime operating systems. In order to  
117405 solidify the capabilities implied by the model above, the following requirements are  
117406 imposed on the interface:
- 117407 — Identification of Typed Memory Pools and Ports
- 117408 All processes (running in all processors) in the system are able to identify a particular  
117409 (system configured) typed memory pool accessed through a particular (system  
117410 configured) port by a name. That name is a member of a name space common to all  
117411 these processes, but need not be the same name space as that containing ordinary  
117412 filenames. The association between memory pools/ports and corresponding names  
117413 is typically established when the system is configured. The “open” operation for  
117414 typed memory objects should be distinct from the *open()* function, for consistency  
117415 with other similar services, but implementable on top of *open()*. This implies that the  
117416 handle for a typed memory object will be a file descriptor.
- 117417 — Allocation and Mapping of Typed Memory
- 117418 Once a typed memory object has been identified by a process, it is possible to both  
117419 map user-selected subareas of that object into process address space and to map  
117420 system-selected (that is, dynamically allocated) subareas of that object, with user-  
117421 specified length, into process address space. It is also possible to determine the  
117422 maximum length of memory allocation that may be requested from a given typed  
117423 memory object.
- 117424 — Sharing Typed Memory
- 117425 Two or more processes are able to share portions of typed memory, either user-  
117426 selected or dynamically allocated. This requirement applies also to dynamically  
117427 allocated regions of memory that are composed of several non-contiguous pieces.
- 117428 — Contiguous Allocation
- 117429 For dynamic allocation, it is the user’s option whether the system is required to  
117430 allocate a contiguous subarea within the typed memory object, or whether it is  
117431 permitted to allocate discontinuous fragments which appear contiguous in the  
117432 process mapping. Contiguous allocation simplifies the process of sharing allocated  
117433 typed memory, while discontinuous allocation allows for potentially better recovery  
117434 of deallocated typed memory.
- 117435 — Accessing Typed Memory Through Different Ports
- 117436 Once a subarea of a typed memory object has been mapped, it is possible to  
117437 determine the location and length corresponding to a user-selected portion of that  
117438 object within the memory pool. This location and length can then be used to remap  
117439 that portion of memory for access from another port. If the referenced portion of  
117440 typed memory was allocated discontinuously, the length thus determined may be  
117441 shorter than anticipated, and the user code must adapt to the value returned.
- 117442 — Deallocation
- 117443 When a previously mapped subarea of typed memory is no longer mapped by any  
117444 process in the system—as a result of a call or calls to *munmap()*—that subarea

117445 becomes potentially reusable for dynamic allocation; actual reuse of the subarea is a  
 117446 function of the dynamic typed memory allocation policy.

117447 — Unallocated Mapping

117448 It must be possible to map user-selected subareas of a typed memory object without  
 117449 marking that subarea as unavailable for allocation. This option is not the default  
 117450 behavior, and requires appropriate privilege.

117451 • Scenario

117452 The following scenario will serve to clarify the use of the typed memory interfaces.

117453 Process A running on  $P_1$  (see Figure B-1, on page 3436) wants to allocate some memory  
 117454 from memory pool  $M_2$ , and it wants to share this portion of memory with process B  
 117455 running on  $P_2$ . Since  $P_2$  only has access to the lower part of  $M_2$ , both processes will use the  
 117456 memory pool named  $M_{2b}$  which is the part of  $M_2$  that is accessible both from  $P_1$  and  $P_2$ . The  
 117457 operations that both processes need to perform are shown below:

117458 — Allocating Typed Memory

117459 Process A calls `posix_typed_mem_open()` with the name `/typed.m2b-b1` and a `tflag` of  
 117460 `POSIX_TYPED_MEM_ALLOCATE` to get a file descriptor usable for allocating from  
 117461 pool  $M_{2b}$  accessed through port  $B_1$ . It then calls `mmap()` with this file descriptor  
 117462 requesting a length of 4096 bytes. The system allocates two discontinuous blocks of  
 117463 sizes 1024 and 3072 bytes within  $M_{2b}$ . The `mmap()` function returns a pointer to a  
 117464 4096-byte array in process A's logical address space, mapping the allocated blocks  
 117465 contiguously. Process A can then utilize the array, and store data in it.

117466 — Determining the Location of the Allocated Blocks

117467 Process A can determine the lengths and offsets (relative to  $M_{2b}$ ) of the two blocks  
 117468 allocated, by using the following procedure: First, process A calls `posix_mem_offset()`  
 117469 with the address of the first element of the array and length 4096. Upon return, the  
 117470 offset and length (1024 bytes) of the first block are returned. A second call to  
 117471 `posix_mem_offset()` is then made using the address of the first element of the array  
 117472 plus 1024 (the length of the first block), and a new length of 4096-1024. If there were  
 117473 more fragments allocated, this procedure could have been continued within a loop  
 117474 until the offsets and lengths of all the blocks were obtained. Notice that this relatively  
 117475 complex procedure can be avoided if contiguous allocation is requested (by opening  
 117476 the typed memory object with the `tflag`  
 117477 `POSIX_TYPED_MEM_ALLOCATE_CONTIG`).

117478 — Sharing Data Across Processes

117479 Process A passes the two offset values and lengths obtained from the  
 117480 `posix_mem_offset()` calls to process B running on  $P_2$ , via some form of interprocess  
 117481 communication. Process B can gain access to process A's data by calling  
 117482 `posix_typed_mem_open()` with the name `/typed.m2b-b2` and a `tflag` of zero, then using  
 117483 two `mmap()` calls on the resulting file descriptor to map the two subareas of that  
 117484 typed memory object to its own address space.

117485 • Rationale for no `mem_alloc()` and `mem_free()`

117486 The standard developers had originally proposed a pair of new flags to `mmap()` which,  
 117487 when applied to a typed memory object descriptor, would cause `mmap()` to allocate  
 117488 dynamically from an unallocated and unmapped area of the typed memory object.  
 117489 Deallocation was similarly accomplished through the use of `munmap()`. This was rejected  
 117490 by the ballot group because it excessively complicated the (already rather complex)

117491 *mmap()* interface and introduced semantics useful only for typed memory, to a function  
 117492 which must also map shared memory and files. They felt that a memory allocator should  
 117493 be built on top of *mmap()* instead of being incorporated within the same interface, much as  
 117494 the ISO C standard libraries build *malloc()* on top of the virtual memory mapping  
 117495 functions *brk()* and *sbrk()*. This would eliminate the complicated semantics involved with  
 117496 unmapping only part of an allocated block of typed memory.

117497 To attempt to achieve ballot group consensus, typed memory allocation and deallocation  
 117498 was first migrated from *mmap()* and *munmap()* to a pair of complementary functions  
 117499 modeled on the ISO C standard *malloc()* and *free()*. The *mem\_alloc()* function specified  
 117500 explicitly the typed memory object (typed memory pool/access port) from which  
 117501 allocation takes place, unlike *malloc()* where the memory pool and port are unspecified.  
 117502 The *mem\_free()* function handled deallocation. These new semantics still met all of the  
 117503 requirements detailed above without modifying the behavior of *mmap()* except to allow it  
 117504 to map specified areas of typed memory objects. An implementation would have been free  
 117505 to implement *mem\_alloc()* and *mem\_free()* over *mmap()*, through *mmap()*, or independently  
 117506 but cooperating with *mmap()*.

117507 The ballot group was queried to see if this was an acceptable alternative, and while there  
 117508 was some agreement that it achieved the goal of removing the complicated semantics of  
 117509 allocation from the *mmap()* interface, several balloters realized that it just created two  
 117510 additional functions that behaved, in great part, like *mmap()*. These balloters proposed an  
 117511 alternative which has been implemented here in place of a separate *mem\_alloc()* and  
 117512 *mem\_free()*. This alternative is based on four specific suggestions:

- 117513 1. The *posix\_typed\_mem\_open()* function should provide a flag which specifies  
 117514 "allocate on *mmap()*" (otherwise, *mmap()* just maps the underlying object). This  
 117515 allows things roughly similar to */dev/zero* versus */dev/swap*. Two such flags have  
 117516 been implemented, one of which forces contiguous allocation.
- 117517 2. The *posix\_mem\_offset()* function is acceptable because it can be applied usefully to  
 117518 mapped objects in general. It should return the file descriptor of the underlying  
 117519 object.
- 117520 3. The *mem\_get\_info()* function in an earlier draft should be renamed  
 117521 *posix\_typed\_mem\_get\_info()* because it is not generally applicable to memory objects.  
 117522 It should probably return the file descriptor's allocation attribute. The renaming of  
 117523 the function has been implemented, but having it return a piece of information  
 117524 which is readily known by an application without this function has been rejected.  
 117525 Its whole purpose is to query the typed memory object for attributes that are not  
 117526 user-specified, but determined by the implementation.
- 117527 4. There should be no separate *mem\_alloc()* or *mem\_free()* functions. Instead, using  
 117528 *mmap()* on a typed memory object opened with an "allocate on *mmap()*" flag  
 117529 should be used to force allocation. These are precisely the semantics defined in the  
 117530 current draft.

117531 • Rationale for no Typed Memory Access Management

117532 The working group had originally defined an additional interface (and an additional kind  
 117533 of object: typed memory master) to establish and dissolve mappings to typed memory on  
 117534 behalf of devices or processors which were independent of the operating system and had  
 117535 no inherent capability to directly establish mappings on their own. This was to have  
 117536 provided functionality similar to device driver interfaces such as *physio()* and their  
 117537 underlying bus-specific interfaces (for example, *mballoc()*) which serve to set up and break  
 117538 down DMA pathways, and derive mapped addresses for use by hardware devices and  
 117539 processor cards.



117540 The ballot group felt that this was beyond the scope of POSIX.1 and its amendments.  
 117541 Furthermore, the removal of interrupt handling interfaces from a preceding amendment  
 117542 (the IEEE Std 1003.1d-1999) during its balloting process renders these typed memory  
 117543 access management interfaces an incomplete solution to portable device management from  
 117544 a user process; it would be possible to initiate a device transfer to/from typed memory, but  
 117545 impossible to handle the transfer-complete interrupt in a portable way.

117546 To achieve ballot group consensus, all references to typed memory access management  
 117547 capabilities were removed. The concept of portable interfaces from a device driver to both  
 117548 operating system and hardware is being addressed by the Uniform Driver Interface (UDI)  
 117549 industry forum, with formal standardization deferred until proof of concept and industry-  
 117550 wide acceptance and implementation.

#### 117551 B.2.8.4 Process Scheduling

117552 IEEE PASC Interpretation 1003.1 #96 has been applied, adding the `pthread_setschedprio()`  
 117553 function. This was added since previously there was no way for a thread to lower its own  
 117554 priority without going to the tail of the threads list for its new priority. This capability is  
 117555 necessary to bound the duration of priority inversion encountered by a thread.

117556 The following portion of the rationale presents models, requirements, and standardization  
 117557 issues relevant to process scheduling; see also [Section B.2.9.4](#) (on page 3480).

117558 In an operating system supporting multiple concurrent processes, the system determines the  
 117559 order in which processes execute to meet implementation-defined goals. For time-sharing  
 117560 systems, the goal is to enhance system throughput and promote fairness; the application is  
 117561 provided with little or no control over this sequencing function. While this is acceptable and  
 117562 desirable behavior in a time-sharing system, it is inappropriate in a realtime system; realtime  
 117563 applications must specifically control the execution sequence of their concurrent processes in  
 117564 order to meet externally defined response requirements.

117565 In POSIX.1-200x, the control over process sequencing is provided using a concept of scheduling  
 117566 policies. These policies, described in detail in this section, define the behavior of the system  
 117567 whenever processor resources are to be allocated to competing processes. Only the behavior of  
 117568 the policy is defined; conforming implementations are free to use any mechanism desired to  
 117569 achieve the described behavior.

- 117570 • Models

117571 In an operating system supporting multiple concurrent processes, the system determines  
 117572 the order in which processes execute and might force long-running processes to yield to  
 117573 other processes at certain intervals. Typically, the scheduling code is executed whenever an  
 117574 event occurs that might alter the process to be executed next.

117575 The simplest scheduling strategy is a “first-in, first-out” (FIFO) dispatcher. Whenever a  
 117576 process becomes runnable, it is placed on the end of a ready list. The process at the front of  
 117577 the ready list is executed until it exits or becomes blocked, at which point it is removed  
 117578 from the list. This scheduling technique is also known as “run-to-completion” or “run-to-  
 117579 block”.

117580 A natural extension to this scheduling technique is the assignment of a “non-migrating  
 117581 priority” to each process. This policy differs from strict FIFO scheduling in only one  
 117582 respect: whenever a process becomes runnable, it is placed at the end of the list of  
 117583 processes runnable at that priority level. When selecting a process to run, the system  
 117584 always selects the first process from the highest priority queue with a runnable process.  
 117585 Thus, when a process becomes unblocked, it will preempt a running process of lower  
 117586 priority without otherwise altering the ready list. Further, if a process elects to alter its

117587 priority, it is removed from the ready list and reinserted, using its new priority, according  
117588 to the policy above.

117589 While the above policy might be considered unfriendly in a time-sharing environment in  
117590 which multiple users require more balanced resource allocation, it could be ideal in a  
117591 realtime environment for several reasons. The most important of these is that it is  
117592 deterministic: the highest-priority process is always run and, among processes of equal  
117593 priority, the process that has been runnable for the longest time is executed first. Because of  
117594 this determinism, cooperating processes can implement more complex scheduling simply  
117595 by altering their priority. For instance, if processes at a single priority were to reschedule  
117596 themselves at fixed time intervals, a time-slice policy would result.

117597 In a dedicated operating system in which all processes are well-behaved realtime  
117598 applications, non-migrating priority scheduling is sufficient. However, many existing  
117599 implementations provide for more complex scheduling policies.

117600 POSIX.1-200x specifies a linear scheduling model. In this model, every process in the  
117601 system has a priority. The system scheduler always dispatches a process that has the  
117602 highest (generally the most time-critical) priority among all runnable processes in the  
117603 system. As long as there is only one such process, the dispatching policy is trivial. When  
117604 multiple processes of equal priority are eligible to run, they are ordered according to a  
117605 strict run-to-completion (FIFO) policy.

117606 The priority is represented as a positive integer and is inherited from the parent process.  
117607 For processes running under a fixed priority scheduling policy, the priority is never altered  
117608 except by an explicit function call.

117609 It was determined arbitrarily that larger integers correspond to “higher priorities”.

117610 Certain implementations might impose restrictions on the priority ranges to which  
117611 processes can be assigned. There also can be restrictions on the set of policies to which  
117612 processes can be set.

117613 • Requirements

117614 Realtime processes require that scheduling be fast and deterministic, and that it guarantees  
117615 to preempt lower priority processes.

117616 Thus, given the linear scheduling model, realtime processes require that they be run at a  
117617 priority that is higher than other processes. Within this framework, realtime processes are  
117618 free to yield execution resources to each other in a completely portable and  
117619 implementation-defined manner.

117620 As there is a generally perceived requirement for processes at the same priority level to  
117621 share processor resources more equitably, provisions are made by providing a scheduling  
117622 policy (that is, SCHED\_RR) intended to provide a timeslice-like facility.

117623 **Note:** The following topics assume that low numeric priority implies low scheduling criticality  
117624 and *vice versa*.

117625 • Rationale for New Interface

117626 Realtime applications need to be able to determine when processes will run in relation to  
117627 each other. It must be possible to guarantee that a critical process will run whenever it is  
117628 runnable; that is, whenever it wants to for as long as it needs. SCHED\_FIFO satisfies this  
117629 requirement. Additionally, SCHED\_RR was defined to meet a realtime requirement for a  
117630 well-defined time-sharing policy for processes at the same priority.

117631 It would be possible to use the BSD *setpriority()* and *getpriority()* functions by redefining  
117632 the meaning of the “nice” parameter according to the scheduling policy currently in use by

117633 the process. The System V *nice()* interface was felt to be undesirable for realtime because it  
 117634 specifies an adjustment to the “nice” value, rather than setting it to an explicit value.  
 117635 Realtime applications will usually want to set priority to an explicit value. Also, System V  
 117636 *nice()* does not allow for changing the priority of another process.

117637 With the POSIX.1b interfaces, the traditional “nice” value does not affect the SCHED\_FIFO  
 117638 or SCHED\_RR scheduling policies. If a “nice” value is supported, it is implementation-  
 117639 defined whether it affects the SCHED\_OTHER policy.

117640 An important aspect of POSIX.1-200x is the explicit description of the queuing and  
 117641 preemption rules. It is critical, to achieve deterministic scheduling, that such rules be  
 117642 stated clearly in POSIX.1-200x.

117643 POSIX.1-200x does not address the interaction between priority and swapping. The issues  
 117644 involved with swapping and virtual memory paging are extremely implementation-  
 117645 defined and would be nearly impossible to standardize at this point. The proposed  
 117646 scheduling paradigm, however, fully describes the scheduling behavior of runnable  
 117647 processes, of which one criterion is that the working set be resident in memory. Assuming  
 117648 the existence of a portable interface for locking portions of a process in memory, paging  
 117649 behavior need not affect the scheduling of realtime processes.

117650 POSIX.1-200x also does not address the priorities of “system” processes. In general, these  
 117651 processes should always execute in low-priority ranges to avoid conflict with other  
 117652 realtime processes. Implementations should document the priority ranges in which system  
 117653 processes run.

117654 The default scheduling policy is not defined. The effect of I/O interrupts and other system  
 117655 processing activities is not defined. The temporary lending of priority from one process to  
 117656 another (such as for the purposes of affecting freeing resources) by the system is not  
 117657 addressed. Preemption of resources is not addressed. Restrictions on the ability of a  
 117658 process to affect other processes beyond a certain level (influence levels) is not addressed.

117659 The rationale used to justify the simple time-quantum scheduler is that it is common  
 117660 practice to depend upon this type of scheduling to ensure “fair” distribution of processor  
 117661 resources among portions of the application that must interoperate in a serial fashion. Note  
 117662 that POSIX.1-200x is silent with respect to the setting of this time quantum, or whether it is  
 117663 a system-wide value or a per-process value, although it appears that the prevailing  
 117664 realtime practice is for it to be a system-wide value.

117665 In a system with  $N$  processes at a given priority, all processor-bound, in which the time  
 117666 quantum is equal for all processes at a specific priority level, the following assumptions  
 117667 are made of such a scheduling policy:

- 117668 1. A time quantum  $Q$  exists and the current process will own control of the processor  
 117669 for at least a duration of  $Q$  and will have the processor for a duration of  $Q$ .
- 117670 2. The  $N$ th process at that priority will control a processor within a duration of  $(N-1)$   
 117671  $\times Q$ .

117672 These assumptions are necessary to provide equal access to the processor and bounded  
 117673 response from the application.

117674 The assumptions hold for the described scheduling policy only if no system overhead,  
 117675 such as interrupt servicing, is present. If the interrupt servicing load is non-zero, then one  
 117676 of the two assumptions becomes fallacious, based upon how  $Q$  is measured by the system.

117677 If  $Q$  is measured by clock time, then the assumption that the process obtains a duration  $Q$   
 117678 processor time is false if interrupt overhead exists. Indeed, a scenario can be constructed  
 117679 with  $N$  processes in which a single process undergoes complete processor starvation if a

117680 peripheral device, such as an analog-to-digital converter, generates significant interrupt  
117681 activity periodically with a period of  $N \times Q$ .

117682 If  $Q$  is measured as actual processor time, then the assumption that the  $N$ th process runs in  
117683 within the duration  $(N-1) \times Q$  is false.

117684 It should be noted that SCHED\_FIFO suffers from interrupt-based delay as well. However,  
117685 for SCHED\_FIFO, the implied response of the system is “as soon as possible”, so that the  
117686 interrupt load for this case is a vendor selection and not a compliance issue.

117687 With this in mind, it is necessary either to complete the definition by including bounds on  
117688 the interrupt load, or to modify the assumptions that can be made about the scheduling  
117689 policy.

117690 Since the motivation of inclusion of the policy is common usage, and since current  
117691 applications do not enjoy the luxury of bounded interrupt load, item (2) above is sufficient  
117692 to express existing application needs and is less restrictive in the standard definition. No  
117693 difference in interface is necessary.

117694 In an implementation in which the time quantum is equal for all processes at a specific  
117695 priority, our assumptions can then be restated as:

- 117696 — A time quantum  $Q$  exists, and a processor-bound process will be rescheduled after a  
117697 duration of, at most,  $Q$ . Time quantum  $Q$  may be defined in either wall clock time or  
117698 execution time.
- 117699 — In general, the  $N$ th process of a priority level should wait no longer than  $(N-1) \times Q$   
117700 time to execute, assuming no processes exist at higher priority levels.
- 117701 — No process should wait indefinitely.

117702 For implementations supporting per-process time quanta, these assumptions can be  
117703 readily extended.

#### 117704 **Sporadic Server Scheduling Policy**

117705 The sporadic server is a mechanism defined for scheduling aperiodic activities in time-critical  
117706 realtime systems. This mechanism reserves a certain bounded amount of execution capacity for  
117707 processing aperiodic events at a high priority level. Any aperiodic events that cannot be  
117708 processed within the bounded amount of execution capacity are executed in the background at a  
117709 low priority level. Thus, a certain amount of execution capacity can be guaranteed to be  
117710 available for processing periodic tasks, even under burst conditions in the arrival of aperiodic  
117711 processing requests (that is, a large number of requests in a short time interval). The sporadic  
117712 server also simplifies the schedulability analysis of the realtime system, because it allows  
117713 aperiodic processes or threads to be treated as if they were periodic. The sporadic server was  
117714 first described by Sprunt, et al.

117715 The key concept of the sporadic server is to provide and limit a certain amount of computation  
117716 capacity for processing aperiodic events at their assigned normal priority, during a time interval  
117717 called the “replenishment period”. Once the entity controlled by the sporadic server mechanism  
117718 is initialized with its period and execution-time budget attributes, it preserves its execution  
117719 capacity until an aperiodic request arrives. The request will be serviced (if there are no higher  
117720 priority activities pending) as long as there is execution capacity left. If the request is completed,  
117721 the actual execution time used to service it is subtracted from the capacity, and a replenishment  
117722 of this amount of execution time is scheduled to happen one replenishment period after the  
117723 arrival of the aperiodic request. If the request is not completed, because there is no execution  
117724 capacity left, then the aperiodic process or thread is assigned a lower background priority. For  
117725 each portion of consumed execution capacity the execution time used is replenished after one

117726 replenishment period. At the time of replenishment, if the sporadic server was executing at a  
 117727 background priority level, its priority is elevated to the normal level. Other similar  
 117728 replenishment policies have been defined, but the one presented here represents a compromise  
 117729 between efficiency and implementation complexity.

117730 The interface that appears in this section defines a new scheduling policy for threads and  
 117731 processes that behaves according to the rules of the sporadic server mechanism. Scheduling  
 117732 attributes are defined and functions are provided to allow the user to set and get the parameters  
 117733 that control the scheduling behavior of this mechanism, namely the normal and low priority, the  
 117734 replenishment period, the maximum number of pending replenishment operations, and the  
 117735 initial execution-time budget.

117736 • Scheduling Aperiodic Activities

117737 Virtually all realtime applications are required to process aperiodic activities. In many  
 117738 cases, there are tight timing constraints that the response to the aperiodic events must  
 117739 meet. Usual timing requirements imposed on the response to these events are:

- 117740 — The effects of an aperiodic activity on the response time of lower priority activities  
 117741 must be controllable and predictable.
- 117742 — The system must provide the fastest possible response time to aperiodic events.
- 117743 — It must be possible to take advantage of all the available processing bandwidth not  
 117744 needed by time-critical activities to enhance average-case response times to aperiodic  
 117745 events.

117746 Traditional methods for scheduling aperiodic activities are background processing, polling  
 117747 tasks, and direct event execution:

- 117748 — Background processing consists of assigning a very low priority to the processing of  
 117749 aperiodic events. It utilizes all the available bandwidth in the system that has not  
 117750 been consumed by higher priority threads. However, it is very difficult, or  
 117751 impossible, to meet requirements on average-case response time, because the  
 117752 aperiodic entity has to wait for the execution of all other entities which have higher  
 117753 priority.
- 117754 — Polling consists of creating a periodic process or thread for servicing aperiodic  
 117755 requests. At regular intervals, the polling entity is started and its services  
 117756 accumulated pending aperiodic requests. If no aperiodic requests are pending, the  
 117757 polling entity suspends itself until its next period. Polling allows the aperiodic  
 117758 requests to be processed at a higher priority level. However, worst and average-case  
 117759 response times of polling entities are a direct function of the polling period, and there  
 117760 is execution overhead for each polling period, even if no event has arrived. If the  
 117761 deadline of the aperiodic activity is short compared to the inter-arrival time, the  
 117762 polling frequency must be increased to guarantee meeting the deadline. For this case,  
 117763 the increase in frequency can dramatically reduce the efficiency of the system and,  
 117764 therefore, its capacity to meet all deadlines. Yet, polling represents a good way to  
 117765 handle a large class of practical problems because it preserves system predictability,  
 117766 and because the amortized overhead drops as load increases.
- 117767 — Direct event execution consists of executing the aperiodic events at a high fixed-  
 117768 priority level. Typically, the aperiodic event is processed by an interrupt service  
 117769 routine as soon as it arrives. This technique provides predictable response times for  
 117770 aperiodic events, but makes the response times of all lower priority activities  
 117771 completely unpredictable under burst arrival conditions. Therefore, if the density of  
 117772 aperiodic event arrivals is unbounded, it may be a dangerous technique for time-  
 117773 critical systems. Yet, for those cases in which the physics of the system imposes a

- 117774 bound on the event arrival rate, it is probably the most efficient technique.
- 117775 — The sporadic server scheduling algorithm combines the predictability of the polling  
117776 approach with the short response times of the direct event execution. Thus, it allows  
117777 systems to meet an important class of application requirements that cannot be met by  
117778 using the traditional approaches. Multiple sporadic servers with different attributes  
117779 can be applied to the scheduling of multiple classes of aperiodic events, each with  
117780 different kinds of timing requirements, such as individual deadlines, average  
117781 response times, and so on. It also has many other interesting applications for  
117782 realtime, such as scheduling producer/consumer tasks in time-critical systems,  
117783 limiting the effects of faults on the estimation of task execution-time requirements,  
117784 and so on.
- 117785 • Existing Practice
 

117786 The sporadic server has been used in different kinds of applications, including military  
117787 avionics, robot control systems, industrial automation systems, and so on. There are  
117788 examples of many systems that cannot be successfully scheduled using the classic  
117789 approaches, such as direct event execution, or polling, and are schedulable using a  
117790 sporadic server scheduler. The sporadic server algorithm itself can successfully schedule  
117791 all systems scheduled with direct event execution or polling.

117792 The sporadic server scheduling policy has been implemented as a commercial product in  
117793 the run-time system of the Verdex Ada compiler. There are also many applications that  
117794 have used a much less efficient application-level sporadic server. These realtime  
117795 applications would benefit from a sporadic server scheduler implemented at the scheduler  
117796 level.
  - 117797 • Library-Level *versus* Kernel-Level Implementation
 

117798 The sporadic server interface described in this section requires the sporadic server policy  
117799 to be implemented at the same level as the scheduler. This means that the process sporadic  
117800 server must be implemented at the kernel level and the thread sporadic server policy  
117801 implemented at the same level as the thread scheduler; that is, kernel or library level.

117802 In an earlier interface for the sporadic server, this mechanism was implementable at a  
117803 different level than the scheduler. This feature allowed the implementor to choose between  
117804 an efficient scheduler-level implementation, or a simpler user or library-level  
117805 implementation. However, the working group considered that this interface made the use  
117806 of sporadic servers more complex, and that library-level implementations would lack some  
117807 of the important functionality of the sporadic server, namely the limitation of the actual  
117808 execution time of aperiodic activities. The working group also felt that the interface  
117809 described in this chapter does not preclude library-level implementations of threads  
117810 intended to provide efficient low-overhead scheduling for those threads that are not  
117811 scheduled under the sporadic server policy.
  - 117812 • Range of Scheduling Priorities
 

117813 Each of the scheduling policies supported in POSIX.1-200x has an associated range of  
117814 priorities. The priority ranges for each policy might or might not overlap with the priority  
117815 ranges of other policies. For time-critical realtime applications it is usual for periodic and  
117816 aperiodic activities to be scheduled together in the same processor. Periodic activities will  
117817 usually be scheduled using the SCHED\_FIFO scheduling policy, while aperiodic activities  
117818 may be scheduled using SCHED\_SPORADIC. Since the application developer will require  
117819 complete control over the relative priorities of these activities in order to meet his timing  
117820 requirements, it would be desirable for the priority ranges of SCHED\_FIFO and  
117821 SCHED\_SPORADIC to overlap completely. Therefore, although POSIX.1-200x does not

117822 require any particular relationship between the different priority ranges, it is  
117823 recommended that these two ranges should coincide.

117824 • Dynamically Setting the Sporadic Server Policy

117825 Several members of the working group requested that implementations should not be  
117826 required to support dynamically setting the sporadic server scheduling policy for a thread.  
117827 The reason is that this policy may have a high overhead for library-level implementations  
117828 of threads, and if threads are allowed to dynamically set this policy, this overhead can be  
117829 experienced even if the thread does not use that policy. By disallowing the dynamic setting  
117830 of the sporadic server scheduling policy, these implementations can accomplish efficient  
117831 scheduling for threads using other policies. If a strictly conforming application needs to  
117832 use the sporadic server policy, and is therefore willing to pay the overhead, it must set this  
117833 policy at the time of thread creation.

117834 • Limitation of the Number of Pending Replenishments

117835 The number of simultaneously pending replenishment operations must be limited for each  
117836 sporadic server for two reasons: an unlimited number of replenishment operations would  
117837 need an unlimited number of system resources to store all the pending replenishment  
117838 operations; on the other hand, in some implementations each replenishment operation will  
117839 represent a source of priority inversion (just for the duration of the replenishment  
117840 operation) and thus, the maximum amount of replenishments must be bounded to  
117841 guarantee bounded response times. The way in which the number of replenishments is  
117842 bounded is by lowering the priority of the sporadic server to *sched\_ss\_low\_priority* when  
117843 the number of pending replenishments has reached its limit. In this way, no new  
117844 replenishments are scheduled until the number of pending replenishments decreases.

117845 In the sporadic server scheduling policy defined in POSIX.1-200x, the application can  
117846 specify the maximum number of pending replenishment operations for a single sporadic  
117847 server, by setting the value of the *sched\_ss\_max\_repl* scheduling parameter. This value must  
117848 be between one and `{SS_REPL_MAX}`, which is a maximum limit imposed by the  
117849 implementation. The limit `{SS_REPL_MAX}` must be greater than or equal to  
117850 `{_POSIX_SS_REPL_MAX}`, which is defined to be four in POSIX.1-200x. The minimum  
117851 limit of four was chosen so that an application can at least guarantee that four different  
117852 aperiodic events can be processed during each interval of length equal to the  
117853 replenishment period.

117854 B.2.8.5 Clocks and Timers

117855 • Clocks

117856 POSIX.1-200x and the ISO C standard both define functions for obtaining system time.  
117857 Implicit behind these functions is a mechanism for measuring passage of time. This  
117858 specification makes this mechanism explicit and calls it a clock. The `CLOCK_REALTIME`  
117859 clock required by POSIX.1-200x is a higher resolution version of the clock that maintains  
117860 POSIX.1 system time. This is a “system-wide” clock, in that it is visible to all processes  
117861 and, were it possible for multiple processes to all read the clock at the same time, they  
117862 would see the same value.

117863 An extensible interface was defined, with the ability for implementations to define  
117864 additional clocks. This was done because of the observation that many realtime platforms  
117865 support multiple clocks, and it was desired to fit this model within the standard interface.  
117866 But implementation-defined clocks need not represent actual hardware devices, nor are  
117867 they necessarily system-wide.

117868  
117869  
117870  
117871  
117872  
117873  
117874  
117875  
117876  
117877  
117878  
117879  
117880  
117881  
117882  
117883  
117884  
117885  
117886  
117887  
117888  
117889  
117890  
117891  
117892  
117893  
117894  
117895  
117896  
117897  
117898  
117899  
117900  
117901  
117902  
117903  
117904  
117905  
117906  
117907  
117908  
117909  
117910

- Timers

Two timer types are required for a system to support realtime applications:

1. One-shot

A one-shot timer is a timer that is armed with an initial expiration time, either relative to the current time or at an absolute time (based on some timing base, such as time in seconds and nanoseconds since the Epoch). The timer expires once and then is disarmed. With the specified facilities, this is accomplished by setting the *it\_value* member of the *value* argument to the desired expiration time and the *it\_interval* member to zero.

2. Periodic

A periodic timer is a timer that is armed with an initial expiration time, again either relative or absolute, and a repetition interval. When the initial expiration occurs, the timer is reloaded with the repetition interval and continues counting. With the specified facilities, this is accomplished by setting the *it\_value* member of the *value* argument to the desired initial expiration time and the *it\_interval* member to the desired repetition interval.

For both of these types of timers, the time of the initial timer expiration can be specified in two ways:

1. Relative (to the current time)
2. Absolute

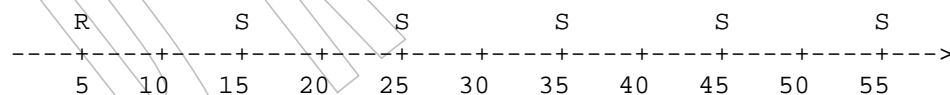
- Examples of Using Realtime Timers

In the diagrams below, *S* indicates a program schedule, *R* shows a schedule method request, and *E* suggests an internal operating system event.

- Periodic Timer: Data Logging

During an experiment, it might be necessary to log realtime data periodically to an internal buffer or to a mass storage device. With a periodic scheduling method, a logging module can be started automatically at fixed time intervals to log the data.

Program schedule is requested every 10 seconds.



[Time (in Seconds)]

To achieve this type of scheduling using the specified facilities, one would allocate a per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an initial expiration value and a repetition interval of 10 seconds.

- One-shot Timer (Relative Time): Device Initialization

In an emission test environment, large sample bags are used to capture the exhaust from a vehicle. The exhaust is purged from these bags before each and every test. With a one-shot timer, a module could initiate the purge function and then suspend itself for a predetermined period of time while the sample bags are prepared.

Program schedule requested 20 seconds after call is issued.





117911 -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----->  
 117912 5 10 15 20 25 30 35 40 45 50 55

117913 [Time (in Seconds)]

117914 To achieve this type of scheduling using the specified facilities, one would allocate a  
 117915 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 117916 armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag reset, and with an  
 117917 initial expiration value of 20 seconds and a repetition interval of zero.

117918 Note that if the program wishes merely to suspend itself for the specified interval, it  
 117919 could more easily use `nanosleep()`.

117920 — One-shot Timer (Absolute Time): Data Transmission

117921 The results from an experiment are often moved to a different system within a  
 117922 network for postprocessing or archiving. With an absolute one-shot timer, a module  
 117923 that moves data from a test-cell computer to a host computer can be automatically  
 117924 scheduled on a daily basis.

117925 Program schedule requested for 2:30 a.m.

117926 R S  
 117927 -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----->  
 117928 23:00 23:30 24:00 00:30 01:00 01:30 02:00 02:30 03:00

117929 [Time of Day]

117930 To achieve this type of scheduling using the specified facilities, a per-process timer  
 117931 would be allocated based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 117932 armed via a call to `timer_settime()` with the `TIMER_ABSTIME` flag set, and an initial  
 117933 expiration value equal to 2:30 a.m. of the next day.

117934 — Periodic Timer (Relative Time): Signal Stabilization

117935 Some measurement devices, such as emission analyzers, do not respond  
 117936 instantaneously to an introduced sample. With a periodic timer with a relative initial  
 117937 expiration time, a module that introduces a sample and records the average response  
 117938 could suspend itself for a predetermined period of time while the signal is stabilized  
 117939 and then sample at a fixed rate.

117940 Program schedule requested 15 seconds after call is issued and every 2 seconds  
 117941 thereafter.

117942 R S S S S S S S S S S S S S S S S S S S S S S S S S  
 117943 -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----->  
 117944 5 10 15 20 25 30 35 40 45 50 55

117945 [Time (in Seconds)]

117946 To achieve this type of scheduling using the specified facilities, one would allocate a  
 117947 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 117948 armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag reset, and with an  
 117949 initial expiration value of 15 seconds and a repetition interval of 2 seconds.

117950 — Periodic Timer (Absolute Time): Work Shift-related Processing

117951 Resource utilization data is useful when time to perform experiments is being  
 117952 scheduled at a facility. With a periodic timer with an absolute initial expiration time,  
 117953 a module can be scheduled at the beginning of a work shift to gather resource  
 117954 utilization data throughout the shift. This data can be used to allocate resources

117955 effectively to minimize bottlenecks and delays and maximize facility throughput.  
 117956 Program schedule requested for 2:00 a.m. and every 15 minutes thereafter.  
 117957 R S S S S S S  
 117958 -----+-----+-----+-----+-----+-----+-----+-----+----->  
 117959 23:00 23:30 24:00 00:30 01:00 01:30 02:00 02:30 03:00

117960 [Time of Day]

117961 To achieve this type of scheduling using the specified facilities, one would allocate a  
 117962 per-process timer based on clock ID `CLOCK_REALTIME`. Then the timer would be  
 117963 armed via a call to `timer_settime()` with `TIMER_ABSTIME` flag set, and with an initial  
 117964 expiration value equal to 2:00 a.m. and a repetition interval equal to 15 minutes.

117965 • Relationship of Timers to Clocks

117966 The relationship between clocks and timers armed with an absolute time is  
 117967 straightforward: a timer expiration signal is requested when the associated clock reaches  
 117968 or exceeds the specified time. The relationship between clocks and timers armed with a  
 117969 relative time (an interval) is less obvious, but not unintuitive. In this case, a timer  
 117970 expiration signal is requested when the specified interval, *as measured by the associated clock*,  
 117971 has passed. For the required `CLOCK_REALTIME` clock, this allows timer expiration  
 117972 signals to be requested at specified “wall clock” times (absolute), or when a specified  
 117973 interval of “realtime” has passed (relative). For an implementation-defined clock—say, a  
 117974 process virtual time clock—timer expirations could be requested when the process has  
 117975 used a specified total amount of virtual time (absolute), or when it has used a specified  
 117976 *additional* amount of virtual time (relative).

117977 The interfaces also allow flexibility in the implementation of the functions. For example, an  
 117978 implementation could convert all absolute times to intervals by subtracting the clock value  
 117979 at the time of the call from the requested expiration time and “counting down” at the  
 117980 supported resolution. Or it could convert all relative times to absolute expiration time by  
 117981 adding in the clock value at the time of the call and comparing the clock value to the  
 117982 expiration time at the supported resolution. Or it might even choose to maintain absolute  
 117983 times as absolute and compare them to the clock value at the supported resolution for  
 117984 absolute timers, and maintain relative times as intervals and count them down at the  
 117985 resolution supported for relative timers. The choice will be driven by efficiency  
 117986 considerations and the underlying hardware or software clock implementation.

117987 • Data Definitions for Clocks and Timers

117988 POSIX.1-200x uses a time representation capable of supporting nanosecond resolution  
 117989 timers for the following reasons:

- 117990 — To enable POSIX.1-200x to represent those computer systems already using  
 117991 nanosecond or submicrosecond resolution clocks.
- 117992 — To accommodate those per-process timers that might need nanoseconds to specify an  
 117993 absolute value of system-wide clocks, even though the resolution of the per-process  
 117994 timer may only be milliseconds, or *vice versa*.
- 117995 — Because the number of nanoseconds in a second can be represented in 32 bits.

117996 Time values are represented in the `timespec` structure. The `tv_sec` member is of type `time_t`  
 117997 so that this member is compatible with time values used by POSIX.1 functions and the  
 117998 ISO C standard. The `tv_nsec` member is a **signed long** in order to simplify and clarify code  
 117999 that decrements or finds differences of time values. Note that because 1 billion (number of  
 118000 nanoseconds per second) is less than half of the value representable by a signed 32-bit

118001 value, it is always possible to add two valid fractional seconds represented as integral  
118002 nanoseconds without overflowing the signed 32-bit value.

118003 A maximum allowable resolution for the CLOCK\_REALTIME clock of 20 ms (1/50  
118004 seconds) was chosen to allow line frequency clocks in European countries to be  
118005 conforming. 60 Hz clocks in the U.S. will also be conforming, as will finer granularity  
118006 clocks, although a Strictly Conforming Application cannot assume a granularity of less  
118007 than 20 ms (1/50 seconds).

118008 The minimum allowable maximum time allowed for the CLOCK\_REALTIME clock and  
118009 the function *nanosleep()*, and timers created with *clock\_id=CLOCK\_REALTIME*, is  
118010 determined by the fact that the *tv\_sec* member is of type **time\_t**.

118011 POSIX.1-200x specifies that timer expirations must not be delivered early, and *nanosleep()*  
118012 must not return early due to quantization error. POSIX.1-200x discusses the various  
118013 implementations of *alarm()* in the rationale and states that implementations that do not  
118014 allow alarm signals to occur early are the most appropriate, but refrained from mandating  
118015 this behavior. Because of the importance of predictability to realtime applications,  
118016 POSIX.1-200x takes a stronger stance.

118017 The standard developers considered using a time representation that differs from  
118018 POSIX.1b in the second 32 bit of the 64-bit value. Whereas POSIX.1b defines this field as a  
118019 fractional second in nanoseconds, the other methodology defines this as a binary fraction  
118020 of one second, with the radix point assumed before the most significant bit.

118021 POSIX.1b is a software, source-level standard and most of the benefits of the alternate  
118022 representation are enjoyed by hardware implementations of clocks and algorithms. It was  
118023 felt that mandating this format for POSIX.1b clocks and timers would unnecessarily  
118024 burden the application developer with writing, possibly non-portable, multiple precision  
118025 arithmetic packages to perform conversion between binary fractions and integral units  
118026 such as nanoseconds, milliseconds, and so on.

### 118027 **Rationale for the Monotonic Clock**

118028 For those applications that use time services to achieve realtime behavior, changing the value of  
118029 the clock on which these services rely may cause erroneous timing behavior. For these  
118030 applications, it is necessary to have a monotonic clock which cannot run backwards, and which  
118031 has a maximum clock jump that is required to be documented by the implementation.  
118032 Additionally, it is desirable (but not required by POSIX.1-200x) that the monotonic clock  
118033 increases its value uniformly. This clock should not be affected by changes to the system time;  
118034 for example, to synchronize the clock with an external source or to account for leap seconds.  
118035 Such changes would cause errors in the measurement of time intervals for those time services  
118036 that use the absolute value of the clock.

118037 One could argue that by defining the behavior of time services when the value of a clock is  
118038 changed, deterministic realtime behavior can be achieved. For example, one could specify that  
118039 relative time services should be unaffected by changes in the value of a clock. However, there are  
118040 time services that are based upon an absolute time, but that are essentially intended as relative  
118041 time services. For example, *pthread\_cond\_timedwait()* uses an absolute time to allow it to wake  
118042 up after the required interval despite spurious wakeups. Although sometimes the  
118043 *pthread\_cond\_timedwait()* timeouts are absolute in nature, there are many occasions in which they  
118044 are relative, and their absolute value is determined from the current time plus a relative time  
118045 interval. In this latter case, if the clock changes while the thread is waiting, the wait interval will  
118046 not be the expected length. If a *pthread\_cond\_timedwait()* function were created that would take a  
118047 relative time, it would not solve the problem because to retain the intended “deadline” a thread  
118048 would need to compensate for latency due to the spurious wakeup, and preemption between

118049 wakeup and the next wait.

118050 The solution is to create a new monotonic clock, whose value does not change except for the  
 118051 regular ticking of the clock, and use this clock for implementing the various relative timeouts  
 118052 that appear in the different POSIX interfaces, as well as allow *pthread\_cond\_timedwait()* to choose  
 118053 this new clock for its timeout. A new *clock\_nanosleep()* function is created to allow an application  
 118054 to take advantage of this newly defined clock. Notice that the monotonic clock may be  
 118055 implemented using the same hardware clock as the system clock.

118056 Relative timeouts for *sigtimedwait()* and *aio\_suspend()* have been redefined to use the monotonic  
 118057 clock, if present. The *alarm()* function has not been redefined, because the same effect but with  
 118058 better resolution can be achieved by creating a timer (for which the appropriate clock may be  
 118059 chosen).

118060 The *pthread\_cond\_timedwait()* function has been treated in a different way, compared to other  
 118061 functions with absolute timeouts, because it is used to wait for an event, and thus it may have a  
 118062 deadline, while the other timeouts are generally used as an error recovery mechanism, and for  
 118063 them the use of the monotonic clock is not so important. Since the desired timeout for the  
 118064 *pthread\_cond\_timedwait()* function may either be a relative interval or an absolute time of day  
 118065 deadline, a new initialization attribute has been created for condition variables to specify the  
 118066 clock that is used for measuring the timeout in a call to *pthread\_cond\_timedwait()*. In this way, if  
 118067 a relative timeout is desired, the monotonic clock will be used; if an absolute deadline is  
 118068 required instead, the *CLOCK\_REALTIME* or another appropriate clock may be used. This  
 118069 capability has not been added to other functions with absolute timeouts because for those  
 118070 functions the expected use of the timeout is mostly to prevent errors, and not so often to meet  
 118071 precise deadlines. As a consequence, the complexity of adding this capability is not justified by  
 118072 its perceived application usage.

118073 The *nanosleep()* function has not been modified with the introduction of the monotonic clock.  
 118074 Instead, a new *clock\_nanosleep()* function has been created, in which the desired clock may be  
 118075 specified in the function call.

118076 • History of Resolution Issues

118077 Due to the shift from relative to absolute timeouts in IEEE Std 1003.1d-1999, the  
 118078 amendments to the *sem\_timedwait()*, *pthread\_mutex\_timedlock()*, *mq\_timedreceive()*, and  
 118079 *mq\_timedsend()* functions of that standard have been removed. Those amendments  
 118080 specified that *CLOCK\_MONOTONIC* would be used for the (relative) timeouts if the  
 118081 Monotonic Clock option was supported.

118082 Having these functions continue to be tied solely to *CLOCK\_MONOTONIC* would not  
 118083 work. Since the absolute value of a time value obtained from *CLOCK\_MONOTONIC* is  
 118084 unspecified, under the absolute timeouts interface, applications would behave differently  
 118085 depending on whether the Monotonic Clock option was supported or not (because the  
 118086 absolute value of the clock would have different meanings in either case).

118087 Two options were considered:

- 118088 1. Leave the current behavior unchanged, which specifies the *CLOCK\_REALTIME*  
 118089 clock for these (absolute) timeouts, to allow portability of applications between  
 118090 implementations supporting or not the Monotonic Clock option.
- 118091 2. Modify these functions in the way that *pthread\_cond\_timedwait()* was modified to  
 118092 allow a choice of clock, so that an application could use *CLOCK\_REALTIME* when  
 118093 it is trying to achieve an absolute timeout and *CLOCK\_MONOTONIC* when it is  
 118094 trying to achieve a relative timeout.

118095 It was decided that the features of *CLOCK\_MONOTONIC* are not as critical to these

118096 functions as they are to `pthread_cond_timedwait()`. The `pthread_cond_timedwait()` function is  
 118097 given a relative timeout; the timeout may represent a deadline for an event. When these  
 118098 functions are given relative timeouts, the timeouts are typically for error recovery  
 118099 purposes and need not be so precise.

118100 Therefore, it was decided that these functions should be tied to `CLOCK_REALTIME` and  
 118101 not complicated by being given a choice of clock.

## 118102 Execution Time Monitoring

- 118103 • Introduction

118104 The main goals of the execution time monitoring facilities defined in this chapter are to  
 118105 measure the execution time of processes and threads and to allow an application to  
 118106 establish CPU time limits for these entities.

118107 The analysis phase of time-critical realtime systems often relies on the measurement of  
 118108 execution times of individual threads or processes to determine whether the timing  
 118109 requirements will be met. Also, performance analysis techniques for soft deadline realtime  
 118110 systems rely heavily on the determination of these execution times. The execution time  
 118111 monitoring functions provide application developers with the ability to measure these  
 118112 execution times online and open the possibility of dynamic execution-time analysis and  
 118113 system reconfiguration, if required.

118114 The second goal of allowing an application to establish execution time limits for individual  
 118115 processes or threads and detecting when they overrun allows program robustness to be  
 118116 increased by enabling online checking of the execution times.

118117 If errors are detected—possibly because of erroneous program constructs, the existence of  
 118118 errors in the analysis phase, or a burst of event arrivals—online detection and recovery is  
 118119 possible in a portable way. This feature can be extremely important for many time-critical  
 118120 applications. Other applications require trapping CPU-time errors as a normal way to exit  
 118121 an algorithm; for instance, some realtime artificial intelligence applications trigger a  
 118122 number of independent inference processes of varying accuracy and speed, limit how long  
 118123 they can run, and pick the best answer available when time runs out. In many periodic  
 118124 systems, overrun processes are simply restarted in the next resource period, after necessary  
 118125 end-of-period actions have been taken. This allows algorithms that are inherently data-  
 118126 dependent to be made predictable.

118127 The interface that appears in this chapter defines a new type of clock, the CPU-time clock,  
 118128 which measures execution time. Each process or thread can invoke the clock and timer  
 118129 functions defined in POSIX.1 to use them. Functions are also provided to access the CPU-  
 118130 time clock of other processes or threads to enable remote monitoring of these clocks.  
 118131 Monitoring of threads of other processes is not supported, since these threads are not  
 118132 visible from outside of their own process with the interfaces defined in POSIX.1.

- 118133 • Execution Time Monitoring Interface

118134 The clock and timer interface defined in POSIX.1 historically only defined one clock, which  
 118135 measures wall-clock time. The requirements for measuring execution time of processes and  
 118136 threads, and setting limits to their execution time by detecting when they overrun, can be  
 118137 accomplished with that interface if a new kind of clock is defined. These new clocks  
 118138 measure execution time, and one is associated with each process and with each thread. The  
 118139 clock functions currently defined in POSIX.1 can be used to read and set these CPU-time  
 118140 clocks, and timers can be created using these clocks as their timing base. These timers can  
 118141 then be used to send a signal when some specified execution time has been exceeded. The  
 118142 CPU-time clocks of each process or thread can be accessed by using the symbols

118143 CLOCK\_PROCESS\_CPUTIME\_ID or CLOCK\_THREAD\_CPUTIME\_ID.

118144 The clock and timer interface defined in POSIX.1 and extended with the new kind of CPU-  
 118145 time clock would only allow processes or threads to access their own CPU-time clocks.  
 118146 However, many realtime systems require the possibility of monitoring the execution time  
 118147 of processes or threads from independent monitoring entities. In order to allow  
 118148 applications to construct independent monitoring entities that do not require cooperation  
 118149 from or modification of the monitored entities, two functions have been added:  
 118150 *clock\_getcpuclockid()*, for accessing CPU-time clocks of other processes, and  
 118151 *pthread\_getcpuclockid()*, for accessing CPU-time clocks of other threads. These functions  
 118152 return the clock identifier associated with the process or thread specified in the call. These  
 118153 clock IDs can then be used in the rest of the clock function calls.

118154 The clocks accessed through these functions could also be used as a timing base for the  
 118155 creation of timers, thereby allowing independent monitoring entities to limit the CPU time  
 118156 consumed by other entities. However, this possibility would imply additional complexity  
 118157 and overhead because of the need to maintain a timer queue for each process or thread, to  
 118158 store the different expiration times associated with timers created by different processes or  
 118159 threads. The working group decided this additional overhead was not justified by  
 118160 application requirements. Therefore, creation of timers attached to the CPU-time clocks of  
 118161 other processes or threads has been specified as implementation-defined.

118162 • Overhead Considerations

118163 The measurement of execution time may introduce additional overhead in the thread  
 118164 scheduling, because of the need to keep track of the time consumed by each of these  
 118165 entities. In library-level implementations of threads, the efficiency of scheduling could be  
 118166 somehow compromised because of the need to make a kernel call, at each context switch,  
 118167 to read the process CPU-time clock. Consequently, a thread creation attribute called *cpu-  
 118168 clock-requirement* was defined, to allow threads to disconnect their respective CPU-time  
 118169 clocks. However, the Ballot Group considered that this attribute itself introduced some  
 118170 overhead, and that in current implementations it was not worth the effort. Therefore, the  
 118171 attribute was deleted, and thus thread CPU-time clocks are required for all threads if the  
 118172 Thread CPU-Time Clocks option is supported.

118173 • Accuracy of CPU-Time Clocks

118174 The mechanism used to measure the execution time of processes and threads is specified in  
 118175 POSIX.1-200x as implementation-defined. The reason for this is that both the underlying  
 118176 hardware and the implementation architecture have a very strong influence on the  
 118177 accuracy achievable for measuring CPU time. For some implementations, the specification  
 118178 of strict accuracy requirements would represent very large overheads, or even the  
 118179 impossibility of being implemented.

118180 Since the mechanism for measuring execution time is implementation-defined, realtime  
 118181 applications will be able to take advantage of accurate implementations using a portable  
 118182 interface. Of course, strictly conforming applications cannot rely on any particular degree  
 118183 of accuracy, in the same way as they cannot rely on a very accurate measurement of wall  
 118184 clock time. There will always exist applications whose accuracy or efficiency requirements  
 118185 on the implementation are more rigid than the values defined in POSIX.1-200x or any  
 118186 other standard.

118187 In any case, there is a minimum set of characteristics that realtime applications would  
 118188 expect from most implementations. One such characteristic is that the sum of all the  
 118189 execution times of all the threads in a process equals the process execution time, when no  
 118190 CPU-time clocks are disabled. This need not always be the case because implementations  
 118191 may differ in how they account for time during context switches. Another characteristic is

118192 that the sum of the execution times of all processes in a system equals the number of  
 118193 processors, multiplied by the elapsed time, assuming that no processor is idle during that  
 118194 elapsed time. However, in some implementations it might not be possible to relate CPU  
 118195 time to elapsed time. For example, in a heterogeneous multi-processor system in which  
 118196 each processor runs at a different speed, an implementation may choose to define each  
 118197 “second” of CPU time to be a certain number of “cycles” that a CPU has executed.

118198 • Existing Practice

118199 Measuring and limiting the execution time of each concurrent activity are common  
 118200 features of most industrial implementations of realtime systems. Almost all critical  
 118201 realtime systems are currently built upon a cyclic executive. With this approach, a regular  
 118202 timer interrupt kicks off the next sequence of computations. It also checks that the current  
 118203 sequence has completed. If it has not, then some error recovery action can be undertaken  
 118204 (or at least an overrun is avoided). Current software engineering principles and the  
 118205 increasing complexity of software are driving application developers to implement these  
 118206 systems on multi-threaded or multi-process operating systems. Therefore, if a POSIX  
 118207 operating system is to be used for this type of application, then it must offer the same level  
 118208 of protection.

118209 Execution time clocks are also common in most UNIX implementations, although these  
 118210 clocks usually have requirements different from those of realtime applications. The  
 118211 POSIX.1 *times()* function supports the measurement of the execution time of the calling  
 118212 process, and its terminated child processes. This execution time is measured in clock ticks  
 118213 and is supplied as two different values with the user and system execution times,  
 118214 respectively. BSD supports the function *getrusage()*, which allows the calling process to get  
 118215 information about the resources used by itself and/or all of its terminated child processes.  
 118216 The resource usage includes user and system CPU time. Some UNIX systems have options  
 118217 to specify high resolution (up to one microsecond) CPU-time clocks using the *times()* or  
 118218 the *getrusage()* functions.

118219 The *times()* and *getrusage()* interfaces do not meet important realtime requirements, such  
 118220 as the possibility of monitoring execution time from a different process or thread, or the  
 118221 possibility of detecting an execution time overrun. The latter requirement is supported in  
 118222 some UNIX implementations that are able to send a signal when the execution time of a  
 118223 process has exceeded some specified value. For example, BSD defines the functions  
 118224 *getitimer()* and *setitimer()*, which can operate either on a realtime clock (wall-clock), or on  
 118225 virtual-time or profile-time clocks which measure CPU time in two different ways. These  
 118226 functions do not support access to the execution time of other processes.

118227 IBM’s MVS operating system supports per-process and per-thread execution time clocks. It  
 118228 also supports limiting the execution time of a given process.

118229 Given all this existing practice, the working group considered that the POSIX.1 clocks and  
 118230 timers interface was appropriate to meet most of the requirements that realtime  
 118231 applications have for execution time clocks. Functions were added to get the CPU time  
 118232 clock IDs, and to allow/disallow the thread CPU-time clocks (in order to preserve the  
 118233 efficiency of some implementations of threads).

118234 • Clock Constants

118235 The definition of the manifest constants `CLOCK_PROCESS_CPUTIME_ID` and  
 118236 `CLOCK_THREAD_CPUTIME_ID` allows processes or threads, respectively, to access their  
 118237 own execution-time clocks. However, given a process or thread, access to its own  
 118238 execution-time clock is also possible if the clock ID of this clock is obtained through a call  
 118239 to *clock\_getcpuclockid()* or *pthread\_getcpuclockid()*. Therefore, these constants are not  
 118240 necessary and could be deleted to make the interface simpler. Their existence saves one

118241 system call in the first access to the CPU-time clock of each process or thread. The working  
 118242 group considered this issue and decided to leave the constants in POSIX.1-200x because  
 118243 they are closer to the POSIX.1b use of clock identifiers.

118244 • Library Implementations of Threads

118245 In library implementations of threads, kernel entities and library threads can coexist. In  
 118246 this case, if the CPU-time clocks are supported, most of the clock and timer functions will  
 118247 need to have two implementations: one in the thread library, and one in the system calls  
 118248 library. The main difference between these two implementations is that the thread library  
 118249 implementation will have to deal with clocks and timers that reside in the thread space,  
 118250 while the kernel implementation will operate on timers and clocks that reside in kernel  
 118251 space. In the library implementation, if the clock ID refers to a clock that resides in the  
 118252 kernel, a kernel call will have to be made. The correct version of the function can be chosen  
 118253 by specifying the appropriate order for the libraries during the link process.

118254 • History of Resolution Issues: Deletion of the *enable* Attribute

118255 In early proposals, consideration was given to inclusion of an attribute called *enable* for  
 118256 CPU-time clocks. This would allow implementations to avoid the overhead of measuring  
 118257 execution time for those processes or threads for which this measurement was not  
 118258 required. However, this is unnecessary since processes are already required to measure  
 118259 execution time by the POSIX.1 *times()* function. Consequently, the *enable* attribute is not  
 118260 present.

118261 **Rationale Relating to Timeouts**

118262 • Requirements for Timeouts

118263 Realtime systems which must operate reliably over extended periods without human  
 118264 intervention are characteristic in embedded applications such as avionics, machine control,  
 118265 and space exploration, as well as more mundane applications such as cable TV, security  
 118266 systems, and plant automation. A multi-tasking paradigm, in which many independent  
 118267 and/or cooperating software functions relinquish the processor(s) while waiting for a  
 118268 specific stimulus, resource, condition, or operation completion, is very useful in producing  
 118269 well engineered programs for such systems. For such systems to be robust and fault-  
 118270 tolerant, expected occurrences that are unduly delayed or that never occur must be  
 118271 detected so that appropriate recovery actions may be taken. This is difficult if there is no  
 118272 way for a task to regain control of a processor once it has relinquished control (blocked)  
 118273 awaiting an occurrence which, perhaps because of corrupted code, hardware malfunction,  
 118274 or latent software bugs, will not happen when expected. Therefore, the common practice  
 118275 in realtime operating systems is to provide a capability to time out such blocking services.  
 118276 Although there are several methods to achieve this already defined by POSIX, none are as  
 118277 reliable or efficient as initiating a timeout simultaneously with initiating a blocking service.  
 118278 This is especially critical in hard-realtime embedded systems because the processors  
 118279 typically have little time reserve, and allowed fault recovery times are measured in  
 118280 milliseconds rather than seconds.

118281 The working group largely agreed that such timeouts were necessary and ought to become  
 118282 part of POSIX.1-200x, particularly vendors of realtime operating systems whose customers  
 118283 had already expressed a strong need for timeouts. There was some resistance to inclusion  
 118284 of timeouts in POSIX.1-200x because the desired effect, fault tolerance, could, in theory, be  
 118285 achieved using existing facilities and alternative software designs, but there was no  
 118286 compelling evidence that realtime system designers would embrace such designs at the  
 118287 sacrifice of performance and/or simplicity.



- 118288 • Which Services should be Timed Out?

118289 Originally, the working group considered the prospect of providing timeouts on all  
 118290 blocking services, including those currently existing in POSIX.1, POSIX.1b, and POSIX.1c,  
 118291 and future interfaces to be defined by other working groups, as sort of a general policy.  
 118292 This was rather quickly rejected because of the scope of such a change, and the fact that  
 118293 many of those services would not normally be used in a realtime context. More traditional  
 118294 timesharing solutions to timeout would suffice for most of the POSIX.1 interfaces, while  
 118295 others had asynchronous alternatives which, while more complex to utilize, would be  
 118296 adequate for some realtime and all non-realtime applications.

118297 The list of potential candidates for timeouts was narrowed to the following for further  
 118298 consideration:

- 118299 — POSIX.1b
- 118300 — *sem\_wait()*
  - 118301 — *mq\_receive()*
  - 118302 — *mq\_send()*
  - 118303 — *lio\_listio()*
  - 118304 — *aio\_suspend()*
  - 118305 — *sigwait()* (timeout already implemented by *sigtimedwait()*)
- 118306 — POSIX.1c
- 118307 — *pthread\_mutex\_lock()*
  - 118308 — *pthread\_join()*
  - 118309 — *pthread\_cond\_wait()*
  - 118310 (timeout already implemented by *pthread\_cond\_timedwait()*)
- 118311 — POSIX.1
- 118312 — *read()*
  - 118313 — *write()*

118314 After further review by the working group, the *lio\_listio()*, *read()*, and *write()* functions (all  
 118315 forms of blocking synchronous I/O) were eliminated from the list because of the  
 118316 following:

- 118317 — Asynchronous alternatives exist
- 118318 — Timeouts can be implemented, albeit non-portably, in device drivers
- 118319 — A strong desire not to introduce modifications to POSIX.1 interfaces

118320 The working group ultimately rejected *pthread\_join()* since both that interface and a timed  
 118321 variant of that interface are non-minimal and may be implemented as a function. See  
 118322 below for a library implementation of *pthread\_join()*.

118323 Thus, there was a consensus among the working group members to add timeouts to 4 of  
 118324 the remaining 5 functions (the timeout for *aio\_suspend()* was ultimately added directly to  
 118325 POSIX.1b, while the others were added by POSIX.1d). However, *pthread\_mutex\_lock()*  
 118326 remained contentious.

118327 Many feel that *pthread\_mutex\_lock()* falls into the same class as the other functions; that is,  
 118328 it is desirable to time out a mutex lock because a mutex may fail to be unlocked due to

errant or corrupted code in a critical section (looping or branching outside of the unlock code), and therefore is equally in need of a reliable, simple, and efficient timeout. In fact, since mutexes are intended to guard small critical sections, most `pthread_mutex_lock()` calls would be expected to obtain the lock without blocking nor utilizing any kernel service, even in implementations of threads with global contention scope; the timeout alternative need only be considered after it is determined that the thread must block.

Those opposed to timing out mutexes feel that the very simplicity of the mutex is compromised by adding a timeout semantic, and that to do so is senseless. They claim that if a timed mutex is really deemed useful by a particular application, then it can be constructed from the facilities already in POSIX.1b and POSIX.1c. The following two C-language library implementations of mutex locking with timeout represent the solutions offered (in both implementations, the timeout parameter is specified as absolute time, not relative time as in the proposed POSIX.1c interfaces).

#### • Spinlock Implementation

```

118343 #include <pthread.h>
118344 #include <time.h>
118345 #include <errno.h>

118346 int pthread_mutex_timedlock(pthread_mutex_t *mutex,
118347 const struct timespec *timeout)
118348 {
118349 struct timespec timenow;
118350 while (pthread_mutex_trylock(mutex) == EBUSY)
118351 {
118352 clock_gettime(CLOCK_REALTIME, &timenow);
118353 if (timespec_cmp(&timenow, timeout) >= 0)
118354 {
118355 return ETIMEDOUT;
118356 }
118357 pthread_yield();
118358 }
118359 return 0;
118360 }

```

The Spinlock implementation is generally unsuitable for any application using priority-based thread scheduling policies such as `SCHED_FIFO` or `SCHED_RR`, since the mutex could currently be held by a thread of lower priority within the same allocation domain, but since the waiting thread never blocks, only threads of equal or higher priority will ever run, and the mutex cannot be unlocked. Setting priority inheritance or priority ceiling protocol on the mutex does not solve this problem, since the priority of a mutex owning thread is only boosted if higher priority threads are blocked waiting for the mutex; clearly not the case for this spinlock.

#### • Condition Wait Implementation

```

118370 #include <pthread.h>
118371 #include <time.h>
118372 #include <errno.h>

118373 struct timed_mutex
118374 {
118375 int locked;
118376 pthread_mutex_t mutex;

```

```

118377 pthread_cond_t cond;
118378 };
118379 typedef struct timed_mutex timed_mutex_t;
118380 int timed_mutex_lock(timed_mutex_t *tm,
118381 const struct timespec *timeout)
118382 {
118383 int timedout=FALSE;
118384 int error_status;
118385
118386 pthread_mutex_lock(&tm->mutex);
118387
118388 while (tm->locked && !timedout)
118389 {
118390 if ((error_status=pthread_cond_timedwait(&tm->cond,
118391 &tm->mutex,
118392 timeout))!=0)
118393 {
118394 if (error_status==ETIMEDOUT) timedout = TRUE;
118395 }
118396 }
118397
118398 if(timedout)
118399 {
118400 pthread_mutex_unlock(&tm->mutex);
118401 return ETIMEDOUT;
118402 }
118403 else
118404 {
118405 tm->locked = TRUE;
118406 pthread_mutex_unlock(&tm->mutex);
118407 return 0;
118408 }
118409 }
118410 void timed_mutex_unlock(timed_mutex_t *tm)
118411 {
118412 pthread_mutex_lock(&tm->mutex); / for case assignment not atomic /
118413 tm->locked = FALSE;
118414 pthread_mutex_unlock(&tm->mutex);
118415 pthread_cond_signal(&tm->cond);
118416 }

```

118414 The Condition Wait implementation effectively substitutes the *pthread\_cond\_timedwait()*  
118415 function (which is currently timed out) for the desired *pthread\_mutex\_timedlock()*. Since  
118416 waits on condition variables currently do not include protocols which avoid priority  
118417 inversion, this method is generally unsuitable for realtime applications because it does not  
118418 provide the same priority inversion protection as the untimed *pthread\_mutex\_lock()*. Also,  
118419 for any given implementations of the current mutex and condition variable primitives, this  
118420 library implementation has a performance cost at least 2.5 times that of the untimed  
118421 *pthread\_mutex\_lock()* even in the case where the timed mutex is readily locked without  
118422 blocking (the interfaces required for this case are shown in bold). Even in uniprocessors or  
118423 where assignment is atomic, at least an additional *pthread\_cond\_signal()* is required.  
118424 *pthread\_mutex\_timedlock()* could be implemented at effectively no performance penalty in  
118425 this case because the timeout parameters need only be considered after it is determined  
118426 that the mutex cannot be locked immediately.

118427 Thus it has not yet been shown that the full semantics of mutex locking with timeout can  
 118428 be efficiently and reliably achieved using existing interfaces. Even if the existence of an  
 118429 acceptable library implementation were proven, it is difficult to justify why the interface  
 118430 itself should not be made portable, especially considering approval for the other four  
 118431 timeouts.

118432 • Rationale for Library Implementation of *pthread\_timedjoin()*

118433 Library implementation of *pthread\_timedjoin()*:

```

118434 /*
118435 * Construct a thread variety entirely from existing functions
118436 * with which a join can be done, allowing the join to time out.
118437 */
118438 #include <pthread.h>
118439 #include <time.h>
118440
118441 struct timed_thread {
118442 pthread_t t;
118443 pthread_mutex_t m;
118444 int exiting;
118445 pthread_cond_t exit_c;
118446 void *(*start_routine)(void *arg);
118447 void *arg;
118448 void *status;
118449 };
118450
118451 typedef struct timed_thread *timed_thread_t;
118452 static pthread_key_t timed_thread_key;
118453 static pthread_once_t timed_thread_once = PTHREAD_ONCE_INIT;
118454
118455 static void timed_thread_init()
118456 {
118457 pthread_key_create(&timed_thread_key, NULL);
118458 }
118459
118460 static void *timed_thread_start_routine(void *args)
118461 /*
118462 * Routine to establish thread-specific data value and run the actual
118463 * thread start routine which was supplied to timed_thread_create().
118464 */
118465 {
118466 timed_thread_t tt = (timed_thread_t) args;
118467 pthread_once(&timed_thread_once, timed_thread_init);
118468 pthread_setspecific(timed_thread_key, (void *)tt);
118469 timed_thread_exit((tt->start_routine)(tt->arg));
118470 }
118471
118472 int timed_thread_create(timed_thread_t ttp, const pthread_attr_t *attr,
118473 void *(*start_routine)(void *), void *arg)
118474 /*
118475 * Allocate a thread which can be used with timed_thread_join().
118476 */
118477 {
118478 timed_thread_t tt;

```

```

118474 int result;
118475 tt = (timed_thread_t) malloc(sizeof(struct timed_thread));
118476 pthread_mutex_init(&tt->m, NULL);
118477 tt->exiting = FALSE;
118478 pthread_cond_init(&tt->exit_c, NULL);
118479 tt->start_routine = start_routine;
118480 tt->arg = arg;
118481 tt->status = NULL;
118482
118483 if ((result = pthread_create(&tt->t, attr,
118484 timed_thread_start_routine, (void *)tt)) != 0) {
118485 free(tt);
118486 return result;
118487 }
118488 pthread_detach(tt->t);
118489 ttp = tt;
118490 return 0;
118491 }
118492
118493 int timed_thread_join(timed_thread_t tt,
118494 struct timespec *timeout,
118495 void **status)
118496 {
118497 int result;
118498 pthread_mutex_lock(&tt->m);
118499 result = 0;
118500 /*
118501 * Wait until the thread announces that it is exiting,
118502 * or until timeout.
118503 */
118504 while (result == 0 && ! tt->exiting) {
118505 result = pthread_cond_timedwait(&tt->exit_c, &tt->m, timeout);
118506 }
118507 pthread_mutex_unlock(&tt->m);
118508 if (result == 0 && tt->exiting) {
118509 *status = tt->status;
118510 free((void *)tt);
118511 return result;
118512 }
118513 return result;
118514 }
118515
118516 void timed_thread_exit(void *status)
118517 {
118518 timed_thread_t tt;
118519 void *specific;
118520
118521 if ((specific=pthread_getspecific(timed_thread_key)) == NULL){
118522 /*
118523 * Handle cases which won't happen with correct usage.
118524 */
118525 pthread_exit(NULL);
118526 }

```

```

118523 tt = (timed_thread_t) specific;
118524 pthread_mutex_lock(&tt->m);
118525 /*
118526 * Tell a joiner that we're exiting.
118527 */
118528 tt->status = status;
118529 tt->exiting = TRUE;
118530 pthread_cond_signal(&tt->exit_c);
118531 pthread_mutex_unlock(&tt->m);
118532 /*
118533 * Call pthread exit() to call destructors and really
118534 * exit the thread.
118535 */
118536 pthread_exit(NULL);
118537 }

```

118538 The *pthread\_join()* C-language example shown above demonstrates that it is possible,  
 118539 using existing pthread facilities, to construct a variety of thread which allows for joining  
 118540 such a thread, but which allows the join operation to time out. It does this by using a  
 118541 *pthread\_cond\_timedwait()* to wait for the thread to exit. A **timed\_thread\_t** descriptor  
 118542 structure is used to pass parameters from the creating thread to the created thread, and  
 118543 from the exiting thread to the joining thread. This implementation is roughly equivalent to  
 118544 what a normal *pthread\_join()* implementation would do, with the single change being that  
 118545 *pthread\_cond\_timedwait()* is used in place of a simple *pthread\_cond\_wait()*.

118546 Since it is possible to implement such a facility entirely from existing pthread interfaces,  
 118547 and with roughly equal efficiency and complexity to an implementation which would be  
 118548 provided directly by a pthreads implementation, it was the consensus of the working  
 118549 group members that any *pthread\_timedjoin()* facility would be unnecessary, and should not  
 118550 be provided.

#### 118551 • Form of the Timeout Interfaces

118552 The working group considered a number of alternative ways to add timeouts to blocking  
 118553 services. At first, a system interface which would specify a one-shot or persistent timeout  
 118554 to be applied to subsequent blocking services invoked by the calling process or thread was  
 118555 considered because it allowed all blocking services to be timed out in a uniform manner  
 118556 with a single additional interface; this was rather quickly rejected because it could easily  
 118557 result in the wrong services being timed out.

118558 It was suggested that a timeout value might be specified as an attribute of the object  
 118559 (semaphore, mutex, message queue, and so on), but there was no consensus on this, either  
 118560 on a case-by-case basis or for all timeouts.

118561 Looking at the two existing timeouts for blocking services indicates that the working  
 118562 group members favor a separate interface for the timed version of a function. However,  
 118563 *pthread\_cond\_timedwait()* utilizes an absolute timeout value while *sigtimedwait()* uses a  
 118564 relative timeout value. The working group members agreed that relative timeout values  
 118565 are appropriate where the timeout mechanism's primary use was to deal with an  
 118566 unexpected or error situation, but they are inappropriate when the timeout must expire at  
 118567 a particular time, or before a specific deadline. For the timeouts being introduced in  
 118568 POSIX.1-200x, the working group considered allowing both relative and absolute timeouts  
 118569 as is done with POSIX.1b timers, but ultimately favored the simpler absolute timeout form.

118570 An absolute time measure can be easily implemented on top of an interface that specifies  
 118571 relative time, by reading the clock, calculating the difference between the current time and

118572 the desired wake-up time, and issuing a relative timeout call. But there is a race condition  
118573 with this approach because the thread could be preempted after reading the clock, but  
118574 before making the timed-out call; in this case, the thread would be awakened later than it  
118575 should and, thus, if the wake-up time represented a deadline, it would miss it.

118576 There is also a race condition when trying to build a relative timeout on top of an interface  
118577 that specifies absolute timeouts. In this case, the clock would have to be read to calculate  
118578 the absolute wake-up time as the sum of the current time plus the relative timeout interval.  
118579 In this case, if the thread is preempted after reading the clock but before making the timed-  
118580 out call, the thread would be awakened earlier than desired.

118581 But the race condition with the absolute timeouts interface is not as bad as the one that  
118582 happens with the relative timeout interface, because there are simple workarounds. For the  
118583 absolute timeouts interface, if the timing requirement is a deadline, the deadline can still  
118584 be met because the thread woke up earlier than the deadline. If the timeout is just used as  
118585 an error recovery mechanism, the precision of timing is not really important. If the timing  
118586 requirement is that between actions A and B a minimum interval of time must elapse, the  
118587 absolute timeout interface can be safely used by reading the clock after action A has been  
118588 started. It could be argued that, since the call with the absolute timeout is atomic from the  
118589 application point of view, it is not possible to read the clock after action A, if this action is  
118590 part of the timed-out call. But looking at the nature of the calls for which timeouts are  
118591 specified (locking a mutex, waiting for a semaphore, waiting for a message, or waiting  
118592 until there is space in a message queue), the timeouts that an application would build on  
118593 these actions would not be triggered by these actions themselves, but by some other  
118594 external action. For example, if waiting for a message to arrive to a message queue, and  
118595 waiting for at least 20 milliseconds, this time interval would start to be counted from some  
118596 event that would trigger both the action that produces the message, as well as the action  
118597 that waits for the message to arrive, and not by the wait-for-message operation itself. In  
118598 this case, the workaround proposed above could be used.

118599 For these reasons, the absolute timeout is preferred over the relative timeout interface.

## 118600 B.2.9 Threads

118601 Threads will normally be more expensive than subroutines (or functions, routines, and so on) if  
118602 specialized hardware support is not provided. Nevertheless, threads should be sufficiently  
118603 efficient to encourage their use as a medium to fine-grained structuring mechanism for  
118604 parallelism in an application. Structuring an application using threads then allows it to take  
118605 immediate advantage of any underlying parallelism available in the host environment. This  
118606 means implementors are encouraged to optimize for fast execution at the possible expense of  
118607 efficient utilization of storage. For example, a common thread creation technique is to cache  
118608 appropriate thread data structures. That is, rather than releasing system resources, the  
118609 implementation retains these resources and reuses them when the program next asks to create a  
118610 new thread. If this reuse of thread resources is to be possible, there has to be very little unique  
118611 state associated with each thread, because any such state has to be reset when the thread is  
118612 reused.

118613 **Thread Creation Attributes**

118614 Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to  
 118615 support probable future standardization in these areas without requiring that the interface itself  
 118616 be changed.

118617 Attributes objects provide clean isolation of the configurable aspects of threads. For example,  
 118618 “stack size” is an important attribute of a thread, but it cannot be expressed portably. When  
 118619 porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects  
 118620 can help by allowing the changes to be isolated in a single place, rather than being spread across  
 118621 every instance of thread creation.

118622 Attributes objects can be used to set up *classes* of threads with similar attributes; for example,  
 118623 “threads with large stacks and high priority” or “threads with minimal stacks”. These classes  
 118624 can be defined in a single place and then referenced wherever threads need to be created.  
 118625 Changes to “class” decisions become straightforward, and detailed analysis of each  
 118626 *pthread\_create()* call is not required.

118627 The attributes objects are defined as opaque types as an aid to extensibility. If these objects had  
 118628 been specified as structures, adding new attributes would force recompilation of all multi-  
 118629 threaded programs when the attributes objects are extended; this might not be possible if  
 118630 different program components were supplied by different vendors.

118631 Additionally, opaque attributes objects present opportunities for improving performance.  
 118632 Argument validity can be checked once when attributes are set, rather than each time a thread is  
 118633 created. Implementations will often need to cache kernel objects that are expensive to create.  
 118634 Opaque attributes objects provide an efficient mechanism to detect when cached objects become  
 118635 invalid due to attribute changes.

118636 Because assignment is not necessarily defined on a given opaque type, implementation-defined  
 118637 default values cannot be defined in a portable way. The solution to this problem is to allow  
 118638 attribute objects to be initialized dynamically by attributes object initialization functions, so that  
 118639 default values can be supplied automatically by the implementation.

118640 The following proposal was provided as a suggested alternative to the supplied attributes:

- 118641 1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to  
 118642 the initialization routines (*pthread\_create()*, *pthread\_mutex\_init()*, *pthread\_cond\_init()*). The  
 118643 parameter containing the flags should be an opaque type for extensibility. If no flags are  
 118644 set in the parameter, then the objects are created with default characteristics. An  
 118645 implementation may specify implementation-defined flag values and associated  
 118646 behavior.
- 118647 2. If further specialization of mutexes and condition variables is necessary, implementations  
 118648 may specify additional procedures that operate on the **pthread\_mutex\_t** and  
 118649 **pthread\_cond\_t** objects (instead of on attributes objects).

118650 The difficulties with this solution are:

- 118651 1. A bitmask is not opaque if bits have to be set into bit-vector attributes objects using  
 118652 explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**,  
 118653 application programmers need to know the location of each bit. If bits are set or read by  
 118654 encapsulation (that is, *get\*()* or *set\*()* functions), then the bitmask is merely an  
 118655 implementation of attributes objects as currently defined and should not be exposed to  
 118656 the programmer.
- 118657 2. Many attributes are not Boolean or very small integral values. For example, scheduling  
 118658 policy may be placed in 3 bits or 4 bits, but priority requires 5 bits or more, thereby taking  
 118659 up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this,



118660 the bitmask can only reasonably control whether particular attributes are set or not, and it  
 118661 cannot serve as the repository of the value itself. The value needs to be specified as a  
 118662 function parameter (which is non-extensible), or by setting a structure field (which is non-  
 118663 opaque), or by *get\**() and *set\**() functions (making the bitmask a redundant addition to  
 118664 the attributes objects).

118665 Stack size is defined as an optional attribute because the very notion of a stack is inherently  
 118666 machine-dependent. Some implementations may not be able to change the size of the stack, for  
 118667 example, and others may not need to because stack pages may be discontinuous and can be  
 118668 allocated and released on demand.

118669 The attribute mechanism has been designed in large measure for extensibility. Future extensions  
 118670 to the attribute mechanism or to any attributes object defined in POSIX.1-200x have to be done  
 118671 with care so as not to affect binary-compatibility.

118672 Attribute objects, even if allocated by means of dynamic allocation functions such as *malloc*(),  
 118673 may have their size fixed at compile time. This means, for example, a *pthread\_create*() in an  
 118674 implementation with extensions to the **pthread\_attr\_t** cannot look beyond the area that the  
 118675 binary application assumes is valid. This suggests that implementations should maintain a size  
 118676 field in the attributes object, as well as possibly version information, if extensions in different  
 118677 directions (possibly by different vendors) are to be accommodated.

### 118678 Thread Implementation Models

118679 There are various thread implementation models. At one end of the spectrum is the “library-  
 118680 thread model”. In such a model, the threads of a process are not visible to the operating system  
 118681 kernel, and the threads are not kernel-scheduled entities. The process is the only kernel-  
 118682 scheduled entity. The process is scheduled onto the processor by the kernel according to the  
 118683 scheduling attributes of the process. The threads are scheduled onto the single kernel-scheduled  
 118684 entity (the process) by the runtime library according to the scheduling attributes of the threads.  
 118685 A problem with this model is that it constrains concurrency. Since there is only one kernel-  
 118686 scheduled entity (namely, the process), only one thread per process can execute at a time. If the  
 118687 thread that is executing blocks on I/O, then the whole process blocks.

118688 At the other end of the spectrum is the “kernel-thread model”. In this model, all threads are  
 118689 visible to the operating system kernel. Thus, all threads are kernel-scheduled entities, and all  
 118690 threads can concurrently execute. The threads are scheduled onto processors by the kernel  
 118691 according to the scheduling attributes of the threads. The drawback to this model is that the  
 118692 creation and management of the threads entails operating system calls, as opposed to subroutine  
 118693 calls, which makes kernel threads heavier weight than library threads.

118694 Hybrids of these two models are common. A hybrid model offers the speed of library threads  
 118695 and the concurrency of kernel threads. In hybrid models, a process has some (relatively small)  
 118696 number of kernel scheduled entities associated with it. It also has a potentially much larger  
 118697 number of library threads associated with it. Some library threads may be bound to kernel-  
 118698 scheduled entities, while the other library threads are multiplexed onto the remaining kernel-  
 118699 scheduled entities. There are two levels of thread scheduling:

- 118700 1. The runtime library manages the scheduling of (unbound) library threads onto kernel-  
 118701 scheduled entities.
- 118702 2. The kernel manages the scheduling of kernel-scheduled entities onto processors.

118703 For this reason, a hybrid model is referred to as a two-level threads scheduling model. In this  
 118704 model, the process can have multiple concurrently executing threads; specifically, it can have as  
 118705 many concurrently executing threads as it has kernel-scheduled entities.

118706 **Thread-Specific Data**

118707 Many applications require that a certain amount of context be maintained on a per-thread basis  
 118708 across procedure calls. A common example is a multi-threaded library routine that allocates  
 118709 resources from a common pool and maintains an active resource list for each thread. The thread-  
 118710 specific data interface provided to meet these needs may be viewed as a two-dimensional array  
 118711 of values with keys serving as the row index and thread IDs as the column index (although the  
 118712 implementation need not work this way).

## 118713 • Models

118714 Three possible thread-specific data models were considered:

## 118715 1. No Explicit Support

118716 A standard thread-specific data interface is not strictly necessary to support  
 118717 applications that require per-thread context. One could, for example, provide a hash  
 118718 function that converted a **pthread\_t** into an integer value that could then be used to  
 118719 index into a global array of per-thread data pointers. This hash function, in  
 118720 conjunction with *pthread\_self()*, would be all the interface required to support a  
 118721 mechanism of this sort. Unfortunately, this technique is cumbersome. It can lead to  
 118722 duplicated code as each set of cooperating modules implements their own per-  
 118723 thread data management schemes.

118724 2. Single (**void \***) Pointer

118725 Another technique would be to provide a single word of per-thread storage and a  
 118726 pair of functions to fetch and store the value of this word. The word could then hold  
 118727 a pointer to a block of per-thread memory. The allocation, partitioning, and general  
 118728 use of this memory would be entirely up to the application. Although this method  
 118729 is not as problematic as technique 1, it suffers from interoperability problems. For  
 118730 example, all modules using the per-thread pointer would have to agree on a  
 118731 common usage protocol.

## 118732 3. Key/Value Mechanism

118733 This method associates an opaque key (for example, stored in a variable of type  
 118734 **pthread\_key\_t**) with each per-thread datum. These keys play the role of identifiers  
 118735 for per-thread data. This technique is the most generic and avoids the problems  
 118736 noted above, albeit at the cost of some complexity.

118737 The primary advantage of the third model is its information hiding properties. Modules  
 118738 using this model are free to create and use their own key(s) independent of all other such  
 118739 usage, whereas the other models require that all modules that use thread-specific context  
 118740 explicitly cooperate with all other such modules. The data-independence provided by the  
 118741 third model is worth the additional interface.

## 118742 • Requirements

118743 It is important that it be possible to implement the thread-specific data interface without  
 118744 the use of thread private memory. To do otherwise would increase the weight of each  
 118745 thread, thereby limiting the range of applications for which the threads interfaces provided  
 118746 by POSIX.1-200x is appropriate.

118747 The values that one binds to the key via *pthread\_setspecific()* may, in fact, be pointers to  
 118748 shared storage locations available to all threads. It is only the key/value bindings that are  
 118749 maintained on a per-thread basis, and these can be kept in any portion of the address space  
 118750 that is reserved for use by the calling thread (for example, on the stack). Thus, no per-  
 118751 thread MMU state is required to implement the interface. On the other hand, there is

118752 nothing in the interface specification to preclude the use of a per-thread MMU state if it is  
 118753 available (for example, the key values returned by *pthread\_key\_create()* could be thread  
 118754 private memory addresses).

118755 • Standardization Issues

118756 Thread-specific data is a requirement for a usable thread interface. The binding described  
 118757 in this section provides a portable thread-specific data mechanism for languages that do  
 118758 not directly support a thread-specific storage class. A binding to POSIX.1-200x for a  
 118759 language that does include such a storage class need not provide this specific interface.

118760 If a language were to include the notion of thread-specific storage, it would be desirable  
 118761 (but *not* required) to provide an implementation of the pthreads thread-specific data  
 118762 interface based on the language feature. For example, assume that a compiler for a C-like  
 118763 language supports a *private* storage class that provides thread-specific storage. Something  
 118764 similar to the following macros might be used to effect a compatible implementation:

```
118765 #define pthread_key_t private void *
118766 #define pthread_key_create(key) /* no-op */
118767 #define pthread_setspecific(key,value) (key)=(value)
118768 #define pthread_getspecific(key) (key)
```

118769 **Note:** For the sake of clarity, this example ignores destructor functions. A correct  
 118770 implementation would have to support them.

118771 **Barriers**

118772 • Background

118773 Barriers are typically used in parallel DO/FOR loops to ensure that all threads have  
 118774 reached a particular stage in a parallel computation before allowing any to proceed to the  
 118775 next stage. Highly efficient implementation is possible on machines which support a  
 118776 “Fetch and Add” operation as described in the referenced Almasi and Gottlieb (1989).

118777 The use of return value PTHREAD\_BARRIER\_SERIAL\_THREAD is shown in the  
 118778 following example:

```
118779 if ((status=pthread_barrier_wait(&barrier)) ==
118780 PTHREAD_BARRIER_SERIAL_THREAD) {
118781 ...serial section
118782 }
118783 else if (status != 0) {
118784 ...error processing
118785 }
118786 status=pthread_barrier_wait(&barrier);
118787 ...
```

118788 This behavior allows a serial section of code to be executed by one thread as soon as all  
 118789 threads reach the first barrier. The second barrier prevents the other threads from  
 118790 proceeding until the serial section being executed by the one thread has completed.

118791 Although barriers can be implemented with mutexes and condition variables, the  
 118792 referenced Almasi and Gottlieb (1989) provides ample illustration that such  
 118793 implementations are significantly less efficient than is possible. While the relative  
 118794 efficiency of barriers may well vary by implementation, it is important that they be  
 118795 recognized in the POSIX.1-200x to facilitate applications portability while providing the  
 118796 necessary freedom to implementors.

- 118797
- Lack of Timeout Feature
- 118798 Alternate versions of most blocking routines have been provided to support watchdog  
118799 timeouts. No alternate interface of this sort has been provided for barrier waits for the  
118800 following reasons:
- Multiple threads may use different timeout values, some of which may be indefinite.  
118801 It is not clear which threads should break through the barrier with a timeout error if  
118802 and when these timeouts expire.
  - The barrier may become unusable once a thread breaks out of a *pthread\_barrier\_wait()*  
118803 with a timeout error. There is, in general, no way to guarantee the consistency of a  
118804 barrier's internal data structures once a thread has timed out of a  
118805 *pthread\_barrier\_wait()*. Even the inclusion of a special barrier reinitialization function  
118806 would not help much since it is not clear how this function would affect the behavior  
118807 of threads that reach the barrier between the original timeout and the call to the  
118808 reinitialization function.
- 118809  
118810

## 118811 Spin Locks

- Background
- 118812
- 118813 Spin locks represent an extremely low-level synchronization mechanism suitable primarily  
118814 for use on shared memory multi-processors. It is typically an atomically modified Boolean  
118815 value that is set to one when the lock is held and to zero when the lock is freed.
- 118816 When a caller requests a spin lock that is already held, it typically spins in a loop testing  
118817 whether the lock has become available. Such spinning wastes processor cycles so the lock  
118818 should only be held for short durations and not across sleep/block operations. Callers  
118819 should unlock spin locks before calling sleep operations.
- 118820 Spin locks are available on a variety of systems. The functions included in POSIX.1-200x  
118821 are an attempt to standardize that existing practice.
- Lack of Timeout Feature
- 118822
- 118823 Alternate versions of most blocking routines have been provided to support watchdog  
118824 timeouts. No alternate interface of this sort has been provided for spin locks for the  
118825 following reasons:
- It is impossible to determine appropriate timeout intervals for spin locks in a  
118826 portable manner. The amount of time one can expect to spend spin-waiting is  
118827 inversely proportional to the degree of parallelism provided by the system.  
118828
- 118829 It can vary from a few cycles when each competing thread is running on its own  
118830 processor, to an indefinite amount of time when all threads are multiplexed on a  
118831 single processor (which is why spin locking is not advisable on uniprocessors).
- When used properly, the amount of time the calling thread spends waiting on a spin  
118832 lock should be considerably less than the time required to set up a corresponding  
118833 watchdog timer. Since the primary purpose of spin locks is to provide a low-  
118834 overhead synchronization mechanism for multi-processors, the overhead of a  
118835 timeout mechanism was deemed unacceptable.  
118836
- 118837 It was also suggested that an additional *count* argument be provided (on the  
118838 *pthread\_spin\_lock()* call) in lieu of a true timeout so that a spin lock call could fail gracefully  
118839 if it was unable to apply the lock after *count* attempts. This idea was rejected because it is  
118840 not existing practice. Furthermore, the same effect can be obtained with  
118841 *pthread\_spin\_trylock()*, as illustrated below:

```

118842 int n = MAX_SPIN;
118843 while (--n >= 0)
118844 {
118845 if (!pthread_spin_try_lock(...))
118846 break;
118847 }
118848 if (n >= 0)
118849 {
118850 /* Successfully acquired the lock */
118851 }
118852 else
118853 {
118854 /* Unable to acquire the lock */
118855 }

```

118856 • *process-shared* Attribute

118857 The initialization functions associated with most POSIX synchronization objects (for  
118858 example, mutexes, barriers, and read-write locks) take an attributes object with a *process-*  
118859 *shared* attribute that specifies whether or not the object is to be shared across processes. In  
118860 the draft corresponding to the first balloting round, two separate initialization functions  
118861 are provided for spin locks, however: one for spin locks that were to be shared across  
118862 processes (*spin\_init()*), and one for locks that were only used by multiple threads within a  
118863 single process (*pthread\_spin\_init()*). This was done so as to keep the overhead associated  
118864 with spin waiting to an absolute minimum. However, the balloting group requested that,  
118865 since the overhead associated to a bit check was small, spin locks should be consistent with  
118866 the rest of the synchronization primitives, and thus the *process-shared* attribute was  
118867 introduced for spin locks.

118868 • Spin Locks *versus* Mutexes

118869 It has been suggested that mutexes are an adequate synchronization mechanism and spin  
118870 locks are not necessary. Locking mechanisms typically must trade off the processor  
118871 resources consumed while setting up to block the thread and the processor resources  
118872 consumed by the thread while it is blocked. Spin locks require very little resources to set  
118873 up the blocking of a thread. Existing practice is to simply loop, repeating the atomic  
118874 locking operation until the lock is available. While the resources consumed to set up  
118875 blocking of the thread are low, the thread continues to consume processor resources while  
118876 it is waiting.

118877 On the other hand, mutexes may be implemented such that the processor resources  
118878 consumed to block the thread are large relative to a spin lock. After detecting that the  
118879 mutex lock is not available, the thread must alter its scheduling state, add itself to a set of  
118880 waiting threads, and, when the lock becomes available again, undo all of this before taking  
118881 over ownership of the mutex. However, while a thread is blocked by a mutex, no processor  
118882 resources are consumed.

118883 Therefore, spin locks and mutexes may be implemented to have different characteristics.  
118884 Spin locks may have lower overall overhead for very short-term blocking, and mutexes  
118885 may have lower overall overhead when a thread will be blocked for longer periods of time.  
118886 The presence of both interfaces allows implementations with these two different  
118887 characteristics, both of which may be useful to a particular application.

118888 It has also been suggested that applications can build their own spin locks from the  
118889 *pthread\_mutex\_trylock()* function:

118890           while (pthread\_mutex\_trylock(&mutex));

118891           The apparent simplicity of this construct is somewhat deceiving, however. While the actual  
118892           wait is quite efficient, various guarantees on the integrity of mutex objects (for example,  
118893           priority inheritance rules) may add overhead to the successful path of the trylock  
118894           operation that is not required of spin locks. One could, of course, add an attribute to the  
118895           mutex to bypass such overhead, but the very act of finding and testing this attribute  
118896           represents more overhead than is found in the typical spin lock.

118897           The need to hold spin lock overhead to an absolute minimum also makes it impossible to  
118898           provide guarantees against starvation similar to those provided for mutexes or read-write  
118899           locks. The overhead required to implement such guarantees (for example, disabling  
118900           preemption before spinning) may well exceed the overhead of the spin wait itself by many  
118901           orders of magnitude. If a “safe” spin wait seems desirable, it can always be provided  
118902           (albeit at some performance cost) via appropriate mutex attributes.

### 118903           **Robust Mutexes**

118904           Robust mutexes are intended to protect applications that use mutexes to protect data shared  
118905           between different processes. If a process is terminated by a signal while a thread is holding a  
118906           mutex, there is no chance for the process to clean up after it. Waiters for the locked mutex might  
118907           wait indefinitely.

118908           With robust mutexes the problem can be solved: whenever a fatal signal terminates a process,  
118909           current or future waiters of the mutex are notified about this fact. The locking function provides  
118910           notification of this condition through the error condition [EOWNERDEAD]. A thread then has  
118911           the chance to clean up the state protected by the mutex and mark the state as consistent again by  
118912           a call to *pthread\_mutex\_consistent()*.

118913           Pre-existing implementations have used the semantics of robust mutexes for a variety of  
118914           situations, some of them not defined in the standard. Where a normally terminated process (i.e.,  
118915           when one thread calls *exit()*) causes notification of other waiters of robust mutexes if the mutex  
118916           is locked by any thread in the process. This behavior is defined in the standard and makes sense  
118917           because no thread other than the thread calling *exit()* has the chance to clean up its data.

118918           If a thread is terminated by cancellation or if it calls *pthread\_exit()*, the situation is different. In  
118919           both these situations the thread has the chance to clean up after itself by registering appropriate  
118920           cleanup handlers. There is no real reason to demand that other waiters for a robust mutex the  
118921           terminating thread owns are notified. The committee felt that this is actively encouraging bad  
118922           practice because programmers are tempted to rely on the robust mutex semantics instead of  
118923           correctly cleaning up after themselves.

118924           Therefore, the standard does not require notification of other waiters at the time a thread is  
118925           terminated while the process continues to run. The mutex is still recognized as being locked by  
118926           the process (with the thread gone it makes no sense to refer to the thread owning the mutex).  
118927           Therefore, a terminating process will cause notifications about the dead owner to be sent to all  
118928           waiters. This delay in the notification is not required, but programmers cannot rely on prompt  
118929           notification after a thread is terminated.

118930           For the same reason is it not required that an implementation supports robust mutexes that are  
118931           not shared between processes. If a robust mutex is used only within one process, all the cleanup  
118932           can be performed by the threads themselves by registering appropriate cleanup handlers. Fatal  
118933           signals are of no importance in this case because after the signal is delivered there is no thread  
118934           remaining to use the mutex.

118935           Some implementations might choose to support intra-process robust mutexes and they might  
118936           also send notification of a dead owner right after the previous owner died. But applications

118937 must not rely on this. Applications should only use robust mutexes for the purpose of handling  
 118938 fatal signals in situations where inter-process mutexes are in use.

### 118939 Supported Threads Functions

118940 On POSIX-conforming systems, the following symbolic constants are always conforming:

118941 `_POSIX_READER_WRITER_LOCKS`  
 118942 `_POSIX_THREADS`

118943 Therefore, the following threads functions are always supported:

|                                                   |                                              |
|---------------------------------------------------|----------------------------------------------|
| 118944 <code>pthread_atfork()</code>              | <code>pthread_mutex_destroy()</code>         |
| 118945 <code>pthread_attr_destroy()</code>        | <code>pthread_mutex_init()</code>            |
| 118946 <code>pthread_attr_getdetachstate()</code> | <code>pthread_mutex_lock()</code>            |
| 118947 <code>pthread_attr_getguardsize()</code>   | <code>pthread_mutex_trylock()</code>         |
| 118948 <code>pthread_attr_getschedparam()</code>  | <code>pthread_mutex_unlock()</code>          |
| 118949 <code>pthread_attr_init()</code>           | <code>pthread_mutexattr_destroy()</code>     |
| 118950 <code>pthread_attr_setdetachstate()</code> | <code>pthread_mutexattr_getpshared()</code>  |
| 118951 <code>pthread_attr_setguardsize()</code>   | <code>pthread_mutexattr_gettype()</code>     |
| 118952 <code>pthread_attr_setschedparam()</code>  | <code>pthread_mutexattr_init()</code>        |
| 118953 <code>pthread_cancel()</code>              | <code>pthread_mutexattr_setpshared()</code>  |
| 118954 <code>pthread_cleanup_pop()</code>         | <code>pthread_mutexattr_settype()</code>     |
| 118955 <code>pthread_cleanup_push()</code>        | <code>pthread_once()</code>                  |
| 118956 <code>pthread_cond_broadcast()</code>      | <code>pthread_rwlock_destroy()</code>        |
| 118957 <code>pthread_cond_destroy()</code>        | <code>pthread_rwlock_init()</code>           |
| 118958 <code>pthread_cond_init()</code>           | <code>pthread_rwlock_rdlock()</code>         |
| 118959 <code>pthread_cond_signal()</code>         | <code>pthread_rwlock_tryrdlock()</code>      |
| 118960 <code>pthread_cond_timedwait()</code>      | <code>pthread_rwlock_trywrlock()</code>      |
| 118961 <code>pthread_cond_wait()</code>           | <code>pthread_rwlock_unlock()</code>         |
| 118962 <code>pthread_condattr_destroy()</code>    | <code>pthread_rwlock_wrlock()</code>         |
| 118963 <code>pthread_condattr_getpshared()</code> | <code>pthread_rwlockattr_destroy()</code>    |
| 118964 <code>pthread_condattr_init()</code>       | <code>pthread_rwlockattr_getpshared()</code> |
| 118965 <code>pthread_condattr_setpshared()</code> | <code>pthread_rwlockattr_init()</code>       |
| 118966 <code>pthread_create()</code>              | <code>pthread_rwlockattr_setpshared()</code> |
| 118967 <code>pthread_detach()</code>              | <code>pthread_self()</code>                  |
| 118968 <code>pthread_equal()</code>               | <code>pthread_setcancelstate()</code>        |
| 118969 <code>pthread_exit()</code>                | <code>pthread_setcanceltype()</code>         |
| 118970 <code>pthread_getconcurrency()</code>      | <code>pthread_setconcurrency()</code>        |
| 118971 <code>pthread_getspecific()</code>         | <code>pthread_setspecific()</code>           |
| 118972 <code>pthread_join()</code>                | <code>pthread_sigmask()</code>               |
| 118973 <code>pthread_key_create()</code>          | <code>pthread_testcancel()</code>            |
| 118974 <code>pthread_key_delete()</code>          | <code>sigwait()</code>                       |
| 118975 <code>pthread_kill()</code>                |                                              |

118976 On POSIX-conforming systems, the symbolic constant `_POSIX_THREAD_SAFE_FUNCTIONS` is  
118977 always defined. Therefore, the following functions are always supported:

|        |                                 |                                 |
|--------|---------------------------------|---------------------------------|
| 118978 | <code>asctime_r()</code>        | <code>getpwuid_r()</code>       |
| 118979 | <code>ctime_r()</code>          | <code>gmtime_r()</code>         |
| 118980 | <code>flockfile()</code>        | <code>localtime_r()</code>      |
| 118981 | <code>ftrylockfile()</code>     | <code>putc_unlocked()</code>    |
| 118982 | <code>funlockfile()</code>      | <code>putchar_unlocked()</code> |
| 118983 | <code>getc_unlocked()</code>    | <code>rand_r()</code>           |
| 118984 | <code>getchar_unlocked()</code> | <code>readdir_r()</code>        |
| 118985 | <code>getgrgid_r()</code>       | <code>strerror_r()</code>       |
| 118986 | <code>getgrnam_r()</code>       | <code>strtok_r()</code>         |
| 118987 | <code>getpwnam_r()</code>       |                                 |

### 118988 Threads Extensions

118989 The following extensions to the IEEE P1003.1c draft standard are now supported in  
118990 POSIX.1-200x as part of the alignment with the Single UNIX Specification:

- 118991 • Extended mutex attribute types
- 118992 • Read-write locks and attributes (also introduced by the IEEE Std 1003.1j-2000 amendment)
- 118993 • Thread concurrency level
- 118994 • Thread stack guard size
- 118995 • Parallel I/O
- 118996 • Robust mutexes

118997 These extensions carefully follow the threads programming model specified in POSIX.1c. As  
118998 with POSIX.1c, all the new functions return zero if successful; otherwise, an error number is  
118999 returned to indicate the error.

119000 The concept of attribute objects was introduced in POSIX.1c to allow implementations to extend  
119001 POSIX.1-200x without changing the existing interfaces. Attribute objects were defined for  
119002 threads, mutexes, and condition variables. Attributes objects are defined as implementation-  
119003 defined opaque types to aid extensibility, and functions are defined to allow attributes to be set  
119004 or retrieved. This model has been followed when adding the new type attribute of  
119005 `pthread_mutexattr_t` or the new read-write lock attributes object `pthread_rwlockattr_t`.

- 119006 • Extended Mutex Attributes

119007 POSIX.1c defines a mutex attributes object as an implementation-defined opaque object of  
119008 type `pthread_mutexattr_t`, and specifies a number of attributes which this object must  
119009 have and a number of functions which manipulate these attributes. These attributes  
119010 include *detachstate*, *inheritsched*, *schedparam*, *schedpolicy*, *contentionscope*, *stackaddr*, and  
119011 *stacksize*.

119012 The System Interfaces volume of POSIX.1-200x specifies another mutex attribute called  
119013 *type*. The *type* attribute allows applications to specify the behavior of mutex locking  
119014 operations in situations where POSIX.1c behavior is undefined. The OSF DCE threads  
119015 implementation, based on Draft 4 of POSIX.1c, specified a similar attribute. Note that the  
119016 names of the attributes have changed somewhat from the OSF DCE threads  
119017 implementation.

119018 The System Interfaces volume of POSIX.1-200x also extends the specification of the  
119019 following POSIX.1c functions which manipulate mutexes:



119020 `pthread_mutex_lock()`  
 119021 `pthread_mutex_trylock()`  
 119022 `pthread_mutex_unlock()`

119023 to take account of the new mutex attribute type and to specify behavior which was  
 119024 declared as undefined in POSIX.1c. How a calling thread acquires or releases a mutex now  
 119025 depends upon the mutex *type* attribute.

119026 The *type* attribute can have the following values:

119027 PTHREAD\_MUTEX\_NORMAL

119028 Basic mutex with no specific error checking built in. Does not report a deadlock error.

119029 PTHREAD\_MUTEX\_RECURSIVE

119030 Allows any thread to recursively lock a mutex. The mutex must be unlocked an equal  
 119031 number of times to release the mutex.

119032 PTHREAD\_MUTEX\_ERRORCHECK

119033 Detects and reports simple usage errors; that is, an attempt to unlock a mutex that is  
 119034 not locked by the calling thread or that is not locked at all, or an attempt to relock a  
 119035 mutex the thread already owns.

119036 PTHREAD\_MUTEX\_DEFAULT

119037 The default mutex type. May be mapped to any of the above mutex types or may be  
 119038 an implementation-defined type.

119039 *Normal* mutexes do not detect deadlock conditions; for example, a thread will hang if it  
 119040 tries to relock a normal mutex that it already owns. Attempting to unlock a mutex locked  
 119041 by another thread, or unlocking an unlocked mutex, results in undefined behavior. Normal  
 119042 mutexes will usually be the fastest type of mutex available on a platform but provide the  
 119043 least error checking.

119044 *Recursive* mutexes are useful for converting old code where it is difficult to establish clear  
 119045 boundaries of synchronization. A thread can relock a recursive mutex without first  
 119046 unlocking it. The relocking deadlock which can occur with normal mutexes cannot occur  
 119047 with this type of mutex. However, multiple locks of a recursive mutex require the same  
 119048 number of unlocks to release the mutex before another thread can acquire the mutex.  
 119049 Furthermore, this type of mutex maintains the concept of an owner. Thus, a thread  
 119050 attempting to unlock a recursive mutex which another thread has locked returns with an  
 119051 error. A thread attempting to unlock a recursive mutex that is not locked returns with an  
 119052 error. Never use a recursive mutex with condition variables because the implicit unlock  
 119053 performed by `pthread_cond_wait()` or `pthread_cond_timedwait()` will not actually release the  
 119054 mutex if it had been locked multiple times.

119055 *Errorcheck* mutexes provide error checking and are useful primarily as a debugging aid. A  
 119056 thread attempting to relock an errorcheck mutex without first unlocking it returns with an  
 119057 error. Again, this type of mutex maintains the concept of an owner. Thus, a thread  
 119058 attempting to unlock an errorcheck mutex which another thread has locked returns with  
 119059 an error. A thread attempting to unlock an errorcheck mutex that is not locked also returns  
 119060 with an error. It should be noted that errorcheck mutexes will almost always be much  
 119061 slower than normal mutexes due to the extra state checks performed.

119062 The default mutex type provides implementation-defined error checking. The default  
 119063 mutex may be mapped to one of the other defined types or may be something entirely  
 119064 different. This enables each vendor to provide the mutex semantics which the vendor feels  
 119065 will be most useful to their target users. Most vendors will probably choose to make  
 119066 normal mutexes the default so as to give applications the benefit of the fastest type of

- 119067 mutexes available on their platform. Check your implementation's documentation.
- 119068 An application developer can use any of the mutex types almost interchangeably as long  
119069 as the application does not depend upon the implementation detecting (or failing to  
119070 detect) any particular errors. Note that a recursive mutex can be used with condition  
119071 variable waits as long as the application never recursively locks the mutex.
- 119072 Two functions are provided for manipulating the *type* attribute of a mutex attributes object.  
119073 This attribute is set or returned in the *type* parameter of these functions. The  
119074 *pthread\_mutexattr\_settype()* function is used to set a specific type value while  
119075 *pthread\_mutexattr\_gettype()* is used to return the type of the mutex. Setting the *type*  
119076 attribute of a mutex attributes object affects only mutexes initialized using that mutex  
119077 attributes object. Changing the *type* attribute does not affect mutexes previously initialized  
119078 using that mutex attributes object.
- 119079 • Read-Write Locks and Attributes
- 119080 The read-write locks introduced have been harmonized with those in IEEE Std  
119081 1003.1j-2000; see also [Section B.2.9.6](#) (on page 3489).
- 119082 Read-write locks (also known as reader-writer locks) allow a thread to exclusively lock  
119083 some shared data while updating that data, or allow any number of threads to have  
119084 simultaneous read-only access to the data.
- 119085 Unlike a mutex, a read-write lock distinguishes between reading data and writing data. A  
119086 mutex excludes all other threads. A read-write lock allows other threads access to the data,  
119087 providing no thread is modifying the data. Thus, a read-write lock is less primitive than  
119088 either a mutex-condition variable pair or a semaphore.
- 119089 Application developers should consider using a read-write lock rather than a mutex to  
119090 protect data that is frequently referenced but seldom modified. Most threads (readers) will  
119091 be able to read the data without waiting and will only have to block when some other  
119092 thread (a writer) is in the process of modifying the data. Conversely a thread that wants to  
119093 change the data is forced to wait until there are no readers. This type of lock is often used  
119094 to facilitate parallel access to data on multi-processor platforms or to avoid context  
119095 switches on single processor platforms where multiple threads access the same data.
- 119096 If a read-write lock becomes unlocked and there are multiple threads waiting to acquire  
119097 the write lock, the implementation's scheduling policy determines which thread acquires  
119098 the read-write lock for writing. If there are multiple threads blocked on a read-write lock  
119099 for both read locks and write locks, it is unspecified whether the readers or a writer  
119100 acquire the lock first. However, for performance reasons, implementations often favor  
119101 writers over readers to avoid potential writer starvation.
- 119102 A read-write lock object is an implementation-defined opaque object of type  
119103 **pthread\_rwlock\_t** as defined in **<pthread.h>**. There are two different sorts of locks  
119104 associated with a read-write lock: a read lock and a write lock.
- 119105 The *pthread\_rwlockattr\_init()* function initializes a read-write lock attributes object with the  
119106 default value for all the attributes defined in the implementation. After a read-write lock  
119107 attributes object has been used to initialize one or more read-write locks, changes to the  
119108 read-write lock attributes object, including destruction, do not affect previously initialized  
119109 read-write locks.
- 119110 Implementations must provide at least the read-write lock attribute *process-shared*. This  
119111 attribute can have the following values:

- 119112 PTHREAD\_PROCESS\_SHARED  
 119113 Any thread of any process that has access to the memory where the read-write lock  
 119114 resides can manipulate the read-write lock.
- 119115 PTHREAD\_PROCESS\_PRIVATE  
 119116 Only threads created within the same process as the thread that initialized the read-  
 119117 write lock can manipulate the read-write lock. This is the default value.
- 119118 The *pthread\_rwlockattr\_setpshared()* function is used to set the *process-shared* attribute of an  
 119119 initialized read-write lock attributes object while the function  
 119120 *pthread\_rwlockattr\_getpshared()* obtains the current value of the *process-shared* attribute.
- 119121 A read-write lock attributes object is destroyed using the *pthread\_rwlockattr\_destroy()*  
 119122 function. The effect of subsequent use of the read-write lock attributes object is undefined.
- 119123 A thread creates a read-write lock using the *pthread\_rwlock\_init()* function. The attributes  
 119124 of the read-write lock can be specified by the application developer; otherwise, the default  
 119125 implementation-defined read-write lock attributes are used if the pointer to the read-write  
 119126 lock attributes object is NULL. In cases where the default attributes are appropriate, the  
 119127 PTHREAD\_RWLOCK\_INITIALIZER macro can be used to initialize statically allocated  
 119128 read-write locks.
- 119129 A thread which wants to apply a read lock to the read-write lock can use either  
 119130 *pthread\_rwlock\_rdlock()* or *pthread\_rwlock\_tryrdlock()*. If *pthread\_rwlock\_rdlock()* is used, the  
 119131 thread acquires a read lock if a writer does not hold the write lock and there are no writers  
 119132 blocked on the write lock. If a read lock is not acquired, the calling thread blocks until it  
 119133 can acquire a lock. However, if *pthread\_rwlock\_tryrdlock()* is used, the function returns  
 119134 immediately with the error [EBUSY] if any thread holds a write lock or there are blocked  
 119135 writers waiting for the write lock.
- 119136 A thread which wants to apply a write lock to the read-write lock can use either of two  
 119137 functions: *pthread\_rwlock\_wrlock()* or *pthread\_rwlock\_trywrlock()*. If *pthread\_rwlock\_wrlock()*  
 119138 is used, the thread acquires the write lock if no other reader or writer threads hold the  
 119139 read-write lock. If the write lock is not acquired, the thread blocks until it can acquire the  
 119140 write lock. However, if *pthread\_rwlock\_trywrlock()* is used, the function returns  
 119141 immediately with the error [EBUSY] if any thread is holding either a read or a write lock.
- 119142 The *pthread\_rwlock\_unlock()* function is used to unlock a read-write lock object held by the  
 119143 calling thread. Results are undefined if the read-write lock is not held by the calling thread.  
 119144 If there are other read locks currently held on the read-write lock object, the read-write  
 119145 lock object remains in the read locked state but without the current thread as one of its  
 119146 owners. If this function releases the last read lock for this read-write lock object, the read-  
 119147 write lock object is put in the unlocked read state. If this function is called to release a write  
 119148 lock for this read-write lock object, the read-write lock object is put in the unlocked state.
- 119149 • Thread Concurrency Level
- 119150 On threads implementations that multiplex user threads onto a smaller set of kernel  
 119151 execution entities, the system attempts to create a reasonable number of kernel execution  
 119152 entities for the application upon application startup.
- 119153 On some implementations, these kernel entities are retained by user threads that block in  
 119154 the kernel. Other implementations do not *timeslice* user threads so that multiple compute-  
 119155 bound user threads can share a kernel thread. On such implementations, some  
 119156 applications may use up all the available kernel execution entities before their user-space  
 119157 threads are used up. The process may be left with user threads capable of doing work for  
 119158 the application but with no way to schedule them.

119159 The `pthread_setconcurrency()` function enables an application to request more kernel  
 119160 entities; that is, specify a desired concurrency level. However, this function merely  
 119161 provides a hint to the implementation. The implementation is free to ignore this request or  
 119162 to provide some other number of kernel entities. If an implementation does not multiplex  
 119163 user threads onto a smaller number of kernel execution entities, the  
 119164 `pthread_setconcurrency()` function has no effect.

119165 The `pthread_setconcurrency()` function may also have an effect on implementations where  
 119166 the kernel mode and user mode schedulers cooperate to ensure that ready user threads are  
 119167 not prevented from running by other threads blocked in the kernel.

119168 The `pthread_getconcurrency()` function always returns the value set by a previous call to  
 119169 `pthread_setconcurrency()`. However, if `pthread_setconcurrency()` was not previously called,  
 119170 this function returns zero to indicate that the threads implementation is maintaining the  
 119171 concurrency level.

119172 • Thread Stack Guard Size

119173 DCE threads introduced the concept of a “thread stack guard size”. Most thread  
 119174 implementations add a region of protected memory to a thread’s stack, commonly known  
 119175 as a “guard region”, as a safety measure to prevent stack pointer overflow in one thread  
 119176 from corrupting the contents of another thread’s stack. The default size of the guard  
 119177 regions attribute is {PAGESIZE} bytes and is implementation-defined.

119178 Some application developers may wish to change the stack guard size. When an  
 119179 application creates a large number of threads, the extra page allocated for each stack may  
 119180 strain system resources. In addition to the extra page of memory, the kernel’s memory  
 119181 manager has to keep track of the different protections on adjoining pages. When this is a  
 119182 problem, the application developer may request a guard size of 0 bytes to conserve system  
 119183 resources by eliminating stack overflow protection.

119184 Conversely an application that allocates large data structures such as arrays on the stack  
 119185 may wish to increase the default guard size in order to detect stack overflow. If a thread  
 119186 allocates two pages for a data array, a single guard page provides little protection against  
 119187 thread stack overflows since the thread can corrupt adjoining memory beyond the guard  
 119188 page.

119189 The System Interfaces volume of POSIX.1-200x defines a new attribute of a thread  
 119190 attributes object; that is, the `guardsize` attribute which allows applications to specify the size  
 119191 of the guard region of a thread’s stack.

119192 Two functions are provided for manipulating a thread’s stack guard size. The  
 119193 `pthread_attr_setguardsize()` function sets the thread `guardsize` attribute, and the  
 119194 `pthread_attr_getguardsize()` function retrieves the current value.

119195 An implementation may round up the requested guard size to a multiple of the  
 119196 configurable system variable {PAGESIZE}. In this case, `pthread_attr_getguardsize()` returns  
 119197 the guard size specified by the previous `pthread_attr_setguardsize()` function call and not  
 119198 the rounded up value.

119199 If an application is managing its own thread stacks using the `stackaddr` attribute, the  
 119200 `guardsize` attribute is ignored and no stack overflow protection is provided. In this case, it is  
 119201 the responsibility of the application to manage stack overflow along with stack allocation.

119202 • Parallel I/O

119203 Suppose two or more threads independently issue read requests on the same file. To read  
 119204 specific data from a file, a thread must first call `lseek()` to seek to the proper offset in the  
 119205 file, and then call `read()` to retrieve the required data. If more than one thread does this at

119206 the same time, the first thread may complete its seek call, but before it gets a chance to  
 119207 issue its read call a second thread may complete its seek call, resulting in the first thread  
 119208 accessing incorrect data when it issues its read call. One workaround is to lock the file  
 119209 descriptor while seeking and reading or writing, but this reduces parallelism and adds  
 119210 overhead.

119211 Instead, the System Interfaces volume of POSIX.1-200x provides two functions to make  
 119212 seek/read and seek/write operations atomic. The file descriptor's current offset is  
 119213 unchanged, thus allowing multiple read and write operations to proceed in parallel. This  
 119214 improves the I/O performance of threaded applications. The *pread()* function is used to do  
 119215 an atomic read of data from a file into a buffer. Conversely, the *pwrite()* function does an  
 119216 atomic write of data from a buffer to a file.

#### 119217 B.2.9.1 Thread-Safety

119218 All functions required by POSIX.1-200x need to be thread-safe. Implementations have to  
 119219 provide internal synchronization when necessary in order to achieve this goal. In certain cases—  
 119220 for example, most floating-point implementations—context switch code may have to manage  
 119221 the writable shared state.

119222 While a read from a pipe of  $\{\text{PIPE\_MAX}\} * 2$  bytes may not generate a single atomic and thread-  
 119223 safe stream of bytes, it should generate “several” (individually atomic) thread-safe streams of  
 119224 bytes. Similarly, while reading from a terminal device may not generate a single atomic and  
 119225 thread-safe stream of bytes, it should generate some finite number of (individually atomic) and  
 119226 thread-safe streams of bytes. That is, concurrent calls to read for a pipe, FIFO, or terminal device  
 119227 are not allowed to result in corrupting the stream of bytes or other internal data. However,  
 119228 *read()*, in these cases, is not required to return a single contiguous and atomic stream of bytes.

119229 It is not required that all functions provided by POSIX.1-200x be either async-cancel-safe or  
 119230 async-signal-safe.

119231 As it turns out, some functions are inherently not thread-safe; that is, their interface  
 119232 specifications preclude reentrancy. For example, some functions (such as *asctime()*) return a  
 119233 pointer to a result stored in memory space allocated by the function on a per-process basis. Such  
 119234 a function is not thread-safe, because its result can be overwritten by successive invocations.  
 119235 Other functions, while not inherently non-thread-safe, may be implemented in ways that lead to  
 119236 them not being thread-safe. For example, some functions (such as *rand()*) store state information  
 119237 (such as a seed value, which survives multiple function invocations) in memory space allocated  
 119238 by the function on a per-process basis. The implementation of such a function is not thread-safe  
 119239 if the implementation fails to synchronize invocations of the function and thus fails to protect  
 119240 the state information. The problem is that when the state information is not protected,  
 119241 concurrent invocations can interfere with one another (for example, applications using *rand()*  
 119242 may see the same seed value).

#### 119243 Thread-Safety and Locking of Existing Functions

119244 Originally, POSIX.1 was not designed to work in a multi-threaded environment, and some  
 119245 implementations of some existing functions will not work properly when executed concurrently.  
 119246 To provide routines that will work correctly in an environment with threads (“thread-safe”), two  
 119247 problems need to be solved:

- 119248 1. Routines that maintain or return pointers to static areas internal to the routine (which  
 119249 may now be shared) need to be modified. The routines *ttyname()* and *localtime()* are  
 119250 examples.

119251 2. Routines that access data space shared by more than one thread need to be modified. The  
119252 *malloc()* function and the *stdio* family routines are examples.

119253 There are a variety of constraints on these changes. The first is compatibility with the existing  
119254 versions of these functions—non-thread-safe functions will continue to be in use for some time,  
119255 as the original interfaces are used by existing code. Another is that the new thread-safe versions  
119256 of these functions represent as small a change as possible over the familiar interfaces provided  
119257 by the existing non-thread-safe versions. The new interfaces should be independent of any  
119258 particular threads implementation. In particular, they should be thread-safe without depending  
119259 on explicit thread-specific memory. Finally, there should be minimal performance penalty due to  
119260 the changes made to the functions.

119261 It is intended that the list of functions from POSIX.1 that cannot be made thread-safe and for  
119262 which corrected versions are provided be complete.

#### 119263 *Thread-Safety and Locking Solutions*

119264 Many of the POSIX.1 functions were thread-safe and did not change at all. However, some  
119265 functions (for example, the math functions typically found in **libm**) are not thread-safe because  
119266 of writable shared global state. For instance, in IEEE Std 754-1985 floating-point  
119267 implementations, the computation modes and flags are global and shared.

119268 Some functions are not thread-safe because a particular implementation is not reentrant,  
119269 typically because of a non-essential use of static storage. These require only a new  
119270 implementation.

119271 Thread-safe libraries are useful in a wide range of parallel (and asynchronous) programming  
119272 environments, not just within pthreads. In order to be used outside the context of pthreads,  
119273 however, such libraries still have to use some synchronization method. These could either be  
119274 independent of the pthread synchronization operations, or they could be a subset of the pthread  
119275 interfaces. Either method results in thread-safe library implementations that can be used without  
119276 the rest of pthreads.

119277 Some functions, such as the *stdio* family interface and dynamic memory allocation functions  
119278 such as *malloc()*, are inter-dependent routines that share resources (for example, buffers) across  
119279 related calls. These require synchronization to work correctly, but they do not require any  
119280 change to their external (user-visible) interfaces.

119281 In some cases, such as *getc()* and *putc()*, adding synchronization is likely to create an  
119282 unacceptable performance impact. In this case, slower thread-safe synchronized functions are to  
119283 be provided, but the original, faster (but unsafe) functions (which may be implemented as  
119284 macros) are retained under new names. Some additional special-purpose synchronization  
119285 facilities are necessary for these macros to be usable in multi-threaded programs. This also  
119286 requires changes in **<stdio.h>**.

119287 The other common reason that functions are unsafe is that they return a pointer to static storage,  
119288 making the functions non-thread-safe. This has to be changed, and there are three natural  
119289 choices:

119290 1. Return a pointer to thread-specific storage

119291 This could incur a severe performance penalty on those architectures with a costly  
119292 implementation of the thread-specific data interface.

119293 A variation on this technique is to use *malloc()* to allocate storage for the function output  
119294 and return a pointer to this storage. This technique may also have an undesirable  
119295 performance impact, however, and a simplistic implementation requires that the user  
119296 program explicitly free the storage object when it is no longer needed. This technique is  
119297 used by some existing POSIX.1 functions. With careful implementation for infrequently

119298 used functions, there may be little or no performance or storage penalty, and the  
119299 maintenance of already-standardized interfaces is a significant benefit.

119300 2. Return the actual value computed by the function

119301 This technique can only be used with functions that return pointers to structures—  
119302 routines that return character strings would have to wrap their output in an enclosing  
119303 structure in order to return the output on the stack. There is also a negative performance  
119304 impact inherent in this solution in that the output value has to be copied twice before it  
119305 can be used by the calling function: once from the called routine's local buffers to the top  
119306 of the stack, then from the top of the stack to the assignment target. Finally, many older  
119307 compilers cannot support this technique due to a historical tendency to use internal static  
119308 buffers to deliver the results of structure-valued functions.

119309 3. Have the caller pass the address of a buffer to contain the computed value

119310 The only disadvantage of this approach is that extra arguments have to be provided by  
119311 the calling program. It represents the most efficient solution to the problem, however,  
119312 and, unlike the *malloc()* technique, it is semantically clear.

119313 There are some routines (often groups of related routines) whose interfaces are inherently non-  
119314 thread-safe because they communicate across multiple function invocations by means of static  
119315 memory locations. The solution is to redesign the calls so that they are thread-safe, typically by  
119316 passing the needed data as extra parameters. Unfortunately, this may require major changes to  
119317 the interface as well.

119318 A floating-point implementation using IEEE Std 754-1985 is a case in point. A less problematic  
119319 example is the *rand48* family of pseudo-random number generators. The functions *getrgid()*,  
119320 *getgrnam()*, *getpwnam()*, and *getpwuid()* are another such case.

119321 The problems with *errno* are discussed in [Alternative Solutions for Per-Thread \*errno\*](#) (on page  
119322 3405).

119323 Some functions can be thread-safe or not, depending on their arguments. These include the  
119324 *tmpnam()* and *ctermid()* functions. These functions have pointers to character strings as  
119325 arguments. If the pointers are not NULL, the functions store their results in the character string;  
119326 however, if the pointers are NULL, the functions store their results in an area that may be static  
119327 and thus subject to overwriting by successive calls. These should only be called by multi-thread  
119328 applications when their arguments are non-NULL.

119329 *Asynchronous Safety and Thread-Safety*

119330 A floating-point implementation has many modes that effect rounding and other aspects of  
119331 computation. Functions in some math library implementations may change the computation  
119332 modes for the duration of a function call. If such a function call is interrupted by a signal or  
119333 cancellation, the floating-point state is not required to be protected.

119334 There is a significant cost to make floating-point operations async-cancel-safe or async-signal-  
119335 safe; accordingly, neither form of async safety is required.

119336 *Functions Returning Pointers to Static Storage*

119337 For those functions that are not thread-safe because they return values in fixed size statically  
119338 allocated structures, alternate “\_r” forms are provided that pass a pointer to an explicit result  
119339 structure. Those that return pointers into library-allocated buffers have forms provided with  
119340 explicit buffer and length parameters.

119341 For functions that return pointers to library-allocated buffers, it makes sense to provide “\_r”  
119342 versions that allow the application control over allocation of the storage in which results are  
119343 returned. This allows the state used by these functions to be managed on an application-specific

119344 basis, supporting per-thread, per-process, or other application-specific sharing relationships.

119345 Early proposals had provided “\_r” versions for functions that returned pointers to variable-size  
 119346 buffers without providing a means for determining the required buffer size. This would have  
 119347 made using such functions exceedingly clumsy, potentially requiring iteratively calling them  
 119348 with increasingly larger guesses for the amount of storage required. Hence, *sysconf()* variables  
 119349 have been provided for such functions that return the maximum required buffer size.

119350 Thus, the rule that has been followed by POSIX.1-200x when adapting single-threaded non-  
 119351 thread-safe functions is as follows: all functions returning pointers to library-allocated storage  
 119352 should have “\_r” versions provided, allowing the application control over the storage  
 119353 allocation. Those with variable-sized return values accept both a buffer address and a length  
 119354 parameter. The *sysconf()* variables are provided to supply the appropriate buffer sizes when  
 119355 required. Implementors are encouraged to apply the same rule when adapting their own  
 119356 existing functions to a pthreads environment.

#### 119357 B.2.9.2 Thread IDs

119358 Separate applications should communicate through well-defined interfaces and should not  
 119359 depend on each other’s implementation. For example, if a programmer decides to rewrite the  
 119360 *sort* utility using multiple threads, it should be easy to do this so that the interface to the *sort*  
 119361 utility does not change. Consider that if the user causes SIGINT to be generated while the *sort*  
 119362 utility is running, keeping the same interface means that the entire *sort* utility is killed, not just  
 119363 one of its threads. As another example, consider a realtime application that manages a reactor.  
 119364 Such an application may wish to allow other applications to control the priority at which it  
 119365 watches the control rods. One technique to accomplish this is to write the ID of the thread  
 119366 watching the control rods into a file and allow other programs to change the priority of that  
 119367 thread as they see fit. A simpler technique is to have the reactor process accept IPCs  
 119368 (Interprocess Communication messages) from other processes, telling it at a semantic level what  
 119369 priority the program should assign to watching the control rods. This allows the programmer  
 119370 greater flexibility in the implementation. For example, the programmer can change the  
 119371 implementation from having one thread per rod to having one thread watching all of the rods  
 119372 without changing the interface. Having threads live inside the process means that the  
 119373 implementation of a process is invisible to outside processes (excepting debuggers and system  
 119374 management tools).

119375 Threads do not provide a protection boundary. Every thread model allows threads to share  
 119376 memory with other threads and encourages this sharing to be widespread. This means that one  
 119377 thread can wipe out memory that is needed for the correct functioning of other threads that are  
 119378 sharing its memory. Consequently, providing each thread with its own user and/or group IDs  
 119379 would not provide a protection boundary between threads sharing memory.

#### 119380 B.2.9.3 Thread Mutexes

119381 There is no additional rationale provided for this section.

#### 119382 B.2.9.4 Thread Scheduling

##### 119383 • Scheduling Implementation Models

119384 The following scheduling implementation models are presented in terms of threads and  
 119385 “kernel entities”. This is to simplify exposition of the models, and it does not imply that  
 119386 an implementation actually has an identifiable “kernel entity”.

119387 A kernel entity is not defined beyond the fact that it has scheduling attributes that are used



119388 to resolve contention with other kernel entities for execution resources. A kernel entity  
 119389 may be thought of as an envelope that holds a thread or a separate kernel thread. It is not a  
 119390 conventional process, although it shares with the process the attribute that it has a single  
 119391 thread of control; it does not necessarily imply an address space, open files, and so on. It is  
 119392 better thought of as a primitive facility upon which conventional processes and threads  
 119393 may be constructed.

119394 — System Thread Scheduling Model

119395 This model consists of one thread per kernel entity. The kernel entity is solely  
 119396 responsible for scheduling thread execution on one or more processors. This model  
 119397 schedules all threads against all other threads in the system using the scheduling  
 119398 attributes of the thread.

119399 — Process Scheduling Model

119400 A generalized process scheduling model consists of two levels of scheduling. A  
 119401 threads library creates a pool of kernel entities, as required, and schedules threads to  
 119402 run on them using the scheduling attributes of the threads. Typically, the size of the  
 119403 pool is a function of the simultaneously runnable threads, not the total number of  
 119404 threads. The kernel then schedules the kernel entities onto processors according to  
 119405 their scheduling attributes, which are managed by the threads library. This set model  
 119406 potentially allows a wide range of mappings between threads and kernel entities.

119407 • System and Process Scheduling Model Performance

119408 There are a number of important implications on the performance of applications using  
 119409 these scheduling models. The process scheduling model potentially provides lower  
 119410 overhead for making scheduling decisions, since there is no need to access kernel-level  
 119411 information or functions and the set of schedulable entities is smaller (only the threads  
 119412 within the process).

119413 On the other hand, since the kernel is also making scheduling decisions regarding the  
 119414 system resources under its control (for example, CPU(s), I/O devices, memory), decisions  
 119415 that do not take thread scheduling parameters into account can result in unspecified  
 119416 delays for realtime application threads, causing them to miss maximum response time  
 119417 limits.

119418 • Rate Monotonic Scheduling

119419 Rate monotonic scheduling was considered, but rejected for standardization in the context  
 119420 of pthreads. A sporadic server policy is included.

119421 • Scheduling Options

119422 In POSIX.1-200x, the basic thread scheduling functions are defined under the threads  
 119423 functionality, so that they are required of all threads implementations. However, there are  
 119424 no specific scheduling policies required by this functionality to allow for conforming  
 119425 thread implementations that are not targeted to realtime applications.

119426 Specific standard scheduling policies are defined to be under the Thread Execution  
 119427 Scheduling option, and they are specifically designed to support realtime applications by  
 119428 providing predictable resource-sharing sequences. The name of this option was chosen to  
 119429 emphasize that this functionality is defined as appropriate for realtime applications that  
 119430 require simple priority-based scheduling.

119431 It is recognized that these policies are not necessarily satisfactory for some multi-processor  
 119432 implementations, and work is ongoing to address a wider range of scheduling behaviors.  
 119433 The interfaces have been chosen to create abundant opportunity for future scheduling

119434 policies to be implemented and standardized based on this interface. In order to  
119435 standardize a new scheduling policy, all that is required (from the standpoint of thread  
119436 scheduling attributes) is to define a new policy name, new members of the thread  
119437 attributes object, and functions to set these members when the scheduling policy is equal  
119438 to the new value.

#### 119439 **Scheduling Contention Scope**

119440 In order to accommodate the requirement for realtime response, each thread has a scheduling  
119441 contention scope attribute. Threads with a system scheduling contention scope have to be  
119442 scheduled with respect to all other threads in the system. These threads are usually bound to a  
119443 single kernel entity that reflects their scheduling attributes and are directly scheduled by the  
119444 kernel.

119445 Threads with a process scheduling contention scope need be scheduled only with respect to the  
119446 other threads in the process. These threads may be scheduled within the process onto a pool of  
119447 kernel entities. The implementation is also free to bind these threads directly to kernel entities  
119448 and let them be scheduled by the kernel. Process scheduling contention scope allows the  
119449 implementation the most flexibility and is the default if both contention scopes are supported  
119450 and none is specified.

119451 Thus, the choice by implementors to provide one or the other (or both) of these scheduling  
119452 models is driven by the need of their supported application domains for worst-case (that is,  
119453 realtime) response, or average-case (non-realtime) response.

#### 119454 **Scheduling Allocation Domain**

119455 The SCHED\_FIFO and SCHED\_RR scheduling policies take on different characteristics on a  
119456 multi-processor. Other scheduling policies are also subject to changed behavior when executed  
119457 on a multi-processor. The concept of scheduling allocation domain determines the set of  
119458 processors on which the threads of an application may run. By considering the application's  
119459 processor scheduling allocation domain for its threads, scheduling policies can be defined in  
119460 terms of their behavior for varying processor scheduling allocation domain values. It is  
119461 conceivable that not all scheduling allocation domain sizes make sense for all scheduling  
119462 policies on all implementations. The concept of scheduling allocation domain, however, is a  
119463 useful tool for the description of multi-processor scheduling policies.

119464 The "process control" approach to scheduling obtains significant performance advantages from  
119465 dynamic scheduling allocation domain sizes when it is applicable.

119466 Non-Uniform Memory Access (NUMA) multi-processors may use a system scheduling structure  
119467 that involves reassignment of threads among scheduling allocation domains. In NUMA  
119468 machines, a natural model of scheduling is to match scheduling allocation domains to clusters of  
119469 processors. Load balancing in such an environment requires changing the scheduling allocation  
119470 domain to which a thread is assigned.

#### 119471 **Scheduling Documentation**

119472 Implementation-provided scheduling policies need to be completely documented in order to be  
119473 useful. This documentation includes a description of the attributes required for the policy, the  
119474 scheduling interaction of threads running under this policy and all other supported policies, and  
119475 the effects of all possible values for processor scheduling allocation domain. Note that for the  
119476 implementor wishing to be minimally-compliant, it is (minimally) acceptable to define the  
119477 behavior as undefined.

**119478 Scheduling Contention Scope Attribute**

119479 The scheduling contention scope defines how threads compete for resources. Within  
119480 POSIX.1-200x, scheduling contention scope is used to describe only how threads are scheduled  
119481 in relation to one another in the system. That is, either they are scheduled against all other  
119482 threads in the system (“system scope”) or only against those threads in the process (“process  
119483 scope”). In fact, scheduling contention scope may apply to additional resources, including  
119484 virtual timers and profiling, which are not currently considered by POSIX.1-200x.

**119485 Mixed Scopes**

119486 If only one scheduling contention scope is supported, the scheduling decision is straightforward.  
119487 To perform the processor scheduling decision in a mixed scope environment, it is necessary to  
119488 map the scheduling attributes of the thread with process-wide contention scope to the same  
119489 attribute space as the thread with system-wide contention scope.

119490 Since a conforming implementation has to support one and may support both scopes, it is useful  
119491 to discuss the effects of such choices with respect to example applications. If an implementation  
119492 supports both scopes, mixing scopes provides a means of better managing system-level (that is,  
119493 kernel-level) and library-level resources. In general, threads with system scope will require the  
119494 resources of a separate kernel entity in order to guarantee the scheduling semantics. On the  
119495 other hand, threads with process scope can share the resources of a kernel entity while  
119496 maintaining the scheduling semantics.

119497 The application is free to create threads with dedicated kernel resources, and other threads that  
119498 multiplex kernel resources. Consider the example of a window server. The server allocates two  
119499 threads per widget: one thread manages the widget user interface (including drawing), while  
119500 the other thread takes any required application action. This allows the widget to be “active”  
119501 while the application is computing. A screen image may be built from thousands of widgets. If  
119502 each of these threads had been created with system scope, then most of the kernel-level  
119503 resources might be wasted, since only a few widgets are active at any one time. In addition,  
119504 mixed scope is particularly useful in a window server where one thread with high priority and  
119505 system scope handles the mouse so that it tracks well. As another example, consider a database  
119506 server. For each of the hundreds or thousands of clients supported by a large server, an  
119507 equivalent number of threads will have to be created. If each of these threads were system scope,  
119508 the consequences would be the same as for the window server example above. However, the  
119509 server could be constructed so that actual retrieval of data is done by several dedicated threads.  
119510 Dedicated threads that do work for all clients frequently justify the added expense of system  
119511 scope. If it were not permissible to mix system and process threads in the same process, this type  
119512 of solution would not be possible.

**119513 Dynamic Thread Scheduling Parameters Access**

119514 In many time-constrained applications, there is no need to change the scheduling attributes  
119515 dynamically during thread or process execution, since the general use of these attributes is to  
119516 reflect directly the time constraints of the application. Since these time constraints are generally  
119517 imposed to meet higher-level system requirements, such as accuracy or availability, they  
119518 frequently should remain unchanged during application execution.

119519 However, there are important situations in which the scheduling attributes should be changed.  
119520 Generally, this will occur when external environmental conditions exist in which the time  
119521 constraints change. Consider, for example, a space vehicle major mode change, such as the  
119522 change from ascent to descent mode, or the change from the space environment to the  
119523 atmospheric environment. In such cases, the frequency with which many of the sensors or  
119524 actuators need to be read or written will change, which will necessitate a priority change. In  
119525 other cases, even the existence of a time constraint might be temporary, necessitating not just a

119526 priority change, but also a policy change for ongoing threads or processes. For this reason, it is  
 119527 critical that the interface should provide functions to change the scheduling parameters  
 119528 dynamically, but, as with many of the other realtime functions, it is important that applications  
 119529 use them properly to avoid the possibility of unnecessarily degrading performance.

119530 In providing functions for dynamically changing the scheduling behavior of threads, there were  
 119531 two options: provide functions to get and set the individual scheduling parameters of threads,  
 119532 or provide a single interface to get and set all the scheduling parameters for a given thread  
 119533 simultaneously. Both approaches have merit. Access functions for individual parameters allow  
 119534 simpler control of thread scheduling for simple thread scheduling parameters. However, a single  
 119535 function for setting all the parameters for a given scheduling policy is required when first setting  
 119536 that scheduling policy. Since the single all-encompassing functions are required, it was decided  
 119537 to leave the interface as minimal as possible. Note that simpler functions (such as  
 119538 *pthread\_setprio()* for threads running under the priority-based schedulers) can be easily defined  
 119539 in terms of the all-encompassing functions.

119540 If the *pthread\_setschedparam()* function executes successfully, it will have set all of the scheduling  
 119541 parameter values indicated in *param*; otherwise, none of the scheduling parameters will have  
 119542 been modified. This is necessary to ensure that the scheduling of this and all other threads  
 119543 continues to be consistent in the presence of an erroneous scheduling parameter.

119544 The [EPERM] error value is included in the list of possible *pthread\_setschedparam()* error returns  
 119545 as a reflection of the fact that the ability to change scheduling parameters increases risks to the  
 119546 implementation and application performance if the scheduling parameters are changed  
 119547 improperly. For this reason, and based on some existing practice, it was felt that some  
 119548 implementations would probably choose to define specific permissions for changing either a  
 119549 thread's own or another thread's scheduling parameters. POSIX.1-200x does not include  
 119550 portable methods for setting or retrieving permissions, so any such use of permissions is  
 119551 completely unspecified.

#### 119552 **Mutex Initialization Scheduling Attributes**

119553 In a priority-driven environment, a direct use of traditional primitives like mutexes and  
 119554 condition variables can lead to unbounded priority inversion, where a higher priority thread can  
 119555 be blocked by a lower priority thread, or set of threads, for an unbounded duration of time. As a  
 119556 result, it becomes impossible to guarantee thread deadlines. Priority inversion can be bounded  
 119557 and minimized by the use of priority inheritance protocols. This allows thread deadlines to be  
 119558 guaranteed even in the presence of synchronization requirements.

119559 Two useful but simple members of the family of priority inheritance protocols are the basic  
 119560 priority inheritance protocol and the priority ceiling protocol emulation. Under the Basic  
 119561 Priority Inheritance protocol (governed by the Non-Robust Mutex Priority Inheritance option), a  
 119562 thread that is blocking higher priority threads executes at the priority of the highest priority  
 119563 thread that it blocks. This simple mechanism allows priority inversion to be bounded by the  
 119564 duration of critical sections and makes timing analysis possible.

119565 Under the Priority Ceiling Protocol Emulation protocol (governed by the Thread Priority  
 119566 Protection option), each mutex has a priority ceiling, usually defined as the priority of the  
 119567 highest priority thread that can lock the mutex. When a thread is executing inside critical  
 119568 sections, its priority is unconditionally increased to the highest of the priority ceilings of all the  
 119569 mutexes owned by the thread. This protocol has two very desirable properties in uni-processor  
 119570 systems. First, a thread can be blocked by a lower priority thread for at most the duration of one  
 119571 single critical section. Furthermore, when the protocol is correctly used in a single processor, and  
 119572 if threads do not become blocked while owning mutexes, mutual deadlocks are prevented.

119573 The priority ceiling emulation can be extended to multiple processor environments, in which

119574 case the values of the priority ceilings will be assigned depending on the kind of mutex that is  
 119575 being used: local to only one processor, or global, shared by several processors. Local priority  
 119576 ceilings will be assigned the usual way, equal to the priority of the highest priority thread that  
 119577 may lock that mutex. Global priority ceilings will usually be assigned a priority level higher  
 119578 than all the priorities assigned to any of the threads that reside in the involved processors to  
 119579 avoid the effect called remote blocking.

#### 119580 **Change the Priority Ceiling of a Mutex**

119581 In order for the priority protect protocol to exhibit its desired properties of bounding priority  
 119582 inversion and avoidance of deadlock, it is critical that the ceiling priority of a mutex be the same  
 119583 as the priority of the highest thread that can ever hold it, or higher. Thus, if the priorities of the  
 119584 threads using such mutexes never change dynamically, there is no need ever to change the  
 119585 priority ceiling of a mutex.

119586 However, if a major system mode change results in an altered response time requirement for one  
 119587 or more application threads, their priority has to change to reflect it. It will occasionally be the  
 119588 case that the priority ceilings of mutexes held also need to change. While changing priority  
 119589 ceilings should generally be avoided, it is important that POSIX.1-200x provide these interfaces  
 119590 for those cases in which it is necessary.

#### 119591 **B.2.9.5 Thread Cancellation**

119592 Many existing threads packages have facilities for canceling an operation or canceling a thread.  
 119593 These facilities are used for implementing user requests (such as the CANCEL button in a  
 119594 window-based application), for implementing OR parallelism (for example, telling the other  
 119595 threads to stop working once one thread has found a forced mate in a parallel chess program), or  
 119596 for implementing the ABORT mechanism in Ada.

119597 POSIX programs traditionally have used the signal mechanism combined with either *longjmp()*  
 119598 or polling to cancel operations. Many POSIX programmers have trouble using these facilities to  
 119599 solve their problems efficiently in a single-threaded process. With the introduction of threads,  
 119600 these solutions become even more difficult to use.

119601 The main issues with implementing a cancellation facility are specifying the operation to be  
 119602 canceled, cleanly releasing any resources allocated to that operation, controlling when the target  
 119603 notices that it has been canceled, and defining the interaction between asynchronous signals and  
 119604 cancellation.

#### 119605 **Specifying the Operation to Cancel**

119606 Consider a thread that calls through five distinct levels of program abstraction and then, inside  
 119607 the lowest-level abstraction, calls a function that suspends the thread. (An abstraction boundary  
 119608 is a layer at which the client of the abstraction sees only the service being provided and can  
 119609 remain ignorant of the implementation. Abstractions are often layered, each level of abstraction  
 119610 being a client of the lower-level abstraction and implementing a higher-level abstraction.)  
 119611 Depending on the semantics of each abstraction, one could imagine wanting to cancel only the  
 119612 call that causes suspension, only the bottom two levels, or the operation being done by the entire  
 119613 thread. Canceling operations at a finer grain than the entire thread is difficult because threads  
 119614 are active and they may be run in parallel on a multi-processor. By the time one thread can make  
 119615 a request to cancel an operation, the thread performing the operation may have completed that  
 119616 operation and gone on to start another operation whose cancellation is not desired. Thread IDs  
 119617 are not reused until the thread has exited, and either it was created with the *Attr detachstate*  
 119618 attribute set to *PTHREAD\_CREATE\_DETACHED* or the *pthread\_join()* or *pthread\_detach()*  
 119619 function has been called for that thread. Consequently, a thread cancellation will never be  
 119620 misdirected when the thread terminates. For these reasons, the canceling of operations is done at

119621 the granularity of the thread. Threads are designed to be inexpensive enough so that a separate  
119622 thread may be created to perform each separately cancelable operation; for example, each  
119623 possibly long running user request.

119624 For cancellation to be used in existing code, cancellation scopes and handlers will have to be  
119625 established for code that needs to release resources upon cancellation, so that it follows the  
119626 programming discipline described in the text.

### 119627 **A Special Signal Versus a Special Interface**

119628 Two different mechanisms were considered for providing the cancellation interfaces. The first  
119629 was to provide an interface to direct signals at a thread and then to define a special signal that  
119630 had the required semantics. The other alternative was to use a special interface that delivered the  
119631 correct semantics to the target thread.

119632 The solution using signals produced a number of problems. It required the implementation to  
119633 provide cancellation in terms of signals whereas a perfectly valid (and possibly more efficient)  
119634 implementation could have both layered on a low-level set of primitives. There were so many  
119635 exceptions to the special signal (it cannot be used with *kill()*, no POSIX.1 interfaces can be used  
119636 with it) that it was clearly not a valid signal. Its semantics on delivery were also completely  
119637 different from any existing POSIX.1 signal. As such, a special interface that did not mandate the  
119638 implementation and did not confuse the semantics of signals and cancellation was felt to be the  
119639 better solution.

### 119640 **Races Between Cancellation and Resuming Execution**

119641 Due to the nature of cancellation, there is generally no synchronization between the thread  
119642 requesting the cancellation of a blocked thread and events that may cause that thread to resume  
119643 execution. For this reason, and because excess serialization hurts performance, when both an  
119644 event that a thread is waiting for has occurred and a cancellation request has been made and  
119645 cancellation is enabled, POSIX.1-200x explicitly allows the implementation to choose between  
119646 returning from the blocking call or acting on the cancellation request.

### 119647 **Interaction of Cancellation with Asynchronous Signals**

119648 A typical use of cancellation is to acquire a lock on some resource and to establish a cancellation  
119649 cleanup handler for releasing the resource when and if the thread is canceled.

119650 A correct and complete implementation of cancellation in the presence of asynchronous signals  
119651 requires considerable care. An implementation has to push a cancellation cleanup handler on the  
119652 cancellation cleanup stack while maintaining the integrity of the stack data structure. If an  
119653 asynchronously-generated signal is posted to the thread during a stack operation, the signal  
119654 handler cannot manipulate the cancellation cleanup stack. As a consequence, asynchronous  
119655 signal handlers may not cancel threads or otherwise manipulate the cancellation state of a  
119656 thread. Threads may, of course, be canceled by another thread that used a *sigwait()* function to  
119657 wait synchronously for an asynchronous signal.

119658 In order for cancellation to function correctly, it is required that asynchronous signal handlers  
119659 not change the cancellation state. This requires that some elements of existing practice, such as  
119660 using *longjmp()* to exit from an asynchronous signal handler implicitly, be prohibited in cases  
119661 where the integrity of the cancellation state of the interrupt thread cannot be ensured.

119662 **Thread Cancellation Overview**

## 119663 • Cancellability States

119664 The three possible cancellability states (disabled, deferred, and asynchronous) are encoded  
 119665 into two separate bits ((disable, enable) and (deferred, asynchronous)) to allow them to be  
 119666 changed and restored independently. For instance, short code sequences that will not block  
 119667 sometimes disable cancellability on entry and restore the previous state upon exit.  
 119668 Likewise, long or unbounded code sequences containing no convenient explicit  
 119669 cancellation points will sometimes set the cancellability type to asynchronous on entry and  
 119670 restore the previous value upon exit.

## 119671 • Cancellation Points

119672 Cancellation points are points inside of certain functions where a thread has to act on any  
 119673 pending cancellation request when cancellability is enabled. For functions in the “shall  
 119674 occur” list, a cancellation check must be performed on every call regardless of whether,  
 119675 absent the cancellation, the call would have blocked. For functions in the “may occur” list,  
 119676 a cancellation check may be performed on some calls but not others; i.e., whether or not a  
 119677 cancellation point occurs when one of these functions is being executed can depend on  
 119678 current conditions.

119679 The idea was considered of allowing implementations to define whether blocking calls  
 119680 such as *read()* should be cancellation points. It was decided that it would adversely affect  
 119681 the design of conforming applications if blocking calls were not cancellation points  
 119682 because threads could be left blocked in an uncancelable state.

119683 There are several important blocking routines that are specifically not made cancellation  
 119684 points:

119685 — *pthread\_mutex\_lock()*

119686 If *pthread\_mutex\_lock()* were a cancellation point, every routine that called it would  
 119687 also become a cancellation point (that is, any routine that touched shared state would  
 119688 automatically become a cancellation point). For example, *malloc()*, *free()*, and *rand()*  
 119689 would become cancellation points under this scheme. Having too many cancellation  
 119690 points makes programming very difficult, leading to either much disabling and  
 119691 restoring of cancellability or much difficulty in trying to arrange for reliable cleanup  
 119692 at every possible place.

119693 Since *pthread\_mutex\_lock()* is not a cancellation point, threads could result in being  
 119694 blocked uninterruptibly for long periods of time if mutexes were used as a general  
 119695 synchronization mechanism. As this is normally not acceptable, mutexes should only  
 119696 be used to protect resources that are held for small fixed lengths of time where not  
 119697 being able to be canceled will not be a problem. Resources that need to be held  
 119698 exclusively for long periods of time should be protected with condition variables.

119699 — *pthread\_barrier\_wait()*

119700 Canceling a barrier wait will render a barrier unusable. Similar to a barrier timeout  
 119701 (which the standard developers rejected), there is no way to guarantee the  
 119702 consistency of a barrier’s internal data structures if a barrier wait is canceled.

119703 — *pthread\_spin\_lock()*

119704 As with mutexes, spin locks should only be used to protect resources that are held for  
 119705 small fixed lengths of time where not being cancelable will not be a problem.

119706 Every library routine should specify whether or not it includes any cancellation points.  
 119707 Typically, only those routines that may block or compute indefinitely need to include

119708 cancellation points.

119709 Correctly coded routines only reach cancellation points after having set up a cancellation  
119710 cleanup handler to restore invariants if the thread is canceled at that point. Being  
119711 cancelable only at specified cancellation points allows programmers to keep track of  
119712 actions needed in a cancellation cleanup handler more easily. A thread should only be  
119713 made asynchronously cancelable when it is not in the process of acquiring or releasing  
119714 resources or otherwise in a state from which it would be difficult or impossible to recover.

119715 • Thread Cancellation Cleanup Handlers

119716 The cancellation cleanup handlers provide a portable mechanism, easy to implement, for  
119717 releasing resources and restoring invariants. They are easier to use than signal handlers  
119718 because they provide a stack of cancellation cleanup handlers rather than a single handler,  
119719 and because they have an argument that can be used to pass context information to the  
119720 handler.

119721 The alternative to providing these simple cancellation cleanup handlers (whose only use is  
119722 for cleaning up when a thread is canceled) is to define a general exception package that  
119723 could be used for handling and cleaning up after hardware traps and software-detected  
119724 errors. This was too far removed from the charter of providing threads to handle  
119725 asynchrony. However, it is an explicit goal of POSIX.1-200x to be compatible with existing  
119726 exception facilities and languages having exceptions.

119727 The interaction of this facility and other procedure-based or language-level exception  
119728 facilities is unspecified in this version of POSIX.1-200x. However, it is intended that it be  
119729 possible for an implementation to define the relationship between these cancellation  
119730 cleanup handlers and Ada, C++, or other language-level exception handling facilities.

119731 It was suggested that the cancellation cleanup handlers should also be called when the  
119732 process exits or calls the *exec* function. This was rejected partly due to the performance  
119733 problem caused by having to call the cancellation cleanup handlers of every thread before  
119734 the operation could continue. The other reason was that the only state expected to be  
119735 cleaned up by the cancellation cleanup handlers would be the intraprocess state. Any  
119736 handlers that are to clean up the interprocess state would be registered with *atexit()*. There  
119737 is the orthogonal problem that the *exec* functions do not honor the *atexit()* handlers, but  
119738 resolving this is beyond the scope of POSIX.1-200x.

119739 • Async-Cancel Safety

119740 A function is said to be async-cancel-safe if it is written in such a way that entering the  
119741 function with asynchronous cancelability enabled will not cause any invariants to be  
119742 violated, even if a cancellation request is delivered at any arbitrary instruction. Functions  
119743 that are async-cancel-safe are often written in such a way that they need to acquire no  
119744 resources for their operation and the visible variables that they may write are strictly  
119745 limited.

119746 Any routine that gets a resource as a side effect cannot be made async-cancel-safe (for  
119747 example, *malloc()*). If such a routine were called with asynchronous cancelability enabled,  
119748 it might acquire the resource successfully, but as it was returning to the client, it could act  
119749 on a cancellation request. In such a case, the application would have no way of knowing  
119750 whether the resource was acquired or not.

119751 Indeed, because many interesting routines cannot be made async-cancel-safe, most library  
119752 routines in general are not async-cancel-safe. Every library routine should specify whether  
119753 or not it is async-cancel safe so that programmers know which routines can be called from  
119754 code that is asynchronously cancelable.



119755 IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/8 is applied, adding the *pselect()* function  
119756 to the list of functions with cancellation points.

119757 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/5 is applied, adding the *fdatasync()*  
119758 function into the table of functions that shall have cancellation points.

119759 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/6 is applied, adding the numerous  
119760 functions into the table of functions that may have cancellation points.

119761 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/7 is applied, clarifying the requirements  
119762 in Thread Cancellation Cleanup Handlers.

#### 119763 B.2.9.6 Thread Read-Write Locks

##### 119764 Background

119765 Read-write locks are often used to allow parallel access to data on multi-processors, to avoid  
119766 context switches on uni-processors when multiple threads access the same data, and to protect  
119767 data structures that are frequently accessed (that is, read) but rarely updated (that is, written).  
119768 The in-core representation of a file system directory is a good example of such a data structure.  
119769 One would like to achieve as much concurrency as possible when searching directories, but limit  
119770 concurrent access when adding or deleting files.

119771 Although read-write locks can be implemented with mutexes and condition variables, such  
119772 implementations are significantly less efficient than is possible. Therefore, this synchronization  
119773 primitive is included in POSIX.1-200x for the purpose of allowing more efficient  
119774 implementations in multi-processor systems.

##### 119775 Queuing of Waiting Threads

119776 The *pthread\_rwlock\_unlock()* function description states that one writer or one or more readers  
119777 must acquire the lock if it is no longer held by any thread as a result of the call. However, the  
119778 function does not specify which thread(s) acquire the lock, unless the Thread Execution  
119779 Scheduling option is supported.

119780 The standard developers considered the issue of scheduling with respect to the queuing of  
119781 threads blocked on a read-write lock. The question turned out to be whether POSIX.1-200x  
119782 should require priority scheduling of read-write locks for threads whose execution scheduling  
119783 policy is priority-based (for example, SCHED\_FIFO or SCHED\_RR). There are tradeoffs  
119784 between priority scheduling, the amount of concurrency achievable among readers, and the  
119785 prevention of writer and/or reader starvation.

119786 For example, suppose one or more readers hold a read-write lock and the following threads  
119787 request the lock in the listed order:

```
119788 pthread_rwlock_wrlock() - Low priority thread writer_a
119789 pthread_rwlock_rdlock() - High priority thread reader_a
119790 pthread_rwlock_rdlock() - High priority thread reader_b
119791 pthread_rwlock_rdlock() - High priority thread reader_c
```

119792 When the lock becomes available, should *writer\_a* block the high priority readers? Or, suppose a  
119793 read-write lock becomes available and the following are queued:

```
119794 pthread_rwlock_rdlock() - Low priority thread reader_a
119795 pthread_rwlock_rdlock() - Low priority thread reader_b
119796 pthread_rwlock_rdlock() - Low priority thread reader_c
119797 pthread_rwlock_wrlock() - Medium priority thread writer_a
119798 pthread_rwlock_rdlock() - High priority thread reader_d
```

119799 If priority scheduling is applied then *reader\_d* would acquire the lock and *writer\_a* would block  
 119800 the remaining readers. But should the remaining readers also acquire the lock to increase  
 119801 concurrency? The solution adopted takes into account that when the Thread Execution  
 119802 Scheduling option is supported, high priority threads may in fact starve low priority threads  
 119803 (the application developer is responsible in this case for designing the system in such a way that  
 119804 this starvation is avoided). Therefore, POSIX.1-200x specifies that high priority readers take  
 119805 precedence over lower priority writers. However, to prevent writer starvation from threads of  
 119806 the same or lower priority, writers take precedence over readers of the same or lower priority.

119807 Priority inheritance mechanisms are non-trivial in the context of read-write locks. When a high  
 119808 priority writer is forced to wait for multiple readers, for example, it is not clear which subset of  
 119809 the readers should inherit the writer's priority. Furthermore, the internal data structures that  
 119810 record the inheritance must be accessible to all readers, and this implies some sort of  
 119811 serialization that could negate any gain in parallelism achieved through the use of multiple  
 119812 readers in the first place. Finally, existing practice does not support the use of priority  
 119813 inheritance for read-write locks. Therefore, no specification of priority inheritance or priority  
 119814 ceiling is attempted. If reliable priority-scheduled synchronization is absolutely required, it can  
 119815 always be obtained through the use of mutexes.

#### 119816 **Comparison to *fcntl()* Locks**

119817 The read-write locks and the *fcntl()* locks in POSIX.1-200x share a common goal: increasing  
 119818 concurrency among readers, thus increasing throughput and decreasing delay.

119819 However, the read-write locks have two features not present in the *fcntl()* locks. First, under  
 119820 priority scheduling, read-write locks are granted in priority order. Second, also under priority  
 119821 scheduling, writer starvation is prevented by giving writers preference over readers of equal or  
 119822 lower priority.

119823 Also, read-write locks can be used in systems lacking a file system, such as those conforming to  
 119824 the minimal realtime system profile of IEEE Std 1003.13-1998.

#### 119825 **History of Resolution Issues**

119826 Based upon some balloting objections, early drafts specified the behavior of threads waiting on a  
 119827 read-write lock during the execution of a signal handler, as if the thread had not called the lock  
 119828 operation. However, this specified behavior would require implementations to establish  
 119829 internal signal handlers even though this situation would be rare, or never happen for many  
 119830 programs. This would introduce an unacceptable performance hit in comparison to the little  
 119831 additional functionality gained. Therefore, the behavior of read-write locks and signals was  
 119832 reverted back to its previous mutex-like specification.

#### 119833 *B.2.9.7 Thread Interactions with Regular File Operations*

119834 There is no additional rationale provided for this section.

#### 119835 *B.2.9.8 Use of Application-Managed Thread Stacks*

119836 IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/8 is applied, adding this new section. It  
 119837 was added to make it clear that the current standard does not allow an application to determine  
 119838 when a stack can be reclaimed. This may be addressed in a future version.

119839 **B.2.10 Sockets**

119840 The base document for the sockets interfaces in POSIX.1-200x is the XNS, Issue 5.2 specification.  
 119841 This was primarily chosen as it aligns with IPv6. Additional material has been added from  
 119842 IEEE Std 1003.1g-2000, notably socket concepts, raw sockets, the *pselect()* function, the  
 119843 *socketmark()* function, and the `<sys/select.h>` header.

119844 *B.2.10.1 Address Families*

119845 There is no additional rationale provided for this section.

119846 *B.2.10.2 Addressing*

119847 There is no additional rationale provided for this section.

119848 *B.2.10.3 Protocols*

119849 There is no additional rationale provided for this section.

119850 *B.2.10.4 Routing*

119851 There is no additional rationale provided for this section.

119852 *B.2.10.5 Interfaces*

119853 There is no additional rationale provided for this section.

119854 *B.2.10.6 Socket Types*

119855 The type **socklen\_t** was invented to cover the range of implementations seen in the field. The  
 119856 intent of **socklen\_t** is to be the type for all lengths that are naturally bounded in size; that is, that  
 119857 they are the length of a buffer which cannot sensibly become of massive size: network addresses,  
 119858 host names, string representations of these, ancillary data, control messages, and socket options  
 119859 are examples. Truly boundless sizes are represented by **size\_t** as in *read()*, *write()*, and so on.

119860 All **socklen\_t** types were originally (in BSD UNIX) of type **int**. During the development of  
 119861 POSIX.1-200x, it was decided to change all buffer lengths to **size\_t**, which appears at face value  
 119862 to make sense. When dual mode 32/64-bit systems came along, this choice unnecessarily  
 119863 complicated system interfaces because **size\_t** (with **long**) was a different size under ILP32 and  
 119864 LP64 models. Reverting to **int** would have happened except that some implementations had  
 119865 already shipped 64-bit-only interfaces. The compromise was a type which could be defined to be  
 119866 any size by the implementation: **socklen\_t**.

119867 *B.2.10.7 Socket I/O Mode*

119868 There is no additional rationale provided for this section.

119869 *B.2.10.8 Socket Owner*

119870 There is no additional rationale provided for this section.

119871 *B.2.10.9 Socket Queue Limits*

119872 There is no additional rationale provided for this section.

119873 *B.2.10.10 Pending Error*

119874 There is no additional rationale provided for this section.

119875 *B.2.10.11 Socket Receive Queue*

119876 There is no additional rationale provided for this section.

119877 *B.2.10.12 Socket Out-of-Band Data State*

119878 There is no additional rationale provided for this section.

119879 *B.2.10.13 Connection Indication Queue*

119880 There is no additional rationale provided for this section.

119881 *B.2.10.14 Signals*

119882 There is no additional rationale provided for this section.

119883 *B.2.10.15 Asynchronous Errors*

119884 There is no additional rationale provided for this section.

119885 *B.2.10.16 Use of Options*

119886 There is no additional rationale provided for this section.

119887 *B.2.10.17 Use of Sockets for Local UNIX Connections*

119888 There is no additional rationale provided for this section.

119889 *B.2.10.18 Use of Sockets over Internet Protocols*

119890 A raw socket allows privileged users direct access to a protocol; for example, raw access to the  
 119891 IP and ICMP protocols is possible through raw sockets. Raw sockets are intended for  
 119892 knowledgeable applications that wish to take advantage of some protocol feature not directly  
 119893 accessible through the other sockets interfaces.

119894 *B.2.10.19 Use of Sockets over Internet Protocols Based on IPv4*

119895 There is no additional rationale provided for this section.

119896 *B.2.10.20 Use of Sockets over Internet Protocols Based on IPv6*

119897 The Open Group Base Resolution bwg2001-012 is applied, clarifying that IPv6 implementations  
 119898 are required to support use of AF\_INET6 sockets over IPv4.

119899 **B.2.11 Tracing**

119900 The organization of the tracing rationale differs from the traditional rationale in that this tracing  
 119901 rationale text is written against the trace interface as a whole, rather than against the individual  
 119902 components of the trace interface or the normative section in which those components are  
 119903 defined. Therefore the sections below do not parallel the sections of normative text in  
 119904 POSIX.1-200x.

## 119905 B.2.11.1 Objectives

119906 The intended uses of tracing are application-system debugging during system development, as a  
 119907 “flight recorder” for maintenance of fielded systems, and as a performance measurement tool.  
 119908 In all of these intended uses, the vendor-supplied computer system and its software are, for this  
 119909 discussion, assumed error-free; the intent being to debug the user-written and/or third-party  
 119910 application code, and their interactions. Clearly, problems with the vendor-supplied system and  
 119911 its software will be uncovered from time to time, but this is a byproduct of the primary activity,  
 119912 debugging user code.

119913 Another need for defining a trace interface in POSIX stems from the objective to provide an  
 119914 efficient portable way to perform benchmarks. Existing practice shows that such interfaces are  
 119915 commonly used in a variety of systems but with little commonality. As part of the benchmarking  
 119916 needs, two aspects within the trace interface must be considered.

119917 The first, and perhaps more important one, is the qualitative aspect.

119918 The second is the quantitative aspect.

## 119919 • Qualitative Aspect

119920 To better understand this aspect, let us consider an example. Suppose that you want to  
 119921 organize a number of actions to be performed during the day. Some of these actions are  
 119922 known at the beginning of the day. Some others, which may be more or less important,  
 119923 will be triggered by reading your mail. During the day you will make some phone calls  
 119924 and synchronously receive some more information. Finally you will receive asynchronous  
 119925 phone calls that also will trigger actions. If you, or somebody else, examines your day at  
 119926 work, you, or he, can discover that you have not efficiently organized your work. For  
 119927 instance, relative to the phone calls you made, would it be preferable to make some of  
 119928 these early in the morning? Or to delay some others until the end of the day? Relative to  
 119929 the phone calls you have received, you might find that somebody you called in the  
 119930 morning has called you 10 times while you were performing some important work. To  
 119931 examine, afterwards, your day at work, you record in sequence all the trace events relative  
 119932 to your work. This should give you a chance of organizing your next day at work.

119933 This is the qualitative aspect of the trace interface. The user of a system needs to keep a  
 119934 trace of particular points the application passes through, so that he can eventually make  
 119935 some changes in the application and/or system configuration, to give the application a  
 119936 chance of running more efficiently.

## 119937 • Quantitative Aspect

119938 This aspect concerns primarily realtime applications, where missed deadlines can be  
 119939 undesirable. Although there are, in POSIX.1-200x, some interfaces useful for such  
 119940 applications (timeouts, execution time monitoring, and so on), there are no APIs to aid in  
 119941 the tuning of a realtime application’s behavior (**timespec** in timeouts, length of message  
 119942 queues, duration of driver interrupt service routine, and so on). The tuning of an  
 119943 application needs a means of recording timestamped important trace events during  
 119944 execution in order to analyze offline, and eventually, to tune some realtime features  
 119945 (redesign the system with less functionalities, readjust timeouts, redesign driver interrupts,  
 119946 and so on).

119947 **Detailed Objectives**

119948 Objectives were defined to build the trace interface and are kept for historical interest. Although  
 119949 some objectives are not fully respected in this trace interface, the concept of the POSIX trace  
 119950 interface assumes the following points:

- 119951 1. It must be possible to trace both system and user trace events concurrently.
- 119952 2. It must be possible to trace per-process trace events and also to trace system trace events  
 119953 which are unrelated to any particular process. A per-process trace event is either user-  
 119954 initiated or system-initiated.
- 119955 3. It must be possible to control tracing on a per-process basis from either inside or outside  
 119956 the process.
- 119957 4. It must be possible to control tracing on a per-thread basis from inside the enclosing  
 119958 process.
- 119959 5. Trace points must be controllable by trace event type ID from inside and outside of the  
 119960 process. Multiple trace points can have the same trace event type ID, and will be  
 119961 controlled jointly.
- 119962 6. Recording of trace events is dependent on both trace event type ID and the  
 119963 process/thread. Both must be enabled in order to record trace events. System trace events  
 119964 may or may not be handled differently.
- 119965 7. The API must not mandate the ability to control tracing for more than one process at the  
 119966 same time.
- 119967 8. There is no objective for trace control on anything bigger than a process; for example,  
 119968 group or session.
- 119969 9. Trace propagation and control:
  - 119970 a. Trace propagation across *fork()* is optional; the default is to not trace a child  
 119971 process.
  - 119972 b. Trace control must span *pthread\_create()* operations; that is, if a process is being  
 119973 traced, any thread will be traced as well if this thread allows tracing. The default is  
 119974 to allow tracing.
- 119975 10. Trace control must not span *exec* or *posix\_spawn()* operations.
- 119976 11. A triggering API is not required. The triggering API is the ability to command or stop  
 119977 tracing based on the occurrence of a specific trace event other than a  
 119978 POSIX\_TRACE\_START trace event or a POSIX\_TRACE\_STOP trace event.
- 119979 12. Trace log entries must have timestamps of implementation-defined resolution.  
 119980 Implementations are exhorted to support at least microsecond resolution. When a trace  
 119981 log entry is retrieved, it must have timestamp, PC address, PID, and TID of the entity that  
 119982 generated the trace event.
- 119983 13. Independently developed code should be able to use trace facilities without coordination  
 119984 and without conflict.
- 119985 14. Even if the trace points in the trace calls are not unique, the trace log entries (after any  
 119986 processing) must be uniquely identified as to trace point.
- 119987 15. There must be a standard API to read the trace stream.

- 119988 16. The format of the trace stream and the trace log is opaque and unspecified.
- 119989 17. It must be possible to read a completed trace, if recorded on some suitable non-volatile  
119990 storage, even subsequent to a power cycle or subsequent cold boot of the system.
- 119991 18. Support of analysis of a trace log while it is being formed is implementation-defined.
- 119992 19. The API must allow the application to write trace stream identification information into  
119993 the trace stream and to be able to retrieve it, without it being overwritten by trace entries,  
119994 even if the trace stream is full.
- 119995 20. It must be possible to specify the destination of trace data produced by trace events.
- 119996 21. It must be possible to have different trace streams, and for the tracing enabled by one  
119997 trace stream to be completely independent of the tracing of another trace stream.
- 119998 22. It must be possible to trace events from threads in different CPUs.
- 119999 23. The API must support one or more trace streams per-system, and one or more trace  
120000 streams per-process, up to an implementation-defined set of per-system and per-process  
120001 maximums.
- 120002 24. It must be possible to determine the order in which the trace events happened, without  
120003 necessarily depending on the clock, up to an implementation-defined time resolution.
- 120004 25. For performance reasons, the trace event point call(s) must be implementable as a macro  
120005 (see the ISO POSIX-1: 1996 standard, 1.3.4, Statement 2).
- 120006 26. POSIX.1-200x must not define the trace points which a conforming system must  
120007 implement, except for trace points used in the control of tracing.
- 120008 27. The APIs must be thread-safe, and trace points should be lock-free (that is, not require a  
120009 lock to gain exclusive access to some resource).
- 120010 28. The user-provided information associated with a trace event is variable-sized, up to some  
120011 maximum size.
- 120012 29. Bounds on record and trace stream sizes:
- 120013 a. The API must permit the application to declare the upper bounds on the length of  
120014 an application data record. The system must return the limit it used. The limit used  
120015 may be smaller than requested.
- 120016 b. The API must permit the application to declare the upper bounds on the size of  
120017 trace streams. The system must return the limit it used. The limit used may be  
120018 different, either larger or smaller, than requested.
- 120019 30. The API must be able to pass any fundamental data type, and a structured data type  
120020 composed only of fundamental types. The API must be able to pass data by reference,  
120021 given only as an address and a length. Fundamental types are the POSIX.1 types (see the  
120022 **<sys/types.h>** header) plus those defined in the ISO C standard.
- 120023 31. The API must apply the POSIX notions of ownership and permission to recorded trace  
120024 data, corresponding to the sources of that data.

120025 **Comments on Objectives**

120026 **Note:** In the following comments, numbers in square brackets refer to the above objectives.

120027 It is necessary to be able to obtain a trace stream for a complete activity. Thus there is a  
 120028 requirement to be able to trace both application and system trace events. A per-process trace  
 120029 event is either user-initiated, like the *write()* function, or system-initiated, like a timer expiration.  
 120030 There is also a need to be able to trace the activity of an entire process even when it has threads  
 120031 in multiple CPUs. To avoid excess trace activity, it is necessary to be able to control tracing on a  
 120032 trace event type basis.

120033 [Objectives 1,2,5,22]

120034 There is a need to be able to control tracing on a per-process basis, both from inside and outside  
 120035 the process; that is, a process can start a trace activity on itself or any other process. There is also  
 120036 the perceived need to allow the definition of a maximum number of trace streams per system.

120037 [Objectives 3,23]

120038 From within a process, it is necessary to be able to control tracing on a per-thread basis. This  
 120039 provides an additional filtering capability to keep the amount of traced data to a minimum. It  
 120040 also allows for less ambiguity as to the origin of trace events. It is recognized that thread-level  
 120041 control is only valid from within the process itself. It is also desirable to know the maximum  
 120042 number of trace streams per process that can be started. The API should not require thread  
 120043 synchronization or mandate priority inversions that would cause the thread to block. However,  
 120044 the API must be thread-safe.

120045 [Objectives 4,23,24,27]

120046 There was no perceived objective to control tracing on anything larger than a process; for  
 120047 example, a group or session. Also, the ability to start or stop a trace activity on multiple  
 120048 processes atomically may be very difficult or cumbersome in some implementations.

120049 [Objectives 6,8]

120050 It is also necessary to be able to control tracing by trace event type identifier, sometimes called a  
 120051 trace hook ID. However, there is no mandated set of system trace events, since such trace points  
 120052 are implementation-defined. The API must not require from the operating system facilities that  
 120053 are not standard.

120054 [Objectives 6,26]

120055 Trace control must span *fork()* and *pthread\_create()*. If not, there will be no way to ensure that an  
 120056 application's activity is entirely traced. The newly forked child would not be able to turn on its  
 120057 tracing until after it obtained control after the fork, and trace control externally would be even  
 120058 more problematic.

120059 [Objective 9]

120060 Since *exec* and *posix\_spawn()* represent a complete change in the execution of a task (a new  
 120061 program), trace control need not persist over an *exec* or *posix\_spawn()*.

120062 [Objective 10]

120063 Where trace activities are started on multiple processes, these trace activities should not interfere  
 120064 with each other.

120065 [Objective 21]

120066 There is no need for a triggering objective, primarily for performance reasons; see also [Section](#)  
 120067 [B.2.11.8](#) (on page 3516), rationale on triggering.

120068 [Objective 11]

120069 It must be possible to determine the origin of each traced event. The process and thread  
 120070 identifiers for each trace event are needed. Also there was a perceived need for a user-specifiable  
 120071 origin, but it was felt that this would create too much overhead.

120072 [Objectives 12,14]



- 120073 An allowance must be made for trace points to come embedded in software components from  
120074 several different sources and vendors without requiring coordination.  
120075 [Objective 13]
- 120076 There is a requirement to be able to uniquely identify trace points that may have the same trace  
120077 stream identifier. This is only necessary when a trace report is produced.  
120078 [Objectives 12,14]
- 120079 Tracing is a very performance-sensitive activity, and will therefore likely be implemented at a  
120080 low level within the system. Hence the interface must not mandate any particular buffering or  
120081 storage method. Therefore, a standard API is needed to read a trace stream. Also the interface  
120082 must not mandate the format of the trace data, and the interface must not assume a trace storage  
120083 method. Due to the possibility of a monolithic kernel and the possible presence of multiple  
120084 processes capable of running trace activities, the two kinds of trace events may be stored in two  
120085 separate streams for performance reasons. A mandatory dump mechanism, common in some  
120086 existing practice, has been avoided to allow the implementation of this set of functions on small  
120087 realtime profiles for which the concept of a file system is not defined. The trace API calls should  
120088 be implemented as macros.  
120089 [Objectives 15,16,25,30]
- 120090 Since a trace facility is a valuable service tool, the output (or log) of a completed trace stream  
120091 that is written to permanent storage must be readable on other systems of the type that  
120092 produced the trace log. Note that there is no objective to be able to interpret a trace log that was  
120093 not successfully completed.  
120094 [Objectives 17,18,19]
- 120095 For trace streams written to permanent storage, a way to specify the destination of the trace  
120096 stream is needed.  
120097 [Objective 20]
- 120098 There is a requirement to be able to depend on the ordering of trace events up to some  
120099 implementation-defined time interval. For example, there is a need to know the time period  
120100 during which, if trace events are closer together, their ordering is unspecified. Events that occur  
120101 within an interval smaller than this resolution may or may not be read back in the correct order.  
120102 [Objective 24]
- 120103 The application should be able to know how much data can be traced. When trace event types  
120104 can be filtered, the application should be able to specify the approximate maximum amount of  
120105 data that will be traced in a trace event so resources can be more efficiently allocated.  
120106 [Objectives 28,29]
- 120107 Users should not be able to trace data to which they would not normally have access. System  
120108 trace events corresponding to a process/thread should be associated with the ownership of that  
120109 process/thread.  
120110 [Objective 31]

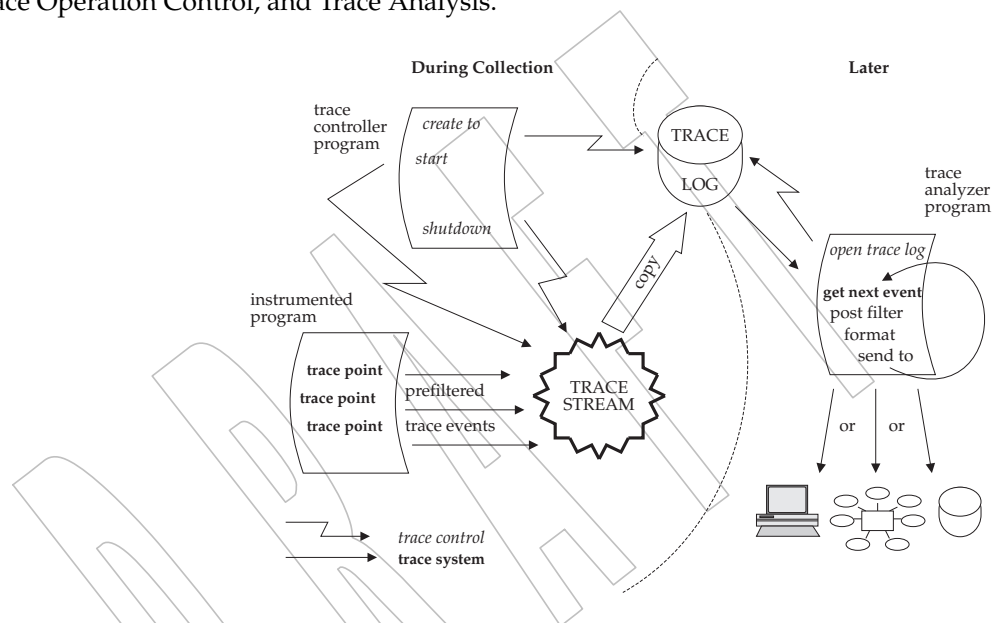
## 120111 B.2.11.2 Trace Model

120112 **Introduction**

120113 The model is based on two base entities: the “Trace Stream” and the “Trace Log”, and a recorded  
 120114 unit called the “Trace Event”. The possibility of using Trace Streams and Trace Logs separately  
 120115 gives two use dimensions and solves both the performance issue and the full-information  
 120116 system issue. In the case of a trace stream without log, specific information, although reduced in  
 120117 quantity, is required to be registered, in a possibly small realtime system, with as little overhead  
 120118 as possible. The Trace Log option has been added for small realtime systems. In the case of a  
 120119 trace stream with log, considerable complex application-specific information needs to be  
 120120 collected.

120121 **Trace Model Description**

120122 The trace model can be examined for three different subfunctions: Application Instrumentation,  
 120123 Trace Operation Control, and Trace Analysis.



120124 **Figure B-2** Trace System Overview: for Offline Analysis

120125 Each of these subfunctions requires specific characteristics of the trace mechanism API.

- 120126 • Application Instrumentation

120127 When instrumenting an application, the programmer is not concerned about the future use  
 120128 of the trace events in the trace stream or the trace log, the full policy of the trace stream, or  
 120129 the eventual pre-filtering of trace events. But he is concerned about the correct  
 120130 determination of the specific trace event type identifier, regardless of how many  
 120131 independent libraries are used in the same user application; see [Figure B-2](#) and [Figure B-3](#)  
 120132 (on page 3499).

120133 This trace API provides the necessary operations to accomplish this subfunction. This is  
 120134 done by providing functions to associate a programmer-defined name with an  
 120135 implementation-defined trace event type identifier (see the `posix_trace_eventid_open()`  
 120136 function), and to send this trace event into a potential trace stream (see the  
 120137 `posix_trace_event()` function).

120138

- Trace Operation Control

120139

120140

120141

120142

When controlling the recording of trace events in a trace stream, the programmer is concerned with the correct initialization of the trace mechanism (that is, the sizing of the trace stream), the correct retention of trace events in a permanent storage, the correct dynamic recording of trace events, and so on.

120143

120144

This trace API provides the necessary material to permit this efficiently. This is done by providing functions to initialize a new trace stream, and optionally a trace log:

120145

- Trace Stream Attributes Object Initialization (see *posix\_trace\_attr\_init()*)

120146

120147

- Functions to Retrieve or Set Information About a Trace Stream (see *posix\_trace\_attr\_getgenversion()*)

120148

120149

- Functions to Retrieve or Set the Behavior of a Trace Stream (see *posix\_trace\_attr\_getinherited()*)

120150

120151

- Functions to Retrieve or Set Trace Stream Size Attributes (see *posix\_trace\_attr\_getmaxusereventsize()*)

120152

120153

- Trace Stream Initialization, Flush, and Shutdown from a Process (see *posix\_trace\_create()*)

120154

- Clear Trace Stream and Trace Log (see *posix\_trace\_clear()*)

120155

To select the trace event types that are to be traced:

120156

- Manipulate Trace Event Type Identifier (see *posix\_trace\_trid\_eventid\_open()*)

120157

- Iterate over a Mapping of Trace Event Type (see *posix\_trace\_eventtypelist\_getnext\_id()*)

120158

- Manipulate Trace Event Type Sets (see *posix\_trace\_eventset\_empty()*)

120159

- Set Filter of an Initialized Trace Stream (see *posix\_trace\_set\_filter()*)

120160

To control the execution of an active trace stream:

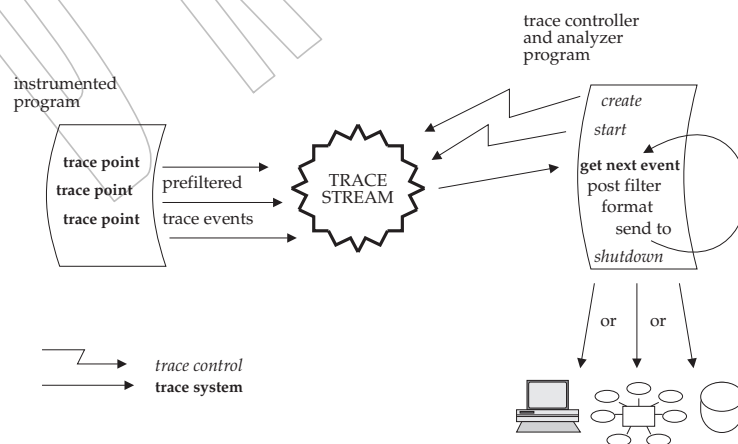
120161

- Trace Start and Stop (see *posix\_trace\_start()*)

120162

120163

- Functions to Retrieve the Trace Attributes or Trace Statuses (see *posix\_trace\_get\_attr()*)



120164

**Figure B-3** Trace System Overview: for Online Analysis

120165  
120166  
120167  
120168  
120169  
120170  
120171  
120172  
120173  
120174  
120175  
120176  
120177  
120178  
120179  
120180  
120181  
120182  
120183  
120184  
120185  
120186  
120187  
120188  
120189  
120190  
120191  
120192  
120193  
120194  
120195  
120196  
120197  
120198  
120199  
120200

- Trace Analysis

Once correctly recorded, on permanent storage or not, an ultimate activity consists of the analysis of the recorded information. If the recorded data is on permanent storage, a specific open operation is required to associate a trace stream to a trace log.

The first intent of the group was to request the presence of a system identification structure in the trace stream attribute. This was, for the application, to allow some portable way to process the recorded information. However, there is no requirement that the **utsname** structure, on which this system identification was based, be portable from one machine to another, so the contents of the attribute cannot be interpreted correctly by an application conforming to POSIX.1-200x.

This modification has been incorporated and requests that some unspecified information be recorded in the trace log in order to fail opening it if the analysis process and the controller process were running in different types of machine, but does not request that this information be accessible to the application. This modification has implied a modification in the *posix\_trace\_open()* function error code returns.

This trace API provides functions to:

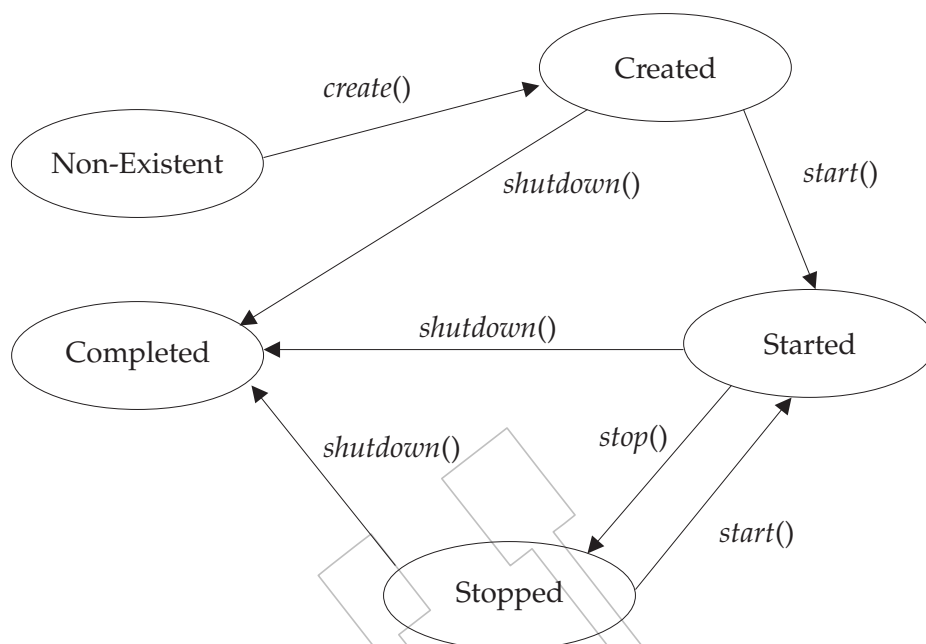
- Extract trace stream identification attributes (see *posix\_trace\_attr\_getgenversion()*)
- Extract trace stream behavior attributes (see *posix\_trace\_attr\_getinherited()*)
- Extract trace event, stream, and log size attributes (see *posix\_trace\_attr\_getmaxusereventsized()*)
- Look up trace event type names (see *posix\_trace\_eventid\_get\_name()*)
- Iterate over trace event type identifiers (see *posix\_trace\_eventtypelist\_getnext\_id()*)
- Open, rewind, and close a trace log (see *posix\_trace\_open()*)
- Read trace stream attributes and status (see *posix\_trace\_get\_attr()*)
- Read trace events (see *posix\_trace\_getnext\_event()*)

Due to the following two reasons:

1. The requirement that the trace system must not add unacceptable overhead to the traced process and so that the trace event point execution must be fast
2. The traced application does not care about tracing errors

the trace system cannot return any internal error to the application. Internal error conditions can range from unrecoverable errors that will force the active trace stream to abort, to small errors that can affect the quality of tracing without aborting the trace stream. The group decided to define a system trace event to report to the analysis process such internal errors. It is not the intention of POSIX.1-200x to require an implementation to report an internal error that corrupts or terminates tracing operation. The implementor is free to decide which internal documented errors, if any, the trace system is able to report.

120201

**States of a Trace Stream**

120202

**Figure B-4** Trace System Overview: States of a Trace Stream

120203

120204

120205

120206

120207

120208

120209

120210

120211

**Figure B-4** shows the different states an active trace stream passes through. After the `posix_trace_create()` function call, a trace stream becomes `CREATED` and a trace stream is associated for the future collection of trace events. The status of the trace stream is `POSIX_TRACE_SUSPENDED`. The state becomes `STARTED` after a call to the `posix_trace_start()` function, and the status becomes `POSIX_TRACE_RUNNING`. In this state, all trace events that are not filtered out will be stored into the trace stream. After a call to `posix_trace_stop()`, the trace stream becomes `STOPPED` (and the status `POSIX_TRACE_SUSPENDED`). In this state, no new trace events will be recorded in the trace stream, but previously recorded trace events may continue to be read.

120212

120213

120214

120215

120216

After a call to `posix_trace_shutdown()`, the trace stream is in the state `COMPLETED`. The trace stream no longer exists but, if the Trace Log option is supported, all the information contained in it has been logged. If a log object has not been associated with the trace stream at the creation, it is the responsibility of the trace controller process to not shut the trace stream down while trace events remain to be read in the stream.

120217

**Tracing All Processes**

120218

120219

120220

120221

Some implementations have a tracing subsystem with the ability to trace all processes. This is useful to debug some types of device drivers such as those for ATM or X25 adapters. These types of adapters are used by several independent processes, that are not issued from the same process.

120222

120223

120224

120225

The POSIX trace interface does not define any constant or option to create a trace stream tracing all processes. POSIX.1 does not prevent this type of implementation and an implementor is free to add this capability. Nevertheless, the trace interface allows tracing of all the system trace events and all the processes issued from the same process.

120226 If such a tracing system capability has to be implemented, when a trace stream is created, it is  
 120227 recommended that a constant named `POSIX_TRACE_ALLPROC` be used instead of the process  
 120228 identifier in the argument of the `posix_trace_create()` or `posix_trace_create_withlog()` function. A  
 120229 possible value for `POSIX_TRACE_ALLPROC` may be `-1` instead of a real process identifier.

120230 The implementor has to be aware that there is some impact on the tracing behavior as defined in  
 120231 the POSIX trace interface. For example:

- 120232 • If the default value for the inheritance attribute is set to  
 120233 `POSIX_TRACE_CLOSE_FOR_CHILD`, the implementation has to stop tracing for the child  
 120234 process.
- 120235 • The trace controller which is creating this type of trace stream must have the appropriate  
 120236 privilege to trace all the processes.

### 120237 Trace Storage

120238 The model is based on two types of trace events: system trace events and user-defined trace  
 120239 events. The internal representation of trace events is implementation-defined, and so the  
 120240 implementor is free to choose the more suitable, practical, and efficient way to design the  
 120241 internal management of trace events. For the timestamping operation, the model does not  
 120242 impose the `CLOCK_REALTIME` or any other clock. The buffering allocation and operation  
 120243 follow the same principle. The implementor is free to use one or more buffers to record trace  
 120244 events; the interface assumes only a logical trace stream of sequentially recorded trace events.  
 120245 Regarding flushing of trace events, the interface allows the definition of a trace log object which  
 120246 typically can be a file. But the group was also aware of defining functions to permit the use of  
 120247 this interface in small realtime systems, which may not have general file system capabilities. For  
 120248 instance, the three functions `posix_trace_getnext_event()` (blocking),  
 120249 `posix_trace_timedgetnext_event()` (blocking with timeout), and `posix_trace_trygetnext_event()` (non-  
 120250 blocking) are proposed to read the recorded trace events.

120251 The policy to be used when the trace stream becomes full also relies on common practice:

- 120252 • For an active trace stream, the `POSIX_TRACE_LOOP` trace stream policy permits  
 120253 automatic overrun (overwrite of oldest trace events) while waiting for some user-defined  
 120254 condition to cause tracing to stop. By contrast, the `POSIX_TRACE_UNTIL_FULL` trace  
 120255 stream policy requires the system to stop tracing when the trace stream is full. However, if  
 120256 the trace stream that is full is at least partially emptied by a call to the `posix_trace_flush()`  
 120257 function or by calls to the `posix_trace_getnext_event()` function, the trace system will  
 120258 automatically resume tracing.

120259 If the Trace Log option is supported, the operation of the `POSIX_TRACE_FLUSH` policy is  
 120260 an extension of the `POSIX_TRACE_UNTIL_FULL` policy. The automatic free operation (by  
 120261 flushing to the associated trace log) is added.

- 120262 • If a log is associated with the trace stream and this log is a regular file, these policies also  
 120263 apply for the log. One more policy, `POSIX_TRACE_APPEND`, is defined to allow  
 120264 indefinite extension of the log. Since the log destination can be any device or pseudo-  
 120265 device, the implementation may not be able to manipulate the destination as required by  
 120266 POSIX.1-200x. For this reason, the behavior of the log full policy may be unspecified  
 120267 depending on the trace log type.

120268 The current trace interface does not define a service to preallocate space for a trace log file,  
 120269 because this space can be preallocated by means of a call to the `posix_fallocate()` function.  
 120270 This function could be called after the file has been opened, but before the trace stream is  
 120271 created. The `posix_fallocate()` function ensures that any required storage for regular file data  
 120272 is allocated on the file system storage media. If `posix_fallocate()` returns successfully,

120273 subsequent writes to the specified file data will not fail due to the lack of free space on the  
 120274 file system storage media. Besides trace events, a trace stream also includes trace attributes  
 120275 and the mapping from trace event names to trace event type identifiers. The implementor  
 120276 is free to choose how to store the trace attributes and the trace event type map, but must  
 120277 ensure that this information is not lost when a trace stream overrun occurs.

### 120278 B.2.11.3 Trace Programming Examples

120279 Several programming examples are presented to show the code of the different possible  
 120280 subfunctions using a trace subsystem. All these programs need to include the `<trace.h>` header.  
 120281 In the examples shown, error checking is omitted for more simplicity.

### 120282 Trace Operation Control

120283 These examples show the creation of a trace stream for another process; one which is already  
 120284 trace instrumented. All the default trace stream attributes are used to simplify programming in  
 120285 the first example. The second example shows more possibilities.

#### 120286 First Example

```

120287 /* Caution. Error checks omitted */
120288 {
120289 trace_attr_t attr;
120290 pid_t pid = traced_process_pid;
120291 int fd;
120292 trace_id_t trid;
120293
120294 - - - - -
120295 /* Initialize trace stream attributes */
120296 posix_trace_attr_init(&attr);
120297 /* Open a trace log */
120298 fd=open("/tmp/mytracelog",...);
120299 /*
120300 * Create a new trace associated with a log
120301 * and with default attributes
120302 */
120303 posix_trace_create_withlog(pid, &attr, fd, &trid);
120304 /* Trace attribute structure can now be destroyed */
120305 posix_trace_attr_destroy(&attr);
120306 /* Start of trace event recording */
120307 posix_trace_start(trid);
120308 - - - - -
120309 /* Duration of tracing */
120310 - - - - -
120311 - - - - -
120312 /* Stop and shutdown of trace activity */
120313 posix_trace_shutdown(trid);
120314 - - - - -
120315 }

```

120316 **Second Example**

120317 Between the initialization of the trace stream attributes and the creation of the trace stream, these  
 120318 trace stream attributes may be modified; see [Trace Stream Attribute Manipulation](#) (on page  
 120319 3507) for a specific programming example. Between the creation and the start of the trace  
 120320 stream, the event filter may be set; after the trace stream is started, the event filter may be  
 120321 changed. The setting of an event set and the change of a filter is shown in [Create a Trace Event  
 120322 Type Set and Change the Trace Event Type Filter](#) (on page 3508).

```

120323 /* Caution. Error checks omitted */
120324 {
120325 trace_attr_t attr;
120326 pid_t pid = traced_process_pid;
120327 int fd;
120328 trace_id_t trid;
120329 - - - - -
120330 /* Initialize trace stream attributes */
120331 posix_trace_attr_init(&attr);
120332 /* Attr default may be changed at this place; see example */
120333 - - - - -
120334 /* Create and open a trace log with R/W user access */
120335 fd=open("/tmp/mytracelog",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
120336 /* Create a new trace associated with a log */
120337 posix_trace_create_withlog(pid, &attr, fd, &trid);
120338 /*
120339 * If the Trace Filter option is supported
120340 * trace event type filter default may be changed at this place;
120341 * see example about changing the trace event type filter
120342 */
120343 posix_trace_start(trid);
120344 - - - - -
120345 /*
120346 * If you have an uninteresting part of the application
120347 * you can stop temporarily.
120348 *
120349 * posix_trace_stop(trid);
120350 * - - - - -
120351 * - - - - -
120352 * posix_trace_start(trid);
120353 */
120354 - - - - -
120355 /*
120356 * If the Trace Filter option is supported
120357 * the current trace event type filter can be changed
120358 * at any time (see example about how to set
120359 * a trace event type filter)
120360 */
120361 - - - - -
120362 /* Stop the recording of trace events */
120363 posix_trace_stop(trid);
120364 /* Shutdown the trace stream */
120365 posix_trace_shutdown(trid);
120366 /*

```



```

120367 * Destroy trace stream attributes; attr structure may have
120368 * been used during tracing to fetch the attributes
120369 */
120370 posix_trace_attr_destroy(&attr);
120371 - - - - -
120372 }

```

### 120373 Application Instrumentation

120374 This example shows an instrumented application. The code is included in a block of instructions,  
 120375 perhaps a function from a library. Possibly in an initialization part of the instrumented  
 120376 application, two user trace event names are mapped to two trace event type identifiers  
 120377 (function `posix_trace_eventid_open()`). Then two trace points are programmed.

```

120378 /* Caution. Error checks omitted */
120379 {
120380 trace_event_id_t eventid1, eventid2;
120381 - - - - -
120382 /* Initialization of two trace event type ids */
120383 posix_trace_eventid_open("my_first_event",&eventid1);
120384 posix_trace_eventid_open("my_second_event",&eventid2);
120385 - - - - -
120386 - - - - -
120387 - - - - -
120388 /* Trace point */
120389 posix_trace_event(eventid1,NULL,0);
120390 - - - - -
120391 /* Trace point */
120392 posix_trace_event(eventid2,NULL,0);
120393 - - - - -
120394 }

```

### 120395 Trace Analyzer

120396 This example shows the manipulation of a trace log resulting from the dumping of a completed  
 120397 trace stream. All the default attributes are used to simplify programming, and data associated  
 120398 with a trace event is not shown in the first example. The second example shows more  
 120399 possibilities.

#### 120400 First Example

```

120401 /* Caution. Error checks omitted */
120402 {
120403 int fd;
120404 trace_id_t trid;
120405 posix_trace_event_info trace_event;
120406 char trace_event_name[TRACE_EVENT_NAME_MAX];
120407 int return_value;
120408 size_t returndatasize;
120409 int lost_event_number;
120410 - - - - -
120411 /* Open an existing trace log */
120412 fd=open("/tmp/tracelog", O_RDONLY);

```

```

120413 /* Open a trace stream on the open log */
120414 posix_trace_open(fd, &trid);
120415 /* Read a trace event */
120416 posix_trace_getnext_event(trid, &trace_event,
120417 NULL, 0, &returndatasize,&return_value);

120418 /* Read and print all trace event names out in a loop */
120419 while (return_value == NULL)
120420 {
120421 /*
120422 * Get the name of the trace event associated
120423 * with trid trace ID
120424 */
120425 posix_trace_eventid_get_name(trid, trace_event.event_id,
120426 trace_event_name);
120427 /* Print the trace event name out */
120428 printf("%s\n",trace_event_name);
120429 /* Read a trace event */
120430 posix_trace_getnext_event(trid, &trace_event,
120431 NULL, 0, &returndatasize,&return_value);
120432 }

120433 /* Close the trace stream */
120434 posix_trace_close(trid);
120435 /* Close the trace log */
120436 close(fd);
120437 }

```

### 120438 Second Example

120439 The complete example includes the two other examples in [Retrieve Information from a Trace Log](#) (on page 3509) and in [Retrieve the List of Trace Event Types Used in a Trace Log](#) (on page 3510). For example, the *maxdatasize* variable is set in [Retrieve the List of Trace Event Types Used in a Trace Log](#) (on page 3510).

```

120443 /* Caution. Error checks omitted */
120444 {
120445 int fd;
120446 trace_id_t trid;
120447 posix_trace_event_info trace_event;
120448 char trace_event_name[TRACE_EVENT_NAME_MAX];
120449 char * data;
120450 size_t maxdatasize=1024, returndatasize;
120451 int return_value;
120452 - - - - -

120453 /* Open an existing trace log */
120454 fd=open("/tmp/tracelog", O_RDONLY);
120455 /* Open a trace stream on the open log */
120456 posix_trace_open(fd, &trid);
120457 /*
120458 * Retrieve information about the trace stream which
120459 * was dumped in this trace log (see example)
120460 */
120461 - - - - -

```

```

120462 /* Allocate a buffer for trace event data */
120463 data=(char *)malloc(maxdatasize);
120464 /*
120465 * Retrieve the list of trace events used in this
120466 * trace log (see example)
120467 */
120468 - - - - -
120469 /* Read and print all trace event names and data out in a loop */
120470 while (1)
120471 {
120472 posix_trace_getnext_event(trid, &trace_event,
120473 data, maxdatasize, &returndatasize,&return_value);
120474 if (return_value != NULL) break;
120475 /*
120476 * Get the name of the trace event type associated
120477 * with trid trace ID
120478 */
120479 posix_trace_eventid_get_name(trid, trace_event.event_id,
120480 trace_event_name);
120481 {
120482 int i;
120483
120484 /* Print the trace event name out */
120485 printf("%s: ", trace_event_name);
120486 /* Print the trace event data out */
120487 for (i=0; i<returndatasize, i++) printf("%02.2X",
120488 (unsigned char)data[i]);
120489 printf("\n");
120490 }
120491 /* Close the trace stream */
120492 posix_trace_close(trid);
120493 /* The buffer data is deallocated */
120494 free(data);
120495 /* Now the file can be closed */
120496 close(fd);
120497 }

```

### Several Programming Manipulations

The following examples show some typical sets of operations needed in some contexts.

#### Trace Stream Attribute Manipulation

This example shows the manipulation of a trace stream attribute object in order to change the default value provided by a previous *posix\_trace\_attr\_init()* call.

```

120503 /* Caution. Error checks omitted */
120504 {
120505 trace_attr_t attr;
120506 size_t logsize=100000;
120507 - - - - -
120508 /* Initialize trace stream attributes */

```

```

120509 posix_trace_attr_init(&attr);
120510 /* Set the trace name in the attributes structure */
120511 posix_trace_attr_setname(&attr, "my_trace");
120512 /* Set the trace full policy */
120513 posix_trace_attr_setstreamfullpolicy(&attr, POSIX_TRACE_LOOP);
120514 /* Set the trace log size */
120515 posix_trace_attr_setlogsize(&attr, logsize);
120516 - - - - -
120517 }

```

#### 120518 Create a Trace Event Type Set and Change the Trace Event Type Filter

120519 This example is valid only if the Trace Event Filter option is supported. This example shows the  
 120520 manipulation of a trace event type set in order to change the trace event type filter for an  
 120521 existing active trace stream, which may be just-created, running, or suspended. Some sets of  
 120522 trace event types are well-known, such as the set of trace event types not associated with a  
 120523 process, some trace event types are just-built trace event types for this trace stream; one trace  
 120524 event type is the predefined trace event error type which is deleted from the trace event type set.

```

120525 /* Caution. Error checks omitted */
120526 {
120527 trace_id_t trid = existing_trace;
120528 trace_event_set_t set;
120529 trace_event_id_t trace_event1, trace_event2;
120530 - - - - -
120531 /* Initialize to an empty set of trace event types */
120532 /* (not strictly required because posix_trace_event_set_fill() */
120533 /* will ignore the prior contents of the event set.) */
120534 posix_trace_eventset_emptyset(&set);
120535 /*
120536 * Fill the set with all system trace events
120537 * not associated with a process
120538 */
120539 posix_trace_eventset_fill(&set, POSIX_TRACE_WOPID_EVENTS);
120540 /*
120541 * Get the trace event type identifier of the known trace event name
120542 * my_first_event for the trid trace stream
120543 */
120544 posix_trace_trid_eventid_open(trid, "my_first_event", &trace_event1);
120545 /* Add the set with this trace event type identifier */
120546 posix_trace_eventset_add_event(trace_event1, &set);
120547 /*
120548 * Get the trace event type identifier of the known trace event name
120549 * my_second_event for the trid trace stream
120550 */
120551 posix_trace_trid_eventid_open(trid, "my_second_event", &trace_event2);
120552 /* Add the set with this trace event type identifier */
120553 posix_trace_eventset_add_event(trace_event2, &set);
120554 - - - - -
120555 /* Delete the system trace event POSIX_TRACE_ERROR from the set */
120556 posix_trace_eventset_del_event(POSIX_TRACE_ERROR, &set);
120557 - - - - -

```

```

120558 /* Modify the trace stream filter making it equal to the new set */
120559 posix_trace_set_filter(trid, &set, POSIX_TRACE_SET_EVENTSET);
120560 - - - - -
120561 /*
120562 * Now trace_event1, trace_event2, and all system trace event types
120563 * not associated with a process, except for the POSIX_TRACE_ERROR
120564 * system trace event type, are filtered out of (not recorded in) the
120565 * existing trace stream.
120566 */
120567 }

```

### 120568 Retrieve Information from a Trace Log

120569 This example shows how to extract information from a trace log, the dump of a trace stream.  
 120570 This code:

- 120571 • Asks if the trace stream has lost trace events
- 120572 • Extracts the information about the version of the trace subsystem which generated this  
 120573 trace log
- 120574 • Retrieves the maximum size of trace event data; this may be used to dynamically allocate  
 120575 an array for extracting trace event data from the trace log without overflow

```

120576 /* Caution. Error checks omitted */
120577 {
120578 struct posix_trace_status_info statusinfo;
120579 trace_attr_t attr;
120580 trace_id_t trid = existing_trace;
120581 size_t maxdatasize;
120582 char genversion[TRACE_NAME_MAX];
120583 - - - - -
120584 /* Get the trace stream status */
120585 posix_trace_get_status(trid, &statusinfo);
120586 /* Detect an overrun condition */
120587 if (statusinfo.posix_stream_overrun_status == POSIX_TRACE_OVERRUN)
120588 printf("trace events have been lost\n");
120589 /* Get attributes from the trid trace stream */
120590 posix_trace_get_attr(trid, &attr);
120591 /* Get the trace generation version from the attributes */
120592 posix_trace_attr_getgenversion(&attr, genversion);
120593 /* Print the trace generation version out */
120594 printf("Information about Trace Generator:%s\n",genversion);
120595 /* Get the trace event max data size from the attributes */
120596 posix_trace_attr_getmaxdatasize(&attr, &maxdatasize);
120597 /* Print the trace event max data size out */
120598 printf("Maximum size of associated data:%d\n",maxdatasize);
120599 /* Destroy the trace stream attributes */
120600 posix_trace_attr_destroy(&attr);
120601 }

```

120602 **Retrieve the List of Trace Event Types Used in a Trace Log**

120603 This example shows the retrieval of a trace stream's trace event type list. This operation may be  
 120604 very useful if you are interested only in tracking the type of trace events in a trace log.

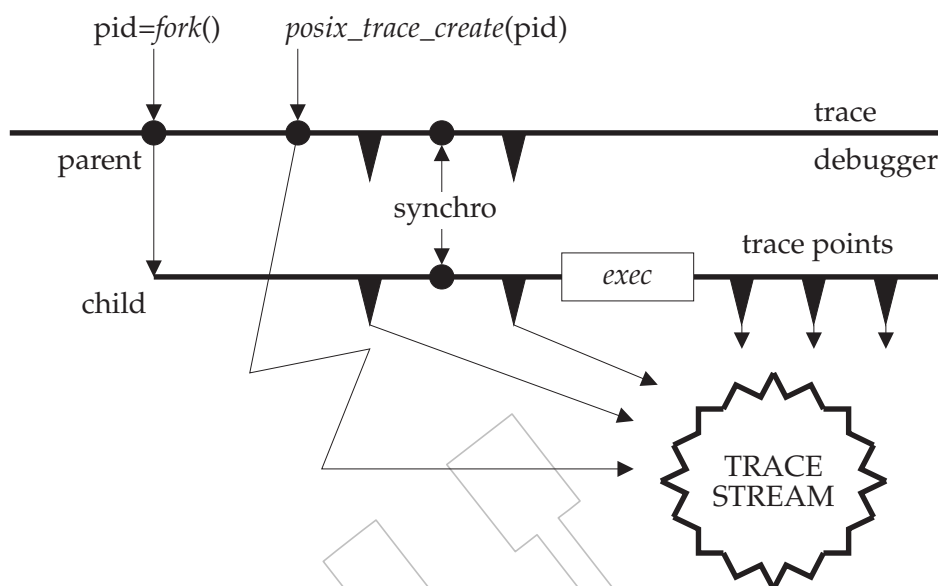
```

120605 /* Caution. Error checks omitted */
120606 {
120607 trace_id_t trid = existing_trace;
120608 trace_event_id_t event_id;
120609 char event_name[TRACE_EVENT_NAME_MAX];
120610 int return_value;
120611 - - - - -
120612 /*
120613 * In a loop print all existing trace event names out
120614 * for the trid trace stream
120615 */
120616 while (1)
120617 {
120618 posix_trace_eventtypelist_getnext_id(trid, &event_id
120619 &return_value);
120620 if (return_value != NULL) break;
120621 /*
120622 * Get the name of the trace event associated
120623 * with trid trace ID
120624 */
120625 posix_trace_eventid_get_name(trid, event_id, event_name);
120626 /* Print the name out */
120627 printf("%s\n", event_name);
120628 }
120629 }

```

120630

## B.2.11.4 Rationale on Trace for Debugging



120631

Figure B-5 Trace Another Process

120632

120633

120634

120635

120636

120637

Among the different possibilities offered by the trace interface defined in POSIX.1-200x, the debugging of an application is the most interesting one. Typical operations in the controlling debugger process are to filter trace event types, to get trace events from the trace stream, to stop the trace stream when the debugged process is executing uninteresting code, to start the trace stream when some interesting point is reached, and so on. The interface defined in POSIX.1-200x should define all the necessary base functions to allow this dynamic debug handling.

120638

120639

120640

120641

Figure B-5 shows an example in which the trace stream is created after the call to the *fork()* function. If the user does not want to lose trace events, some synchronization mechanism (represented in the figure) may be needed before calling the *exec* function, to give the parent a chance to create the trace stream before the child begins the execution of its trace points.

120642

## B.2.11.5 Rationale on Trace Event Type Name Space

120643

120644

120645

120646

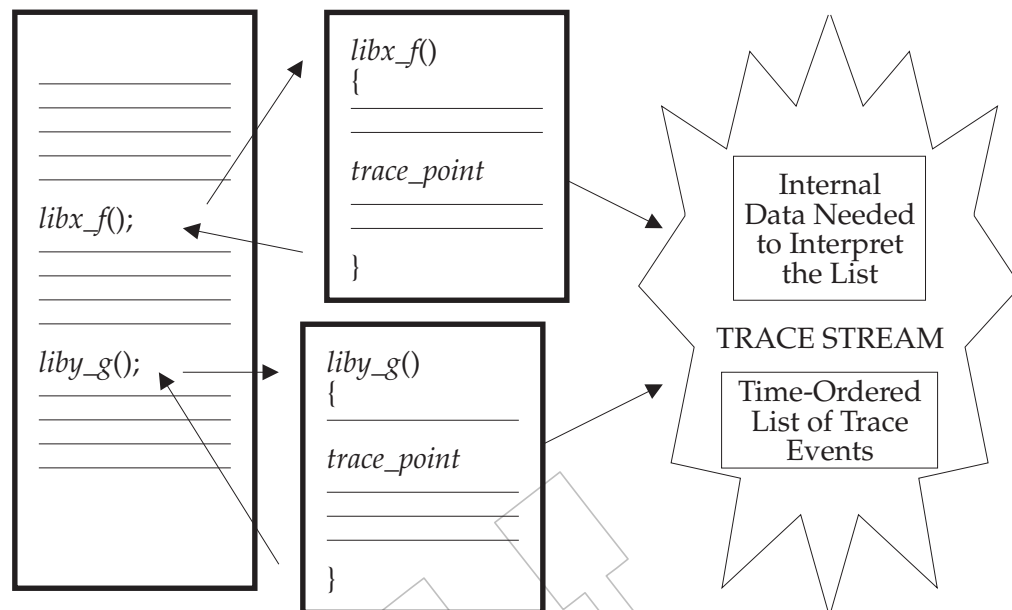
120647

120648

120649

120650

At first, the working group was in favor of the representation of a trace event type by an integer (*event\_name*). It seems that existing practice shows the weakness of such a representation. The collision of trace event types is the main problem that cannot be simply resolved using this sort of representation. Suppose, for example, that a third party designs an instrumented library. The user does not have the source of this library and wants to trace his application which uses in some part the third-party library. There is no means for him to know what are the trace event types used in the instrumented library so he has some chance of duplicating some of them and thus to obtain a contaminated tracing of his application.



120651 **Figure B-6** Trace Name Space Overview: With Third-Party Library

120652 There are requirements to allow program images containing pieces from various vendors to be  
 120653 traced without also requiring those of any other vendors to coordinate their uses of the trace  
 120654 facility, and especially the naming of their various trace event types and trace point IDs. The  
 120655 chosen solution is to provide a very large name space, large enough so that the individual  
 120656 vendors can give their trace types and tracepoint IDs sufficiently long and descriptive names  
 120657 making the occurrence of collisions quite unlikely. The probability of collision is thus made  
 120658 sufficiently low so that the problem may, as a practical matter, be ignored. By requirement, the  
 120659 consequence of collisions will be a slight ambiguity in the trace streams; tracing will continue in  
 120660 spite of collisions and ambiguities. “The show must go on”. The *posix\_prog\_address* member of  
 120661 the **posix\_trace\_event\_info** structure is used to allow trace streams to be unambiguously  
 120662 interpreted, despite the fact that trace event types and trace event names need not be unique.

120663 The *posix\_trace\_eventid\_open()* function is required to allow the instrumented third-party library  
 120664 to get a valid trace event type identifier for its trace event names. This operation is, somehow,  
 120665 an allocation, and the group was aware of proposing some deallocation mechanism which the  
 120666 instrumented application could use to recover the resources used by a trace event type identifier.  
 120667 This would have given the instrumented application the benefit of being capable of reusing a  
 120668 possible minimum set of trace event type identifiers, but also the inconvenience to have,  
 120669 possibly in the same trace stream, one trace event type identifier identifying two different trace  
 120670 event types. After some discussions the group decided to not define such a function which  
 120671 would make this API thicker for little benefit, the user having always the possibility of adding  
 120672 identification information in the *data* member of the trace event structure.

120673 The set of the trace event type identifiers the controlling process wants to filter out is initialized  
 120674 in the trace mechanism using the function *posix\_trace\_set\_filter()*, setting the arguments  
 120675 according to the definitions explained in *posix\_trace\_set\_filter()*. This operation can be done  
 120676 statically (when the trace is in the STOPPED state) or dynamically (when the trace is in the  
 120677 STARTED state). The preparation of the filter is normally done using the function defined in  
 120678 *posix\_trace\_eventtypelist\_getnext\_id()* and eventually the function  
 120679 *posix\_trace\_eventtypelist\_rewind()* in order to know (before the recording) the list of the potential



120680 set of trace event types that can be recorded. In the case of an active trace stream, this list may  
 120681 not be exhaustive. Actually, the target process may not have yet called the function  
 120682 *posix\_trace\_eventid\_open()*. But it is a common practice, for a controlling process, to prepare the  
 120683 filtering of a future trace stream before its start. Therefore the user must have a way to get the  
 120684 trace event type identifier corresponding to a well-known trace event name before its future  
 120685 association by the pre-cited function. This is done by calling the *posix\_trace\_trid\_eventid\_open()*  
 120686 function, given the trace stream identifier and the trace name, and described hereafter. Because  
 120687 this trace event type identifier is associated with a trace stream identifier, where a unique  
 120688 process has initialized two or more traces, the implementation is expected to return the same  
 120689 trace event type identifier for successive calls to *posix\_trace\_trid\_eventid\_open()* with different  
 120690 trace stream identifiers. The *posix\_trace\_eventid\_get\_name()* function is used by the controller  
 120691 process to identify, by the name, the trace event type returned by a call to the  
 120692 *posix\_trace\_eventtypelist\_getnext\_id()* function.

120693 Afterwards, the set of trace event types is constructed using the functions defined in  
 120694 *posix\_trace\_eventset\_empty()*, *posix\_trace\_eventset\_fill()*, *posix\_trace\_eventset\_add()*, and  
 120695 *posix\_trace\_eventset\_del()*.

120696 A set of functions is provided devoted to the manipulation of the trace event type identifier and  
 120697 names for an active trace stream. All these functions require the trace stream identifier argument  
 120698 as the first parameter. The opacity of the trace event type identifier implies that the user cannot  
 120699 associate directly its well-known trace event name with the system-associated trace event type  
 120700 identifier.

120701 The *posix\_trace\_trid\_eventid\_open()* function allows the application to get the system trace event  
 120702 type identifier back from the system, given its well-known trace event name. This function is  
 120703 useful only when a controlling process needs to specify specific events to be filtered.

120704 The *posix\_trace\_eventid\_get\_name()* function allows the application to obtain a trace event name  
 120705 given its trace event type identifier. One possible use of this function is to identify the type of a  
 120706 trace event retrieved from the trace stream, and print it. The easiest way to implement this  
 120707 requirement, is to use a single trace event type map for all the processes whose maps are  
 120708 required to be identical. A more difficult way is to attempt to keep multiple maps identical at  
 120709 every call to *posix\_trace\_eventid\_open()* and *posix\_trace\_trid\_eventid\_open()*.

#### 120710 B.2.11.6 Rationale on Trace Events Type Filtering

120711 The most basic rationale for runtime and pre-registration filtering (selection/rejection) of trace  
 120712 event types is to prevent choking of the trace collection facility, and/or overloading of the  
 120713 computer system. Any worthwhile trace facility can bring even the largest computer to its  
 120714 knees. Otherwise, everything would be recorded and filtered after the fact; it would be much  
 120715 simpler, but impractical.

120716 To achieve debugging, measurement, or whatever the purpose of tracing, the filtering of trace  
 120717 event types is an important part of trace analysis. Due to the fact that the trace events are put  
 120718 into a trace stream and probably logged afterwards into a file, different levels of filtering—that  
 120719 is, rejection of trace event types—are possible.

120720 **Filtering of Trace Event Types Before Tracing**

120721 This function, represented by the `posix_trace_set_filter()` function in POSIX.1-200x (see  
 120722 `posix_trace_set_filter()`), selects, before or during tracing, the set of trace event types to be filtered  
 120723 out. It should be possible also (as OSF suggested in their ETAP trace specifications) to select the  
 120724 kernel trace event types to be traced in a system-wide fashion. These two functionalities are  
 120725 called the pre-filtering of trace event types.

120726 The restriction on the actual type used for the `trace_event_set_t` type is intended to guarantee  
 120727 that these objects can always be assigned, have their address taken, and be passed by value as  
 120728 parameters. It is not intended that this type be a structure including pointers to other data  
 120729 structures, as that could impact the portability of applications performing such operations. A  
 120730 reasonable implementation could be a structure containing an array of integer types.

120731 **Filtering of Trace Event Types at Runtime**

120732 It is possible to build this functionality using the `posix_trace_set_filter()` function. A privileged  
 120733 process or a privileged thread can get trace events from the trace stream of another process or  
 120734 thread, and thus specify the type of trace events to record into a file, using implementation-  
 120735 defined methods and interfaces. This functionality, called inline filtering of trace event types, is  
 120736 used for runtime analysis of trace streams.

120737 **Post-Mortem Filtering of Trace Event Types**

120738 The word “post-mortem” is used here to indicate that some unanticipated situation occurs  
 120739 during execution that does not permit a pre or inline filtering of trace events and that it is  
 120740 necessary to record all trace event types to have a chance to discover the problem afterwards.  
 120741 When the program stops, all the trace events recorded previously can be analyzed in order to  
 120742 find the solution. This functionality could be named the post-filtering of trace event types.

120743 **Discussions about Trace Event Type-Filtering**

120744 After long discussions with the parties involved in the process of defining the trace interface, it  
 120745 seems that the sensitivity to the filtering problem is different, but everybody agrees that the level  
 120746 of the overhead introduced during the tracing operation depends on the filtering method  
 120747 elected. If the time that it takes the trace event to be recorded can be neglected, the overhead  
 120748 introduced by the filtering process can be classified as follows:

120749 Pre-filtering      System and process/thread-level overhead

120750 Inline-filtering    Process/thread-level overhead

120751 Post-filtering     No overhead; done offline

120752 The pre-filtering could be named “critical realtime” filtering in the sense that the filtering of  
 120753 trace event type is manageable at the user level so the user can lower to a minimum the filtering  
 120754 overhead at some user selected level of priority for the inline filtering, or delay the filtering to  
 120755 after execution for the post-filtering. The counterpart of this solution is that the size of the trace  
 120756 stream must be sufficient to record all the trace events. The advantage of the pre-filtering is that  
 120757 the utilization of the trace stream is optimized.

120758 Only pre-filtering is defined by POSIX.1-200x. However, great care must be taken in specifying  
 120759 pre-filtering, so that it does not impose unacceptable overhead. Moreover, it is necessary to  
 120760 isolate all the functionality relative to the pre-filtering.

120761 The result of this rationale is to define a new option, the Trace Event Filter option, not  
 120762 necessarily implemented in small realtime systems, where system overhead is minimized to the  
 120763 extent possible.

120764 B.2.11.7 Tracing, *pthread* API

120765 The objective to be able to control tracing for individual threads may be in conflict with the  
 120766 efficiency expected in threads with a *contentionscope* attribute of `PTHREAD_SCOPE_PROCESS`.  
 120767 For these threads, context switches from one thread that has tracing enabled to another thread  
 120768 that has tracing disabled may require a kernel call to inform the kernel whether it has to trace  
 120769 system events executed by that thread or not. For this reason, it was proposed that the ability to  
 120770 enable or disable tracing for `PTHREAD_SCOPE_PROCESS` threads be made optional, through  
 120771 the introduction of a Trace Scope Process option. A trace implementation which did not  
 120772 implement the Trace Scope Process option would not honor the tracing-state attribute of a thread  
 120773 with `PTHREAD_SCOPE_PROCESS`; it would, however, honor the tracing-state attribute of a  
 120774 thread with `PTHREAD_SCOPE_SYSTEM`. This proposal was rejected as:

- 120775 1. Removing desired functionality (per-thread trace control)
- 120776 2. Introducing counter-intuitive behavior for the tracing-state attribute
- 120777 3. Mixing logically orthogonal ideas (thread scheduling and thread tracing)
- 120778 [Objective 4]

120779 Finally, to solve this complex issue, this API does not provide `pthread_gettracingstate()`,  
 120780 `pthread_settracingstate()`, `pthread_attr_gettracingstate()`, and `pthread_attr_settracingstate()`  
 120781 interfaces. These interfaces force the thread implementation to add to the weight of the thread  
 120782 and cause a revision of the threads libraries, just to support tracing. Worse yet,  
 120783 `posix_trace_event()` must always test this per-thread variable even in the common case where it is  
 120784 not used at all. Per-thread tracing is easy to implement using existing interfaces where  
 120785 necessary; see the following example.

120786 **Example**

```

120787 /* Caution. Error checks omitted */
120788 static pthread_key_t my_key;
120789 static trace_event_id_t my_event_id;
120790 static pthread_once_t my_once = PTHREAD_ONCE_INIT;
120791
120792 void my_init(void)
120793 {
120794 (void) pthread_key_create(&my_key, NULL);
120795 (void) posix_trace_eventid_open("my", &my_event_id);
120796 }
120797
120798 int get_trace_flag(void)
120799 {
120800 pthread_once(&my_once, my_init);
120801 return (pthread_getspecific(my_key) != NULL);
120802 }
120803
120804 void set_trace_flag(int f)
120805 {
120806 pthread_once(&my_once, my_init);
120807 pthread_setspecific(my_key, f? &my_event_id: NULL);
120808 }
120809
120810 fn()
120811 {
120812 if (get_trace_flag())
120813 posix_trace_event(my_event_id, ...)
120814 }

```

120811 The above example does not implement third-party state setting.

120812 Lastly, per-thread tracing works poorly for threads with PTHREAD\_SCOPE\_PROCESS  
120813 contention scope. These “library” threads have minimal interaction with the kernel and would  
120814 have to explicitly set the attributes whenever they are context switched to a new kernel thread in  
120815 order to trace system events. Such state was explicitly avoided in POSIX threads to keep  
120816 PTHREAD\_SCOPE\_PROCESS threads lightweight.

120817 The reason that keeping PTHREAD\_SCOPE\_PROCESS threads lightweight is important is that  
120818 such threads can be used not just for simple multi-processors but also for co-routine style  
120819 programming (such as discrete event simulation) without inventing a new threads paradigm.  
120820 Adding extra runtime cost to thread context switches will make using POSIX threads less  
120821 attractive in these situations.

#### 120822 B.2.11.8 Rationale on Triggering

120823 The ability to start or stop tracing based on the occurrence of specific trace event types has been  
120824 proposed as a parallel to similar functionality appearing in logic analyzers. Such triggering, in  
120825 order to be very useful, should be based not only on the trace event type, but on trace event-  
120826 specific data, including tests of user-specified fields for matching or threshold values.

120827 Such a facility is unnecessary where the buffering of the stream is not a constraint, since such  
120828 checks can be performed offline during post-mortem analysis.

120829 For example, a large system could incorporate a daemon utility to collect the trace records from  
120830 memory buffers and spool them to secondary storage for later analysis. In the instances where  
120831 resources are truly limited, such as embedded applications, the application incorporation of  
120832 application code to test the circumstances of a trace event and call the trace point only if needed  
120833 is usually straightforward.

120834 For performance reasons, the *posix\_trace\_event()* function should be implemented using a macro,  
120835 so if the trace is inactive, the trace event point calls are latent code and must cost no more than a  
120836 scalar test.

120837 The API proposed in POSIX.1-200x does not include any triggering functionality.

#### 120838 B.2.11.9 Rationale on Timestamp Clock

120839 It has been suggested that the tracing mechanism should include the possibility of specifying the  
120840 clock to be used in timestamping the trace events. When application trace events must be  
120841 correlated to remote trace events, such a facility could provide a global time reference not  
120842 available from a local clock. Further, the application may be driven by timers based on a clock  
120843 different from that used for the timestamp, and the correlation of the trace to those untraced  
120844 timer activities could be an important part of the analysis of the application.

120845 However, the tracing mechanism needs to be fast and just the provision of such an option can  
120846 materially affect its performance. Leaving aside the performance costs of reading some clocks,  
120847 this notion is also ill-defined when kernel trace events are to be traced by two applications  
120848 making use of different tracing clocks. This can even happen within a single application where  
120849 different parts of the application are served by different clocks. Another complication can occur  
120850 when a clock is maintained strictly at the user level and is unavailable at the kernel level.

120851 It is felt that the benefits of a selectable trace clock do not match its costs. Applications that wish  
120852 to correlate clocks other than the default tracing clock can include trace events with sample  
120853 values of those other clocks, allowing correlation of timestamps from the various independent  
120854 clocks. In any case, such a technique would be required when applications are sensitive to  
120855 multiple clocks.

120856 **B.2.11.10 Rationale on Different Overrun Conditions**

120857 The analysis of the dynamic behavior of the trace mechanism shows that different overrun  
 120858 conditions may occur. The API must provide a means to manage such conditions in a portable  
 120859 way.

120860 **Overrun in Trace Streams Initialized with POSIX\_TRACE\_LOOP Policy**

120861 In this case, the user of the trace mechanism is interested in using the trace stream with  
 120862 POSIX\_TRACE\_LOOP policy to record trace events continuously, but ideally without losing any  
 120863 trace events. The online analyzer process must get the trace events at a mean speed equivalent to  
 120864 the recording speed. Should the trace stream become full, a trace stream overrun occurs. This  
 120865 condition is detected by getting the status of the active trace stream (function  
 120866 `posix_trace_get_status()`) and looking at the member `posix_stream_overrun_status` of the read  
 120867 **posix\_stream\_status** structure. In addition, two predefined trace event types are defined:

- 120868 1. The beginning of a trace overflow, to locate the beginning of an overflow when reading a  
 120869 trace stream
- 120870 2. The end of a trace overflow, to locate the end of an overflow, when reading a trace stream

120871 As a timestamp is associated with these predefined trace events, it is possible to know the  
 120872 duration of the overflow.

120873 **Overrun in Dumping Trace Streams into Trace Logs**

120874 The user lets the trace mechanism dump the trace stream initialized with  
 120875 POSIX\_TRACE\_FLUSH policy automatically into a trace log. If the dump operation is slower  
 120876 than the recording of trace events, the trace stream can overrun. This condition is detected by  
 120877 getting the status of the active trace stream (the `posix_trace_get_status()` function) and looking at  
 120878 the member `posix_stream_overrun_status` of the read **posix\_stream\_status** structure. This overrun  
 120879 indicates that the trace mechanism is not able to operate in this mode at this speed. It is the  
 120880 responsibility of the user to modify one of the trace parameters (the stream size or the trace  
 120881 event type filter, for instance) to avoid such overrun conditions, if overruns are to be prevented.  
 120882 The same already predefined trace event types (see [Overrun in Trace Streams Initialized with  
 120883 POSIX\\_TRACE\\_LOOP Policy](#)) are used to detect and to know the duration of an overflow.

120884 **Reading an Active Trace Stream**

120885 Although this trace API allows one to read an active trace stream with log while it is tracing, this  
 120886 feature can lead to false overflow origin interpretation: the trace log or the reader of the trace  
 120887 stream. Reading from an active trace stream with log is thus non-portable, and has been left  
 120888 unspecified.

120889 **B.2.12 Data Types**120890 **B.2.12.1 Defined Types**

120891 The requirement that additional types defined in this section end in “\_t” was prompted by the  
 120892 problem of name space pollution. It is difficult to define a type (where that type is not one  
 120893 defined by POSIX.1-200x) in one header file and use it in another without adding symbols to the  
 120894 name space of the program. To allow implementors to provide their own types, all conforming  
 120895 applications are required to avoid symbols ending in “\_t”, which permits the implementor to  
 120896 provide additional types. Because a major use of types is in the definition of structure members,  
 120897 which can (and in many cases must) be added to the structures defined in POSIX.1-200x, the

120898 need for additional types is compelling.

120899 The types, such as **ushort** and **ulong**, which are in common usage, are not defined in  
 120900 POSIX.1-200x (although **ushort\_t** would be permitted as an extension). They can be added to  
 120901 **<sys/types.h>** using a feature test macro (see [Section B.2.2.1](#), on page 3398). A suggested symbol  
 120902 for these is **\_SYSIII**. Similarly, the types like **u\_short** would probably be best controlled by **\_BSD**.

120903 Some of these symbols may appear in other headers; see [Section B.2.2.2](#) (on page 3399).

120904 **dev\_t** This type may be made large enough to accommodate host-locality considerations  
 120905 of networked systems.

120906 This type must be arithmetic. Earlier proposals allowed this to be non-arithmetic  
 120907 (such as a structure) and provided a *samefile()* function for comparison.

120908 **gid\_t** Some implementations had separated **gid\_t** from **uid\_t** before POSIX.1 was  
 120909 completed. It would be difficult for them to coalesce them when it was  
 120910 unnecessary. Additionally, it is quite possible that user IDs might be different from  
 120911 group IDs because the user ID might wish to span a heterogeneous network,  
 120912 where the group ID might not.

120913 For current implementations, the cost of having a separate **gid\_t** will be only  
 120914 lexical.

120915 **mode\_t** This type was chosen so that implementations could choose the appropriate  
 120916 integer type, and for compatibility with the ISO C standard. 4.3 BSD uses  
 120917 **unsigned short** and the SVID uses **ushort**, which is the same. Historically, only the  
 120918 low-order sixteen bits are significant.

120919 **nlink\_t** This type was introduced in place of **short** for *st\_nlink* (see the **<sys/stat.h>** header)  
 120920 in response to an objection that **short** was too small.

120921 **off\_t** This type is used only in *lseek()*, *fcntl()*, and **<sys/stat.h>**. Many implementations  
 120922 would have difficulties if it were defined as anything other than **long**. Requiring  
 120923 an integer type limits the capabilities of *lseek()* to four gigabytes. The ISO C  
 120924 standard supplies routines that use larger types; see *fgetpos()* and *fsetpos()*. XSI-  
 120925 conformant systems provide the *fseeko()* and *ftello()* functions that use larger types.

120926 **pid\_t** The inclusion of this symbol was controversial because it is tied to the issue of the  
 120927 representation of a process ID as a number. From the point of view of a  
 120928 conforming application, process IDs should be “magic cookies”<sup>7</sup> that are produced  
 120929 by calls such as *fork()*, used by calls such as *waitpid()* or *kill()*, and not otherwise  
 120930 analyzed (except that the sign is used as a flag for certain operations).

120931 The concept of a {PID\_MAX} value interacted with this in early proposals. Treating  
 120932 process IDs as an opaque type both removes the requirement for {PID\_MAX} and  
 120933 allows systems to be more flexible in providing process IDs that span a large range  
 120934 of values, or a small one.

120935 Since the values in **uid\_t**, **gid\_t**, and **pid\_t** will be numbers generally, and  
 120936 potentially both large in magnitude and sparse, applications that are based on  
 120937 arrays of objects of this type are unlikely to be fully portable in any case. Solutions  
 120938 that treat them as magic cookies will be portable.

120939 {CHILD\_MAX} precludes the possibility of a “toy implementation”, where there

---

120940 7. An historical term meaning: “An opaque object, or token, of determinate size, whose significance is known only to the entity which  
 120941 created it. An entity receiving such a token from the generating entity may only make such use of the ‘cookie’ as is defined and permitted  
 120942 by the supplying entity.”

|        |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 120943 |                 | would only be one process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 120944 | <b>ssize_t</b>  | This is intended to be a signed analog of <b>size_t</b> . The wording is such that an implementation may either choose to use a longer type or simply to use the signed version of the type that underlies <b>size_t</b> . All functions that return <b>ssize_t</b> ( <i>read()</i> and <i>write()</i> ) describe as “implementation-defined” the result of an input exceeding {SSIZE_MAX}. It is recognized that some implementations might have <b>ints</b> that are smaller than <b>size_t</b> . A conforming application would be constrained not to perform I/O in pieces larger than {SSIZE_MAX}, but a conforming application using extensions would be able to use the full range if the implementation provided an extended range, while still having a single type-compatible interface. |
| 120945 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120946 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120947 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120948 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120949 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120950 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120951 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120952 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120953 |                 | The symbols <b>size_t</b> and <b>ssize_t</b> are also required in <b>&lt;unistd.h&gt;</b> to minimize the changes needed for calls to <i>read()</i> and <i>write()</i> . Implementors are reminded that it must be possible to include both <b>&lt;sys/types.h&gt;</b> and <b>&lt;unistd.h&gt;</b> in the same program (in either order) without error.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 120954 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120955 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120956 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120957 | <b>uid_t</b>    | Before the addition of this type, the data types used to represent these values varied throughout early proposals. The <b>&lt;sys/stat.h&gt;</b> header defined these values as type <b>short</b> , the <b>&lt;passwd.h&gt;</b> file (now <b>&lt;pwd.h&gt;</b> and <b>&lt;grp.h&gt;</b> ) used an <b>int</b> , and <i>getuid()</i> returned an <b>int</b> . In response to a strong objection to the inconsistent definitions, all the types were switched to <b>uid_t</b> .                                                                                                                                                                                                                                                                                                                       |
| 120958 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120959 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120960 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120961 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120962 |                 | In practice, those historical implementations that use varying types of this sort can typedef <b>uid_t</b> to <b>short</b> with no serious consequences.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 120963 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120964 |                 | The problem associated with this change concerns object compatibility after structure size changes. Since most implementations will define <b>uid_t</b> as a <b>short</b> , the only substantive change will be a reduction in the size of the <b>passwd</b> structure. Consequently, implementations with an overriding concern for object compatibility can pad the structure back to its current size. For that reason, this problem was not considered critical enough to warrant the addition of a separate type to POSIX.1.                                                                                                                                                                                                                                                                  |
| 120965 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120966 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120967 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120968 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120969 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120970 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120971 |                 | The types <b>uid_t</b> and <b>gid_t</b> are magic cookies. There is no {UID_MAX} defined by POSIX.1, and no structure imposed on <b>uid_t</b> and <b>gid_t</b> other than that they be positive arithmetic types. (In fact, they could be <b>unsigned char</b> .) There is no maximum or minimum specified for the number of distinct user or group IDs.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 120972 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120973 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120974 |                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 120975 | <b>B.2.12.2</b> | <i>The char Type</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 120976 |                 | POSIX.1-200x explicitly requires that a <b>char</b> type is exactly one byte (8 bits).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 120977 | <b>B.2.12.3</b> | <i>Pointer Types</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 120978 |                 | POSIX.1-200x explicitly requires implementations to convert pointers to <b>void *</b> and back with no                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 120979 |                 | loss of information. This is an extension over the ISO C standard.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## 120980 B.3 System Interfaces

120981 See the RATIONALE sections on the individual reference pages.

### 120982 B.3.1 System Interfaces Removed in this Version

120983 The following section contains a list of the interfaces removed in POSIX.1-200x, together with  
 120984 advice for application developers on the alternative interfaces that should be used for maximum  
 120985 portability.

#### 120986 B.3.1.1 *bcmp*

120987 Applications are recommended to use the *memcmp()* function instead of this function.

120988 For maximum portability, it is recommended to replace the function call to *bcmp()* as follows:

```
120989 #define bcmp(b1,b2,len) memcmp((b1), (b2), (size_t)(len))
```

#### 120990 B.3.1.2 *bcopy*

120991 Applications are recommended to use the *memmove()* function instead of this function.

120992 The following are approximately equivalent (note the order of the arguments):

```
120993 bcopy(s1,s2,n) ~ memmove(s2,s1,n)
```

120994 For maximum portability, it is recommended to replace the function call to *bcopy()* as follows:

```
120995 #define bcopy(b1,b2,len) (void)(memmove((b2), (b1), (len)))
```

#### 120996 B.3.1.3 *bsd\_signal*

120997 Applications are recommended to use the *sigaction()* function instead of this function.

120998 The *bsd\_signal()* function was supplied as a migration path for the BSD *signal()* function for  
 120999 simple applications that installed a single-argument signal handler function.

121000 Historically, the *bsd\_signal()* function differs from *signal()* in that the SA\_RESTART flag is set  
 121001 and the SA\_RESETHAND flag is clear when *bsd\_signal()* is used. The state of these flags is not  
 121002 specified for *signal()*.

#### 121003 B.3.1.4 *bzero*

121004 Applications are recommended to use the *memset()* function instead of this function.

121005 For maximum portability, it is recommended to replace the function call to *bzero()* as follows:

```
121006 #define bzero(b,len) (void)(memset((b), '\0', (len)))
```

#### 121007 B.3.1.5 *ecvt, fcvt, gcvt*

121008 Applications are recommended to use the *sprintf()* function instead of these functions.

121009 The *sprintf()* function is required by ISO C and is thus more portable.



121010 B.3.1.6 *ftime*

121011 Applications are recommended to use the *time()* function to determine the current time.  
 121012 Realtime applications should use *clock\_gettime()* to determine the current time.

121013 B.3.1.7 *getcontext, makecontext, swapcontext*

121014 Due to portability issues with these functions, especially with the manipulation of contexts,  
 121015 applications are recommended to be rewritten to use POSIX threads.

121016 B.3.1.8 *gethostbyaddr, gethostbyname*

121017 Applications are recommended to use the *getaddrinfo()* and *getnameinfo()* functions instead of  
 121018 these functions.

121019 The *gethostbyaddr()* and *gethostbyname()* functions may return pointers to static data, which may  
 121020 be overwritten by subsequent calls to any of these functions. The suggested replacements do not  
 121021 have this problem and are also IPv6-capable.

121022 B.3.1.9 *getwd*

121023 Applications are recommended to use the *getcwd()* function to determine the current working  
 121024 directory.

121025 B.3.1.10 *h\_errno*

121026 Applications are recommended not to use this error return code. Previously it was set by the  
 121027 *gethostbyname()* and *gethostbyaddr()* functions.

121028 B.3.1.11 *index*

121029 Applications are recommended to use the *strchr()* function instead of this function.

121030 For maximum portability, it is recommended to replace the function call to *index()* as follows:

```
121031 #define index(a,b) strchr((a),(b))
```

121032 B.3.1.12 *makecontext*

121033 Applications using the *getcontext()*, *makecontext()*, and *swapcontext()* functions should be  
 121034 rewritten to use POSIX threads.

121035 B.3.1.13 *mktemp*

121036 Applications are recommended to use the *mkstemp()* function instead of this function.

121037 The *mktemp()* function makes an application vulnerable to possible security problems since  
 121038 between the time a pathname is created and the file opened, it is possible for some other process  
 121039 to create a file with the same name. The *mkstemp()* function does not have this vulnerability.

121040 B.3.1.14 *pthread\_attr\_getstackaddr, pthread\_attr\_setstackaddr*

121041 Applications are recommended to use the *pthread\_attr\_setstack()* and *pthread\_attr\_getstack()*  
 121042 functions instead of these functions.

121043 There are a number of ambiguities in the specification of the *stackaddr* attribute that makes  
 121044 portable use of these interfaces impossible.

121045 **B.3.1.15** *rindex*121046 Applications are recommended to use the *strchr()* function instead of this function.121047 For maximum portability, it is recommended to replace the function call to *rindex()* as follows:121048 

```
#define rindex(a,b) strchr((a),(b))
```

121049 **B.3.1.16** *scalb*121050 Applications are recommended to use either *scalbln()*, *scalblnf()*, or *scalblnl()* instead of these  
121051 functions.121052 The behavior for the *scalb()* function was only defined when the *n* argument is an integer, a  
121053 NaN, or Inf. The behavior of other values for the *n* argument was unspecified.121054 **B.3.1.17** *ualarm*121055 Applications are recommended to use *timer\_create()*, *timer\_delete()*, *timer\_getoverrun()*,  
121056 *timer\_gettime()*, or *timer\_settime()* instead of this function.121057 **B.3.1.18** *usleep*121058 Applications are recommended to use the *nanosleep()* function instead of this function.121059 **B.3.1.19** *vfork*121060 Applications are recommended to use the *fork()* function instead of this function.121061 The *vfork()* function was previously under-specified.121062 **B.3.1.20** *wcs wcs*121063 Applications are recommended to use the *wcsstr()* function instead of this function.121064 The *wcsstr()* function is technically equivalent and is portable across all ISO C implementations.121065 **B.3.2 Examples for Spawn**121066 The following long examples are provided in the Rationale (Informative) volume of  
121067 POSIX.1-200x as a supplement to the reference page for *posix\_spawn()*.121068 **Example Library Implementation of Spawn**121069 The *posix\_spawn()* or *posix\_spawnnp()* functions provide the following:

- 121070 • Simply start a process executing a process image. This is the simplest application for  
121071 process creation, and it may cover most executions of *fork()*.
- 121072 • Support I/O redirection, including pipes.
- 121073 • Run the child under a user and group ID in the domain of the parent.
- 121074 • Run the child at any priority in the domain of the parent.

121075 The *posix\_spawn()* or *posix\_spawnnp()* functions do not cover every possible use of the *fork()*  
121076 function, but they do span the common applications: typical use by a shell and a login utility.121077 The price for an application is that before it calls *posix\_spawn()* or *posix\_spawnnp()*, the parent  
121078 must adjust to a state that *posix\_spawn()* or *posix\_spawnnp()* can map to the desired state for the  
121079 child. Environment changes require the parent to save some of its state and restore it afterwards.  
121080 The effective behavior of a successful invocation of *posix\_spawn()* is as if the operation were

```

121081 implemented with POSIX operations as follows:
121082 #include <sys/types.h>
121083 #include <stdlib.h>
121084 #include <stdio.h>
121085 #include <unistd.h>
121086 #include <sched.h>
121087 #include <fcntl.h>
121088 #include <signal.h>
121089 #include <errno.h>
121090 #include <string.h>
121091 #include <signal.h>
121092
121093 /* #include <spawn.h> */
121094 /*****
121095 /* Things that could be defined in spawn.h */
121096 /*****
121097 typedef struct
121098 {
121099 short posix_attr_flags;
121100 #define POSIX_SPAWN_SETPGROUP 0x1
121101 #define POSIX_SPAWN_SETSIGMASK 0x2
121102 #define POSIX_SPAWN_SETSIGDEF 0x4
121103 #define POSIX_SPAWN_SETSCHEDULER 0x8
121104 #define POSIX_SPAWN_SETSCHEDPARAM 0x10
121105 #define POSIX_SPAWN_RESETIDS 0x20
121106 pid_t posix_attr_pgroup;
121107 sigset_t posix_attr_sigmask;
121108 sigset_t posix_attr_sigdefault;
121109 int posix_attr_schedpolicy;
121110 struct sched_param posix_attr_schedparam;
121111 } posix_spawnattr_t;
121112
121113 typedef char *posix_spawn_file_actions_t;
121114
121115 int posix_spawn_file_actions_init(
121116 posix_spawn_file_actions_t *file_actions);
121117 int posix_spawn_file_actions_destroy(
121118 posix_spawn_file_actions_t *file_actions);
121119 int posix_spawn_file_actions_addclose(
121120 posix_spawn_file_actions_t *file_actions, int fildes);
121121 int posix_spawn_file_actions_adddup2(
121122 posix_spawn_file_actions_t *file_actions, int fildes,
121123 int newfildes);
121124 int posix_spawn_file_actions_addopen(
121125 posix_spawn_file_actions_t *file_actions, int fildes,
121126 const char *path, int oflag, mode_t mode);
121127 int posix_spawnattr_init(posix_spawnattr_t *attr);
121128 int posix_spawnattr_destroy(posix_spawnattr_t *attr);
121129 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
121130 short *lags);
121131 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
121132 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
121133 pid_t *pgroup);
121134 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);

```

```

121132 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,
121133 int *schedpolicy);
121134 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
121135 int schedpolicy);
121136 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
121137 struct sched_param *schedparam);
121138 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
121139 const struct sched_param *schedparam);
121140 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
121141 sigset_t *sigmask);
121142 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
121143 const sigset_t *sigmask);
121144 int posix_spawnattr_getdefault(const posix_spawnattr_t *attr,
121145 sigset_t *sigdefault);
121146 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
121147 const sigset_t *sigdefault);
121148 int posix_spawn(pid_t *pid, const char *path,
121149 const posix_spawn_file_actions_t *file_actions,
121150 const posix_spawnattr_t *attrp, char *const argv[],
121151 char *const envp[]);
121152 int posix_spawnp(pid_t *pid, const char *file,
121153 const posix_spawn_file_actions_t *file_actions,
121154 const posix_spawnattr_t *attrp, char *const argv[],
121155 char *const envp[]);
121156 /*****
121157 /* Example posix_spawn() library routine */
121158 /*****
121159 int posix_spawn(pid_t *pid,
121160 const char *path,
121161 const posix_spawn_file_actions_t *file_actions,
121162 const posix_spawnattr_t *attrp,
121163 char *const argv[],
121164 char *const envp[])
121165 {
121166 /* Create process */
121167 if ((*pid = fork()) == (pid_t) 0)
121168 {
121169 /* This is the child process */
121170 /* Worry about process group */
121171 if (attrp->posix_attr_flags & POSIX_SPAWN_SETPGROUP)
121172 {
121173 /* Override inherited process group */
121174 if (setpgid(0, attrp->posix_attr_pgroup) != 0)
121175 {
121176 /* Failed */
121177 exit(127);
121178 }
121179 }
121180 /* Worry about thread signal mask */
121181 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGMASK)
121182 {
121183 /* Set the signal mask (can't fail) */

```

```

121184 sigprocmask(SIG_SETMASK, &attrp->posix_attr_sigmask, NULL);
121185 }
121186 /* Worry about resetting effective user and group IDs */
121187 if (attrp->posix_attr_flags & POSIX_SPAWN_RESETPIDS)
121188 {
121189 /* None of these can fail for this case. */
121190 setuid(getuid());
121191 setgid(getgid());
121192 }
121193 /* Worry about defaulted signals */
121194 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSIGDEF)
121195 {
121196 struct sigaction deflt;
121197 sigset_t all_signals;
121198
121199 int s;
121200 /* Construct default signal action */
121201 deflt.sa_handler = SIG_DFL;
121202 deflt.sa_flags = 0;
121203 /* Construct the set of all signals */
121204 sigfillset(&all_signals);
121205 /* Loop for all signals */
121206 for (s = 0; sigismember(&all_signals, s); s++)
121207 {
121208 /* Signal to be defaulted? */
121209 if (sigismember(&attrp->posix_attr_sigdefault, s))
121210 {
121211 /* Yes; default this signal */
121212 if (sigaction(s, &deflt, NULL) == -1)
121213 {
121214 /* Failed */
121215 exit(127);
121216 }
121217 }
121218 }
121219 /* Worry about the fds if they are to be mapped */
121220 if (file_actions != NULL)
121221 {
121222 /* Loop for all actions in object file_actions */
121223 /* (implementation dives beneath abstraction) */
121224 char *p = *file_actions;
121225 while (*p != '\0')
121226 {
121227 if (strncmp(p, "close(", 6) == 0)
121228 {
121229 int fd;
121230 if (sscanf(p + 6, "%d", &fd) != 1)
121231 {

```

```

121232 exit(127);
121233 }
121234 if (close(fd) == -1)
121235 exit(127);
121236 }
121237 else if (strncmp(p, "dup2(", 5) == 0)
121238 {
121239 int fd, newfd;
121240 if (sscanf(p + 5, "%d,%d", &fd, &newfd) != 2)
121241 {
121242 exit(127);
121243 }
121244 if (dup2(fd, newfd) == -1)
121245 exit(127);
121246 }
121247 else if (strncmp(p, "open(", 5) == 0)
121248 {
121249 int fd, oflag;
121250 mode_t mode;
121251 int tempfd;
121252 char path[1000]; /* Should be dynamic */
121253 char *q;
121254 if (sscanf(p + 5, "%d", &fd) != 1)
121255 {
121256 exit(127);
121257 }
121258 p = strchr(p, ',') + 1;
121259 q = strchr(p, '*');
121260 if (q == NULL)
121261 exit(127);
121262 strncpy(path, p, q - p);
121263 path[q - p] = '\0';
121264 if (sscanf(q + 1, "%o,%o", &oflag, &mode) != 2)
121265 {
121266 exit(127);
121267 }
121268 if (close(fd) == -1)
121269 {
121270 if (errno != EBADF)
121271 exit(127);
121272 }
121273 tempfd = open(path, oflag, mode);
121274 if (tempfd == -1)
121275 exit(127);
121276 if (tempfd != fd)
121277 {
121278 if (dup2(tempfd, fd) == -1)
121279 {
121280 exit(127);
121281 }
121282 if (close(tempfd) == -1)
121283 {

```

```

121284 exit(127);
121285 }
121286 }
121287 }
121288 else
121289 {
121290 exit(127);
121291 }
121292 p = strchr(p, ')') + 1;
121293 }
121294 }
121295 /* Worry about setting new scheduling policy and parameters */
121296 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDULER)
121297 {
121298 if (sched_setscheduler(0, attrp->posix_attr_schedpolicy,
121299 &attrp->posix_attr_schedparam) == -1)
121300 {
121301 exit(127);
121302 }
121303 }
121304 /* Worry about setting only new scheduling parameters */
121305 if (attrp->posix_attr_flags & POSIX_SPAWN_SETSCHEDPARAM)
121306 {
121307 if (sched_setparam(0, &attrp->posix_attr_schedparam) == -1)
121308 {
121309 exit(127);
121310 }
121311 }
121312 /* Now execute the program at path */
121313 /* Any fd that still has FD_CLOEXEC set will be closed */
121314 execve(path, argv, envp);
121315 exit(127); /* exec failed */
121316 }
121317 else
121318 {
121319 /* This is the parent (calling) process */
121320 if (*pid == (pid_t) - 1)
121321 return errno;
121322 return 0;
121323 }
121324 }
121325
121326 /* *****
121327 /* Here is a crude but effective implementation of the */
121328 /* file action object operators which store actions as */
121329 /* concatenated token-separated strings. */
121330 /* *****
121331 /* Create object with no actions. */
121332 int posix_spawn_file_actions_init(
121333 posix_spawn_file_actions_t *file_actions)
121334 {
121335 *file_actions = malloc(sizeof(char));

```

```

121335 if (*file_actions == NULL)
121336 return ENOMEM;
121337 strcpy(*file_actions, "");
121338 return 0;
121339 }
121340 /* Free object storage and make invalid. */
121341 int posix_spawn_file_actions_destroy(
121342 posix_spawn_file_actions_t *file_actions)
121343 {
121344 free(*file_actions);
121345 *file_actions = NULL;
121346 return 0;
121347 }
121348 /* Add a new action string to object. */
121349 static int add_to_file_actions(
121350 posix_spawn_file_actions_t *file_actions, char *new_action)
121351 {
121352 *file_actions = realloc
121353 (*file_actions, strlen(*file_actions) + strlen(new_action) + 1);
121354 if (*file_actions == NULL)
121355 return ENOMEM;
121356 strcat(*file_actions, new_action);
121357 return 0;
121358 }
121359 /* Add a close action to object. */
121360 int posix_spawn_file_actions_addclose(
121361 posix_spawn_file_actions_t *file_actions, int fildes)
121362 {
121363 char temp[100];
121364 sprintf(temp, "close(%d)", fildes);
121365 return add_to_file_actions(file_actions, temp);
121366 }
121367 /* Add a dup2 action to object. */
121368 int posix_spawn_file_actions_adddup2(
121369 posix_spawn_file_actions_t *file_actions, int fildes,
121370 int newfildes)
121371 {
121372 char temp[100];
121373 sprintf(temp, "dup2(%d,%d)", fildes, newfildes);
121374 return add_to_file_actions(file_actions, temp);
121375 }
121376 /* Add an open action to object. */
121377 int posix_spawn_file_actions_addopen(
121378 posix_spawn_file_actions_t *file_actions, int fildes,
121379 const char *path, int oflag, mode_t mode)
121380 {
121381 char temp[100];
121382 sprintf(temp, "open(%d,%s*%o,%o)", fildes, path, oflag, mode);
121383 return add_to_file_actions(file_actions, temp);

```



```

121384 }
121385 /*****
121386 /* Here is a crude but effective implementation of the */
121387 /* spawn attributes object functions which manipulate */
121388 /* the individual attributes. */
121389 /*****
121390 /* Initialize object with default values. */
121391 int posix_spawnattr_init(posix_spawnattr_t *attr)
121392 {
121393 attr->posix_attr_flags = 0;
121394 attr->posix_attr_pgroup = 0;
121395 /* Default value of signal mask is the parent's signal mask; */
121396 /* other values are also allowed */
121397 sigprocmask(0, NULL, &attr->posix_attr_sigmask);
121398 sigemptyset(&attr->posix_attr_sigdefault);
121399 /* Default values of scheduling attr inherited from the parent; */
121400 /* other values are also allowed */
121401 attr->posix_attr_schedpolicy = sched_getscheduler(0);
121402 sched_getparam(0, &attr->posix_attr_schedparam);
121403 return 0;
121404 }
121405 int posix_spawnattr_destroy(posix_spawnattr_t *attr)
121406 {
121407 /* No action needed */
121408 return 0;
121409 }
121410 int posix_spawnattr_getflags(const posix_spawnattr_t *attr,
121411 short *flags)
121412 {
121413 *flags = attr->posix_attr_flags;
121414 return 0;
121415 }
121416 int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags)
121417 {
121418 attr->posix_attr_flags = flags;
121419 return 0;
121420 }
121421 int posix_spawnattr_getpgroup(const posix_spawnattr_t *attr,
121422 pid_t *pgroup)
121423 {
121424 *pgroup = attr->posix_attr_pgroup;
121425 return 0;
121426 }
121427 int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup)
121428 {
121429 attr->posix_attr_pgroup = pgroup;
121430 return 0;
121431 }
121432 int posix_spawnattr_getschedpolicy(const posix_spawnattr_t *attr,

```

```

121433 int *schedpolicy)
121434 {
121435 *schedpolicy = attr->posix_attr_schedpolicy;
121436 return 0;
121437 }
121438 int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
121439 int schedpolicy)
121440 {
121441 attr->posix_attr_schedpolicy = schedpolicy;
121442 return 0;
121443 }
121444 int posix_spawnattr_getschedparam(const posix_spawnattr_t *attr,
121445 struct sched_param *schedparam)
121446 {
121447 *schedparam = attr->posix_attr_schedparam;
121448 return 0;
121449 }
121450 int posix_spawnattr_setschedparam(posix_spawnattr_t *attr,
121451 const struct sched_param *schedparam)
121452 {
121453 attr->posix_attr_schedparam = *schedparam;
121454 return 0;
121455 }
121456 int posix_spawnattr_getsigmask(const posix_spawnattr_t *attr,
121457 sigset_t *sigmask)
121458 {
121459 *sigmask = attr->posix_attr_sigmask;
121460 return 0;
121461 }
121462 int posix_spawnattr_setsigmask(posix_spawnattr_t *attr,
121463 const sigset_t *sigmask)
121464 {
121465 attr->posix_attr_sigmask = *sigmask;
121466 return 0;
121467 }
121468 int posix_spawnattr_getsigdefault(const posix_spawnattr_t *attr,
121469 sigset_t *sigdefault)
121470 {
121471 *sigdefault = attr->posix_attr_sigdefault;
121472 return 0;
121473 }
121474 int posix_spawnattr_setsigdefault(posix_spawnattr_t *attr,
121475 const sigset_t *sigdefault)
121476 {
121477 attr->posix_attr_sigdefault = *sigdefault;
121478 return 0;
121479 }

```

121480 **I/O Redirection with Spawn**

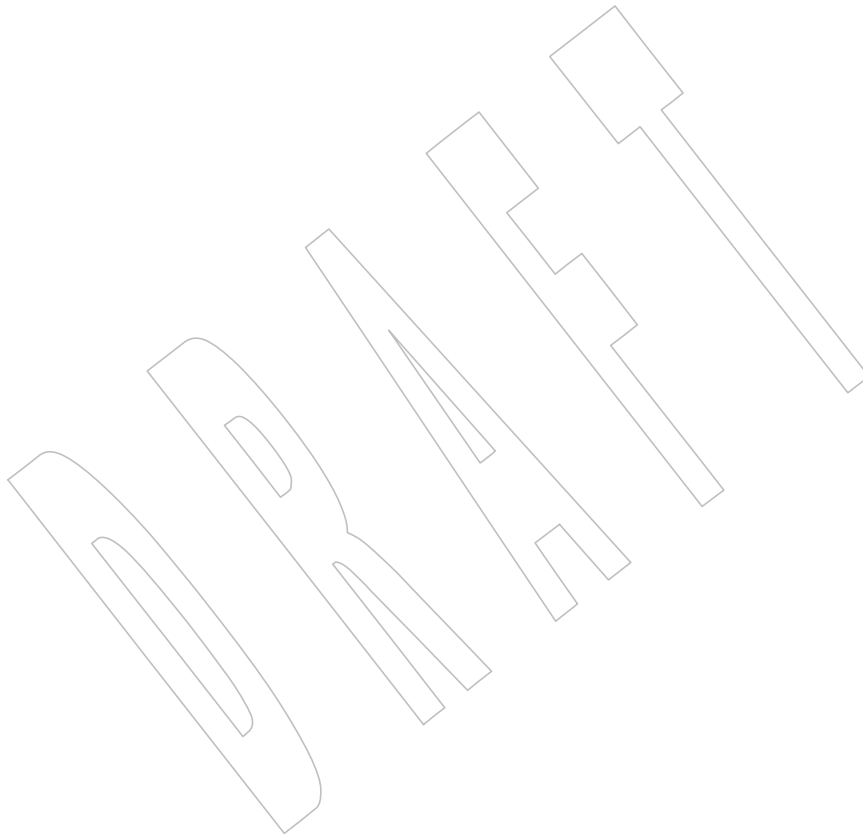
121481 I/O redirection with *posix\_spawn()* or *posix\_spawnnp()* is accomplished by crafting a *file\_actions*  
121482 argument to effect the desired redirection. Such a redirection follows the general outline of the  
121483 following example:

```
121484 /* To redirect new standard output (fd 1) to a file, */
121485 /* and redirect new standard input (fd 0) from my fd socket_pair[1], */
121486 /* and close my fd socket_pair[0] in the new process. */
121487 posix_spawn_file_actions_t file_actions;
121488 posix_spawn_file_actions_init(&file_actions);
121489 posix_spawn_file_actions_addopen(&file_actions, 1, "newout", ...);
121490 posix_spawn_file_actions_dup2(&file_actions, socket_pair[1], 0);
121491 posix_spawn_file_actions_close(&file_actions, socket_pair[0]);
121492 posix_spawn_file_actions_close(&file_actions, socket_pair[1]);
121493 posix_spawn(..., &file_actions, ...);
121494 posix_spawn_file_actions_destroy(&file_actions);
```

121495 **Spawning a Process Under a New User ID**

121496 Spawning a process under a new user ID follows the outline shown in the following example:

```
121497 Save = getuid();
121498 setuid(newid);
121499 posix_spawn(...);
121500 setuid(Save);
```



121501

**/** *Rationale (Informative)*

121502

**Part C:**

121503

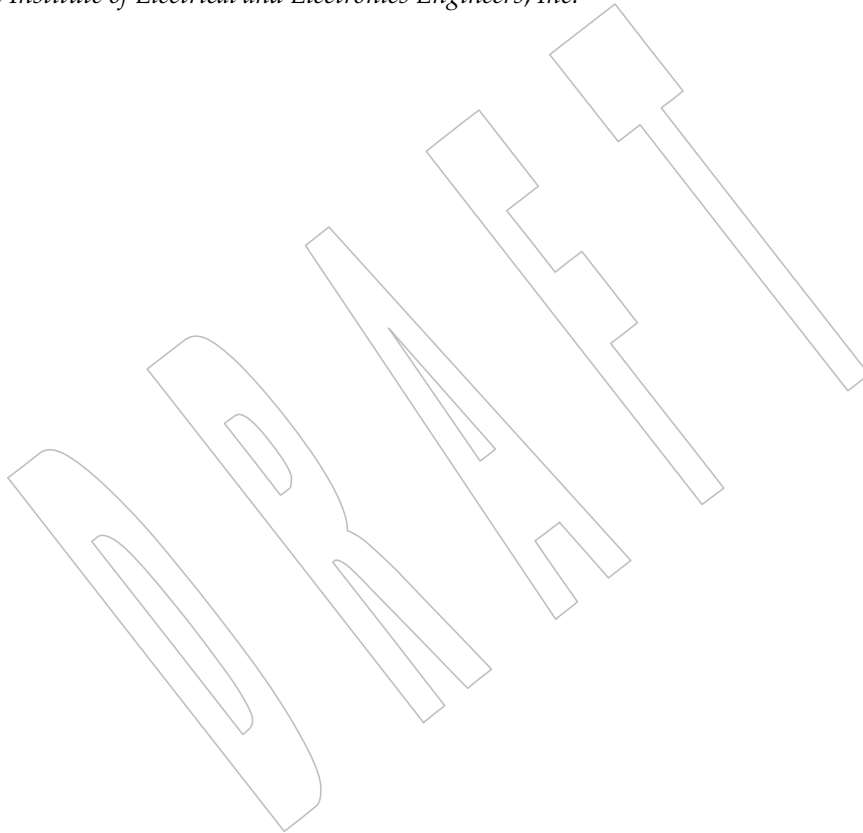
**Shell and Utilities**

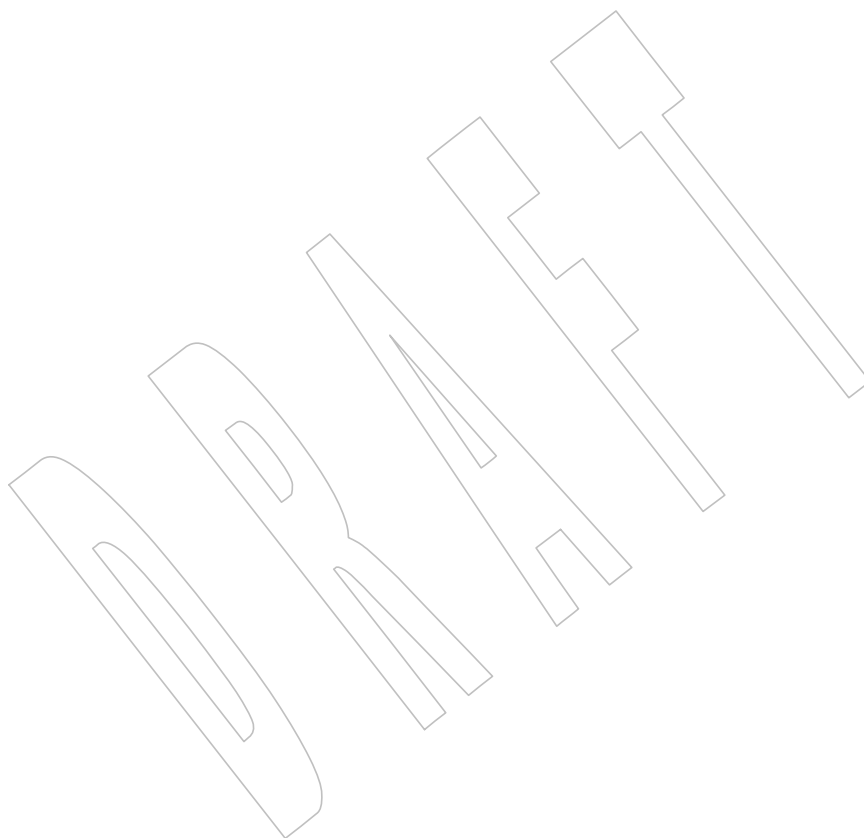
121504

*The Open Group*

121505

*The Institute of Electrical and Electronics Engineers, Inc.*





# Rationale for Shell and Utilities

121506

121507

## C.1 Introduction

121509

### C.1.1 Change History

The change history is provided as an informative section, to track changes from earlier versions of this standard.

The following sections describe changes made to the Shell and Utilities volume of POSIX.1-200x since Issue 6 of the base document. The CHANGE HISTORY section for each utility describes technical changes made to that utility from Issue 5. Changes between earlier versions of the base document and Issue 5 are not included.

#### Changes from Issue 6 to Issue 7 (POSIX.1-200x)

The following list summarizes the major changes that were made in the Shell and Utilities volume of POSIX.1-200x from Issue 6 to Issue 7:

- Austin Group defect reports, IEEE Interpretations against IEEE Std 1003.1, and responses to ISO/IEC defect reports against ISO/IEC 9945 are applied.
- The Open Group corrigenda and resolutions are applied.
- Features, marked legacy or obsolescent in the base document, have been considered for removal in this version.
- A review of the use of fixed path filenames within the standard has been undertaken; for example, the *at*, *batch*, and *crontab* utilities previously had a requirement for use of the directory */usr/lib/cron*.
- The options within the standard have been revised.
  - The Batch Environment Services and Utilities option is marked obsolescent.
  - The UUCP utilities option is added.
  - The User Portability Utilities option is revised so that only the *bg*, *ex*, *fc*, *fg*, *jobs*, *more*, *talk*, and *vi* utilities are included, the rest being moved to the Base.

#### New Features in Issue 7

There are no new utilities in Issue 7.

121534

121535 **C.1.2 Relationship to Other Documents**121536 **C.1.2.1 System Interfaces**

121537 It has been pointed out that the Shell and Utilities volume of POSIX.1-200x assumes that a great  
 121538 deal of functionality from the System Interfaces volume of POSIX.1-200x is present, but never  
 121539 states exactly how much (and strictly does not need to since both are mandated on a conforming  
 121540 system). This section is an attempt to clarify the assumptions.

121541 **File Read, Write, and Creation**

121542 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/2 is applied, updating Table 1-1.

121543 **File Removal**

121544 This is intended to be a summary of the *unlink()* and *rmdir()* requirements. Note that it is  
 121545 possible using the *unlink()* function for item 4. to occur.

121546 **C.1.2.2 Concepts Derived from the ISO C Standard**

121547 This section was introduced to address the issue that there was insufficient detail presented by  
 121548 such utilities as *awk* or *sh* about their procedural control statements and their methods of  
 121549 performing arithmetic functions.

121550 The ISO C standard was selected as a model because most historical implementations of the  
 121551 standard utilities were written in C. Thus, it was more likely that they would act in the desired  
 121552 manner without modification.

121553 Using the ISO C standard is primarily a notational convenience so that the many procedural  
 121554 languages in the Shell and Utilities volume of POSIX.1-200x would not have to be rigorously  
 121555 described in every aspect. Its selection does not require that the standard utilities be written in  
 121556 Standard C; they could be written in Common Usage C, Ada, Pascal, assembler language, or  
 121557 anything else.

121558 The sizes of the various numeric values refer to C-language data types that are allowed to be  
 121559 different sizes by the ISO C standard. Thus, like a C-language application, a shell application  
 121560 cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the  
 121561 ISO C standard, such as {LONG\_MAX} for a **long** type.

121562 The behavior on overflow is undefined for ISO C standard arithmetic. Therefore, the standard  
 121563 utilities can use “bignum” representation for integers so that there is no fixed maximum unless  
 121564 otherwise stated in the utility description. Similarly, standard utilities can use infinite-precision  
 121565 representations for floating-point arithmetic, as long as these representations exceed the ISO C  
 121566 standard requirements.

121567 This section addresses only the issue of semantics; it is not intended to specify syntax. For  
 121568 example, the ISO C standard requires that 0L be recognized as an integer constant equal to zero,  
 121569 but utilities such as *awk* and *sh* are not required to recognize 0L (though they are allowed to, as  
 121570 an extension).

121571 The ISO C standard requires that a C compiler must issue a diagnostic for constants that are too  
 121572 large to represent. Most standard utilities are not required to issue these diagnostics; for  
 121573 example, the command:

121574 `diff -C 2147483648 file1 file2`

121575 has undefined behavior, and the *diff* utility is not required to issue a diagnostic even if the



121576 number 2 147 483 648 cannot be represented.

### 121577 C.1.3 Utility Limits

121578 This section grew out of an idea that originated with the original POSIX.1, in the tables of system  
 121579 limits for the *sysconf()* and *pathconf()* functions. The idea being that a conforming application  
 121580 can be written to use the most restrictive values that a minimal system can provide, but it should  
 121581 not have to. The values provided represent compromises so that some vendors can use  
 121582 historically limited versions of UNIX system utilities. They are the highest values that a strictly  
 121583 conforming application can assume, given no other information.

121584 However, by using the *getconf* utility or the *sysconf()* function, the elegant application can be  
 121585 tailored to more liberal values on some of the specific instances of specific implementations.

121586 There is no explicitly stated requirement that an implementation provide finite limits for any of  
 121587 these numeric values; the implementation is free to provide essentially unbounded capabilities  
 121588 (where it makes sense), stopping only at reasonable points such as {ULONG\_MAX} (from the  
 121589 ISO C standard). Therefore, applications desiring to tailor themselves to the values on a  
 121590 particular implementation need to be ready for possibly huge values; it may not be a good idea  
 121591 to allocate blindly a buffer for an input line based on the value of {LINE\_MAX}, for instance.  
 121592 However, unlike the System Interfaces volume of POSIX.1-200x, there is no set of limits that  
 121593 return a special indication meaning “unbounded”. The implementation should always return an  
 121594 actual number, even if the number is very large.

121595 The statement:

121596 “It is not guaranteed that the application ...”

121597 is an indication that many of these limits are designed to ensure that implementors design their  
 121598 utilities without arbitrary constraints related to unimaginative programming. There are certainly  
 121599 conditions under which combinations of options can cause failures that would not render an  
 121600 implementation non-conforming. For example, {EXPR\_NEST\_MAX} and {ARG\_MAX} could  
 121601 collide when expressions are large; combinations of {BC\_SCALE\_MAX} and {BC\_DIM\_MAX}  
 121602 could exceed virtual memory.

121603 In the Shell and Utilities volume of POSIX.1-200x, the notion of a limit being guaranteed for the  
 121604 process lifetime, as it is in the System Interfaces volume of POSIX.1-200x, is not as useful to a  
 121605 shell script. The *getconf* utility is probably a process itself, so the guarantee would be without  
 121606 value. Therefore, the Shell and Utilities volume of POSIX.1-200x requires the guarantee to be for  
 121607 the session lifetime. This will mean that many vendors will either return very conservative  
 121608 values or possibly implement *getconf* as a built-in.

121609 It may seem confusing to have limits that apply only to a single utility grouped into one global  
 121610 section. However, the alternative, which would be to disperse them out into their utility  
 121611 description sections, would cause great difficulty when *sysconf()* and *getconf* were described.  
 121612 Therefore, the standard developers chose the global approach.

121613 Each language binding could provide symbol names that are slightly different from those shown  
 121614 here. For example, the C-Language Binding option adds a leading underscore to the symbols as  
 121615 a prefix.

121616 The following comments describe selection criteria for the symbols and their values:

121617 {ARG\_MAX}

121618 This is defined by the System Interfaces volume of POSIX.1-200x. Unfortunately, it is very  
 121619 difficult for a conforming application to deal with this value, as it does not know how much  
 121620 of its argument space is being consumed by the environment variables of the user.

- 121621 {BC\_BASE\_MAX}  
 121622 {BC\_DIM\_MAX}  
 121623 {BC\_SCALE\_MAX}  
 121624 These were originally one value, {BC\_SCALE\_MAX}, but it was unreasonable to link all  
 121625 three concepts into one limit.
- 121626 {CHILD\_MAX}  
 121627 This is defined by the System Interfaces volume of POSIX.1-200x.
- 121628 {COLL\_WEIGHTS\_MAX}  
 121629 The weights assigned to **order** can be considered as “passes” through the collation  
 121630 algorithm.
- 121631 {EXPR\_NEST\_MAX}  
 121632 The value for expression nesting was borrowed from the ISO C standard.
- 121633 {LINE\_MAX}  
 121634 This is a global limit that affects all utilities, unless otherwise noted. The {MAX\_CANON}  
 121635 value from the System Interfaces volume of POSIX.1-200x may further limit input lines from  
 121636 terminals. The {LINE\_MAX} value was the subject of much debate and is a compromise  
 121637 between those who wished to have unlimited lines and those who understood that many  
 121638 historical utilities were written with fixed buffers. Frequently, utility writers selected the  
 121639 UNIX system constant BUFSIZ to allocate these buffers; therefore, some utilities were  
 121640 limited to 512 bytes for I/O lines, while others achieved 4096 bytes or greater.
- 121641 It should be noted that {LINE\_MAX} applies only to input line length; there is no  
 121642 requirement in POSIX.1-200x that limits the length of output lines. Utilities such as *awk*, *sed*,  
 121643 and *paste* could theoretically construct lines longer than any of the input lines they received,  
 121644 depending on the options used or the instructions from the application. They are not  
 121645 required to truncate their output to {LINE\_MAX}. It is the responsibility of the application  
 121646 to deal with this. If the output of one of those utilities is to be piped into another of the  
 121647 standard utilities, line length restrictions will have to be considered; the *fold* utility, among  
 121648 others, could be used to ensure that only reasonable line lengths reach utilities or  
 121649 applications.
- 121650 {LINK\_MAX}  
 121651 This is defined by the System Interfaces volume of POSIX.1-200x.
- 121652 {MAX\_CANON}  
 121653 {MAX\_INPUT}  
 121654 {NAME\_MAX}  
 121655 {NGROUPS\_MAX}  
 121656 {OPEN\_MAX}  
 121657 {PATH\_MAX}  
 121658 {PIPE\_BUF}
- 121659 These limits are defined by the System Interfaces volume of POSIX.1-200x. Note that the  
 121660 byte lengths described by some of these values continue to represent bytes, even if the  
 121661 applicable character set uses a multi-byte encoding.
- 121662 {RE\_DUP\_MAX}  
 121663 The value selected is consistent with historical practice. Although the name implies that it  
 121664 applies to all REs, only BREs use the interval notation  $\{m,n\}$  addressed by this limit.
- 121665 {POSIX2\_SYMLINKS}  
 121666 The {POSIX2\_SYMLINKS} variable indicates that the underlying operating system supports  
 121667 the creation of symbolic links in specific directories. Many of the utilities defined in  
 121668 POSIX.1-200x that deal with symbolic links do not depend on this value. For example, a

121669 utility that follows symbolic links (or does not, as the case may be) will only be affected by a  
 121670 symbolic link if it encounters one. Presumably, a file system that does not support symbolic  
 121671 links will not contain any. This variable does affect such utilities as *ln -s* and *pax* that  
 121672 attempt to create symbolic links.

121673 There are different limits associated with command lines and input to utilities, depending on the  
 121674 method of invocation. In the case of a C program *exec*-ing a utility, {ARG\_MAX} is the  
 121675 underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE\_MAX}  
 121676 limits the length of lines the shell is required to process, and {ARG\_MAX} will still be a limit. If a  
 121677 user is entering a command on a terminal to the shell, requesting that it invoke the utility,  
 121678 {MAX\_INPUT} may restrict the length of the line that can be given to the shell to a value below  
 121679 {LINE\_MAX}.

121680 When an option is supported, *getconf* returns a value of 1. For example, when C development is  
 121681 supported:

```
121682 if ["$(getconf POSIX2_C_DEV)" -eq 1]; then
121683 echo C supported
121684 fi
```

121685 The *sysconf()* function in the C-Language Binding option would return 1.

121686 The following comments describe selection criteria for the symbols and their values:

```
121687 POSIX2_C_BIND
121688 POSIX2_C_DEV
121689 POSIX2_FORT_DEV
121690 POSIX2_FORT_RUN
121691 POSIX2_SW_DEV
121692 POSIX2_UPE
```

121693 It is possible for some (usually privileged) operations to remove utilities that support these  
 121694 options or otherwise to render these options unsupported. The header files, the *sysconf()*  
 121695 function, or the *getconf* utility will not necessarily detect such actions, in which case they  
 121696 should not be considered as rendering the implementation non-conforming. A test suite  
 121697 should not attempt tests such as:

```
121698 rm /usr/bin/c99
121699 getconf POSIX2_C_DEV
```

```
121700 POSIX2_LOCALEDEF
```

121701 This symbol was introduced to allow implementations to restrict supported locales to only  
 121702 those supplied by the implementation.

121703 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/2 is applied, deleting the entry for  
 121704 {POSIX2\_VERSION} since it is not a utility limit minimum value.

121705 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/3 is applied, changing the text in Utility  
 121706 Limits from: "utility (see *getconf*) through the *sysconf()* function defined in the System Interfaces  
 121707 volume of POSIX.1-200x. The literal names shown in Table 1-3 apply only to the *getconf* utility;  
 121708 the high-level language binding describes the exact form of each name to be used by the  
 121709 interfaces in that binding." to: "utility (see *getconf*).".

121710 **C.1.4 Grammar Conventions**

121711 There is no additional rationale provided for this section.

121712 **C.1.5 Utility Description Defaults**121713 This section is arranged with headings in the same order as all the utility descriptions. It is a  
121714 collection of related and unrelated information concerning:

- 121715 1. The default actions of utilities
- 
- 121716 2. The meanings of notations used in POSIX.1-200x that are specific to individual utility
- 
- 121717 sections

121718 Although this material may seem out of place here, it is important that this information appear  
121719 before any of the utilities to be described later.121720 **NAME**

121721 There is no additional rationale provided for this section.

121722 **SYNOPSIS**

121723 There is no additional rationale provided for this section.

121724 **DESCRIPTION**

121725 There is no additional rationale provided for this section.

121726 **OPTIONS**121727 Although it has not always been possible, the standard developers tried to avoid repeating  
121728 information to reduce the risk that duplicate explanations could each be modified differently.121729 The need to recognize `--` is required because conforming applications need to shield their  
121730 operands from any arbitrary options that the implementation may provide as an extension. For  
121731 example, if the standard utility *foo* is listed as taking no options, and the application needed to  
121732 give it a pathname with a leading hyphen, it could safely do it as:121733 `foo -- -myfile`121734 and avoid any problems with `-m` used as an extension.121735 **OPERANDS**121736 The usage of `-` is never shown in the SYNOPSIS. Similarly, the usage of `--` is never shown.121737 The requirement for processing operands in command-line order is to avoid a “WeirdNIX”  
121738 utility that might choose to sort the input files alphabetically, by size, or by directory order.  
121739 Although this might be acceptable for some utilities, in general the programmer has a right to  
121740 know exactly what order will be chosen.121741 Some of the standard utilities take multiple *file* operands and act as if they were processing the  
121742 concatenation of those files. For example:121743 `asa file1 file2`

121744 and:

121745 `cat file1 file2 | asa`

121746 have similar results when questions of file access, errors, and performance are ignored. Other

121747 utilities such as *grep* or *wc* have completely different results in these two cases. This latter type of  
 121748 utility is always identified in its DESCRIPTION or OPERANDS sections, whereas the former is  
 121749 not. Although it might be possible to create a general assertion about the former case, the  
 121750 following points must be addressed:

- 121751 • Access times for the files might be different in the operand case *versus* the *cat* case.
- 121752 • The utility may have error messages that are cognizant of the input filename, and this  
 121753 added value should not be suppressed. (As an example, *awk* sets a variable with the  
 121754 filename at each file boundary.)

## 121755 STDIN

121756 There is no additional rationale provided for this section.

## 121757 INPUT FILES

121758 A conforming application cannot assume the following three commands are equivalent:

```
121759 tail -n +2 file
121760 (sed -n 1q; cat) < file
121761 cat file | (sed -n 1q; cat)
```

121762 The second command is equivalent to the first only when the file is seekable. In the third  
 121763 command, if the file offset in the open file description were not unspecified, *sed* would have to  
 121764 be implemented so that it read from the pipe 1 byte at a time or it would have to employ some  
 121765 method to seek backwards on the pipe. Such functionality is not defined currently in POSIX.1  
 121766 and does not exist on all historical systems. Other utilities, such as *head*, *read*, and *sh*, have similar  
 121767 properties, so the restriction is described globally in this section.

121768 The definition of “text file” is strictly enforced for input to the standard utilities; very few of  
 121769 them list exceptions to the undefined results called for here. (Of course, “undefined” here does  
 121770 not mean that historical implementations necessarily have to change to start indicating error  
 121771 conditions. Conforming applications cannot rely on implementations succeeding or failing when  
 121772 non-text files are used.)

121773 The utilities that allow line continuation are generally those that accept input languages, rather  
 121774 than pure data. It would be unusual for an input line of this type to exceed {LINE\_MAX} bytes  
 121775 and unreasonable to require that the implementation allow unlimited accumulation of multiple  
 121776 lines, each of which could reach {LINE\_MAX}. Thus, for a conforming application the total of all  
 121777 the continued lines in a set cannot exceed {LINE\_MAX}.

121778 The format description is intended to be sufficiently rigorous to allow other applications to  
 121779 generate these input files. However, since <blank>s can legitimately be included in some of the  
 121780 fields described by the standard utilities, particularly in locales other than the POSIX locale, this  
 121781 intent is not always realized.

## 121782 ENVIRONMENT VARIABLES

121783 There is no additional rationale provided for this section.

121784 **ASYNCHRONOUS EVENTS**

121785 Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some  
 121786 additional processing (such as deleting temporary files), restore the default signal action (or  
 121787 action inherited from the parent process), and resignal itself.

121788 **STDOUT**

121789 The format description is intended to be sufficiently rigorous to allow post-processing of output  
 121790 by other programs, particularly by an *awk* or *lex* parser.

121791 **STDERR**

121792 This section does not describe error messages that refer to incorrect operation of the utility.  
 121793 Consider a utility that processes program source code as its input. This section is used to  
 121794 describe messages produced by a correctly operating utility that encounters an error in the  
 121795 program source code on which it is processing. However, a message indicating that the utility  
 121796 had insufficient memory in which to operate would not be described.

121797 Some utilities have traditionally produced warning messages without returning a non-zero exit  
 121798 status; these are specifically noted in their sections. Other utilities shall not write to standard  
 121799 error if they complete successfully, unless the implementation provides some sort of extension to  
 121800 increase the verbosity or debugging level.

121801 The format descriptions are intended to be sufficiently rigorous to allow post-processing of  
 121802 output by other programs.

121803 **OUTPUT FILES**

121804 The format description is intended to be sufficiently rigorous to allow post-processing of output  
 121805 by other programs, particularly by an *awk* or *lex* parser.

121806 Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging  
 121807 mode) that would bypass any attempted recovery actions.

121808 **EXTENDED DESCRIPTION**

121809 There is no additional rationale provided for this section.

121810 **EXIT STATUS**

121811 Note the additional discussion of exit values in *Exit Status for Commands* in the *sh* utility. It  
 121812 describes requirements for returning exit values greater than 125.

121813 A utility may list zero as a successful return, 1 as a failure for a specific reason, and greater than  
 121814 1 as “an error occurred”. In this case, unspecified conditions may cause a 2 or 3, or other value,  
 121815 to be returned. A strictly conforming application should be written so that it tests for successful  
 121816 exit status values (zero in this case), rather than relying upon the single specific error value listed  
 121817 in POSIX.1-200x. In that way, it will have maximum portability, even on implementations with  
 121818 extensions.

121819 The standard developers are aware that the general non-enumeration of errors makes it difficult  
 121820 to write test suites that test the *incorrect* operation of utilities. There are some historical  
 121821 implementations that have expended effort to provide detailed status messages and a helpful  
 121822 environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant  
 121823 syntax errors; other implementations have not. Since there is no realistic way to mandate system  
 121824 behavior in cases of undefined application actions or system problems—in a manner acceptable  
 121825 to all cultures and environments—attention has been limited to the correct operation of utilities

121826 by the conforming application. Furthermore, the conforming application does not need detailed  
121827 information concerning errors that it caused through incorrect usage or that it cannot correct.

121828 There is no description of defaults for this section because all of the standard utilities specify  
121829 something (or explicitly state “Unspecified”) for exit status.

#### 121830 **CONSEQUENCES OF ERRORS**

121831 Several actions are possible when a utility encounters an error condition, depending on the  
121832 severity of the error and the state of the utility. Included in the possible actions of various  
121833 utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; and  
121834 validity checking of the file system or directory.

121835 The text about recursive traversing is meant to ensure that utilities such as *find* process as many  
121836 files in the hierarchy as they can. They should not abandon all of the hierarchy at the first error  
121837 and resume with the next command-line operand, but should attempt to keep going.

#### 121838 **APPLICATION USAGE**

121839 This section provides additional caveats, issues, and recommendations to the developer.

#### 121840 **EXAMPLES**

121841 This section provides sample usage.

#### 121842 **RATIONALE**

121843 There is no additional rationale provided for this section.

#### 121844 **FUTURE DIRECTIONS**

121845 FUTURE DIRECTIONS sections act as pointers to related work that may impact the interface in  
121846 the future, and often cautions the developer to architect the code to account for a change in this  
121847 area. Note that a future directions statement should not be taken as a commitment to adopt a  
121848 feature or interface in the future.

#### 121849 **SEE ALSO**

121850 There is no additional rationale provided for this section.

#### 121851 **CHANGE HISTORY**

121852 There is no additional rationale provided for this section.

### 121853 **C.1.6 Considerations for Utilities in Support of Files of Arbitrary Size**

121854 This section is intended to clarify the requirements for utilities in support of large files.

121855 The utilities listed in this section are utilities which are used to perform administrative tasks  
121856 such as to create, move, copy, remove, change the permissions, or measure the resources of a file.  
121857 They are useful both as end-user tools and as utilities invoked by applications during software  
121858 installation and operation.

121859 The *chgrp*, *chmod*, *chown*, *ln*, and *rm* utilities probably require use of large file-capable versions of  
121860 *stat()*, *lstat()*, *ftw()*, and the **stat** structure.

121861 The *cat*, *cksum*, *cmp*, *cp*, *dd*, *mv*, *sum*, and *touch* utilities probably require use of large file-capable  
121862 versions of *creat()*, *open()*, and *fopen()*.

121863 The *cat*, *cksum*, *cmp*, *dd*, *df*, *du*, *ls*, and *sum* utilities may require writing large integer values. For

121864 example:

- 121865 • The *cat* utility might have a `-n` option which counts <newline>s.
- 121866 • The *cksum* and *ls* utilities report file sizes.
- 121867 • The *cmp* utility reports the line number at which the first difference occurs, and also has a
- 121868 `-l` option which reports file offsets.
- 121869 • The *dd*, *df*, *du*, *ls*, and *sum* utilities report block counts.

121870 The *dd*, *find*, and *test* utilities may need to interpret command arguments that contain 64-bit  
 121871 values. For *dd*, the arguments include `skip=n`, `seek=n`, and `count=n`. For *find*, the arguments  
 121872 include `-sizen`. For *test*, the arguments are those associated with algebraic comparisons.

121873 The *df* utility might need to access large file systems with *statofs*().

121874 The *ulimit* utility will need to use large file-capable versions of *getrlimit*() and *setrlimit*() and be  
 121875 able to read and write large integer values.

### 121876 C.1.7 Built-In Utilities

121877 All of these utilities can be *exec*-ed. There is no requirement that these utilities are actually built  
 121878 into the shell itself, but many shells need the capability to do so because XCU Section 2.9.1.1 (on  
 121879 page 2264) requires that they be found prior to the *PATH* search. The shell could satisfy its  
 121880 requirements by keeping a list of the names and directly accessing the file-system versions  
 121881 regardless of *PATH*. Providing all of the required functionality for those such as *cd* or *read* would  
 121882 be more difficult.

121883 There were originally three justifications for allowing the omission of *exec*-able versions:

- 121884 1. It would require wasting space in the file system, at the expense of very small systems.  
 121885 However, it has been pointed out that all 16 utilities in the table can be provided with 16  
 121886 links to a single-line shell script:

```
121887 $0 "$@"
```

- 121888 2. It is not logical to require invocation of utilities such as *cd* because they have no value  
 121889 outside the shell environment or cannot be useful in a child process. However, counter-  
 121890 examples always seemed to be available for even the most unusual cases:

```
121891 find . -type d -exec cd {} \; -exec foo {} \;

 121892 (which invokes "foo" on accessible directories)
```

```
121893 ps ... | sed ... | xargs kill
```

```
121894 find . -exec true \; -a ...

 121895 (where "true" is used for temporary debugging)
```

- 121896 3. It is confusing to have a utility such as *kill* that can easily be in the file system in the base  
 121897 standard, but that requires built-in status for the User Portability Utilities option (for the  
 121898 % job control job ID notation). It was decided that it was more appropriate to describe the  
 121899 required functionality (rather than the implementation) to the system implementors and  
 121900 let them decide how to satisfy it.

121901 On the other hand, it was realized that any distinction like this between utilities was not useful  
 121902 to applications, and that the cost to correct it was small. These arguments were ultimately the  
 121903 most effective.

121904 There were varying reasons for including utilities in the table of built-ins:



- 121905 *alias, fc, unalias*  
 121906 The functionality of these utilities is performed more simply within the shell itself and that  
 121907 is the model most historical implementations have used.
- 121908 *bg, fg, jobs*  
 121909 All of the job control-related utilities are eligible for built-in status because that is the model  
 121910 most historical implementations have used.
- 121911 *cd, getopts, newgrp, read, umask, wait*  
 121912 The functionality of these utilities is performed more simply within the context of the  
 121913 current process. An example can be taken from the usage of the *cd* utility. The purpose of  
 121914 the *cd* utility is to change the working directory for subsequent operations. The actions of *cd*  
 121915 affect the process in which *cd* is executed and all subsequent child processes of that process.  
 121916 Based on the POSIX standard process model, changes in the process environment of a child  
 121917 process have no effect on the parent process. If the *cd* utility were executed from a child  
 121918 process, the working directory change would be effective only in the child process. Child  
 121919 processes initiated subsequent to the child process that executed the *cd* utility would not  
 121920 have a changed working directory relative to the parent process.
- 121921 *command*  
 121922 This utility was placed in the table primarily to protect scripts that are concerned about  
 121923 their *PATH* being manipulated. The "secure" shell script example in the *command* utility in  
 121924 the Shell and Utilities volume of POSIX.1-200x would not be possible if a *PATH* change  
 121925 retrieved an alien version of *command*. (An alternative would have been to implement  
 121926 *getconf* as a built-in, but the standard developers considered that it carried too many  
 121927 changing configuration strings to require in the shell.)
- 121928 *kill* Since *kill* provides optional job control functionality using shell notation (%1, %2, and so on),  
 121929 some implementations would find it extremely difficult to provide this outside the shell.
- 121930 *true, false*  
 121931 These are in the table as a courtesy to programmers who wish to use the "while true"  
 121932 shell construct without protecting *true* from *PATH* searches. (It is acknowledged that  
 121933 "while : " also works, but the idiom with *true* is historically pervasive.)
- 121934 All utilities, including those in the table, are accessible via the *system()* and *popen()* functions in  
 121935 the System Interfaces volume of POSIX.1-200x. There are situations where the return  
 121936 functionality of *system()* and *popen()* is not desirable. Applications that require the exit status of  
 121937 the invoked utility will not be able to use *system()* or *popen()*, since the exit status returned is  
 121938 that of the command language interpreter rather than that of the invoked utility. The alternative  
 121939 for such applications is the use of the *exec* family.

## 121940 C.2 Shell Command Language

### 121941 C.2.1 Shell Introduction

121942 The System V shell was selected as the starting point for the Shell and Utilities volume of  
121943 POSIX.1-200x. The BSD C shell was excluded from consideration for the following reasons:

- 121944 • Most historically portable shell scripts assume the Version 7 Bourne shell, from which the  
121945 System V shell is derived.
- 121946 • The majority of tutorial materials on shell programming assume the System V shell.

121947 The construct "#!" is reserved for implementations wishing to provide that extension. If it were  
121948 not reserved, the Shell and Utilities volume of POSIX.1-200x would disallow it by forcing it to be  
121949 a comment. As it stands, a strictly conforming application must not use "#!" as the first two  
121950 characters of the file.

### 121951 C.2.2 Quoting

121952 There is no additional rationale provided for this section.

#### 121953 C.2.2.1 Escape Character (Backslash)

121954 There is no additional rationale provided for this section.

#### 121955 C.2.2.2 Single-Quotes

121956 A backslash cannot be used to escape a single-quote in a single-quoted string. An embedded  
121957 quote can be created by writing, for example: "'a'\''b'", which yields "a'b". (See XCU  
121958 [Section 2.6.5](#) (on page 2258) for a better understanding of how portions of words are either split  
121959 into fields or remain concatenated.) A single token can be made up of concatenated partial  
121960 strings containing all three kinds of quoting or escaping, thus permitting any combination of  
121961 characters.

#### 121962 C.2.2.3 Double-Quotes

121963 The escaped <newline> used for line continuation is removed entirely from the input and is not  
121964 replaced by any white space. Therefore, it cannot serve as a token separator.

121965 In double-quoting, if a backslash is immediately followed by a character that would be  
121966 interpreted as having a special meaning, the backslash is deleted and the subsequent character is  
121967 taken literally. If a backslash does not precede a character that would have a special meaning, it  
121968 is left in place unmodified and the character immediately following it is also left unmodified.  
121969 Thus, for example:

121970 "\\$" → \$

121971 "\a" → \a

121972 It would be desirable to include the statement "The characters from an enclosed "\${" to the  
121973 matching '}' shall not be affected by the double quotes", similar to the one for "\$()".  
121974 However, historical practice in the System V shell prevents this.

121975 The requirement that double-quotes be matched inside "\${...}" within double-quotes and the  
121976 rule for finding the matching '}' in XCU [Section 2.6.2](#) (on page 2254) eliminate several subtle  
121977 inconsistencies in expansion for historical shells in rare cases; for example:

121978 "\${foo-bar }

121979 yields **bar** when **foo** is not defined, and is an invalid substitution when **foo** is defined, in many  
 121980 historical shells. The differences in processing the "\${...}" form have led to inconsistencies  
 121981 between historical systems. A consequence of this rule is that single-quotes cannot be used to  
 121982 quote the '}' within "\${...}"; for example:

```
121983 unset bar
121984 foo="${bar-'}'"
```

121985 is invalid because the "\${...}" substitution contains an unpaired unescaped single-quote. The  
 121986 backslash can be used to escape the '}' in this example to achieve the desired result:

```
121987 unset bar
121988 foo="${bar-\}]"
```

121989 The differences in processing the "\${...}" form have led to inconsistencies between the  
 121990 historical System V shell, BSD, and KornShells, and the text in the Shell and Utilities volume of  
 121991 POSIX.1-200x is an attempt to converge them without breaking too many applications. The only  
 121992 alternative to this compromise between shells would be to make the behavior unspecified  
 121993 whenever the literal characters ' ', '{', '}', and '"' appear within "\${...}". To write a  
 121994 portable script that uses these values, a user would have to assign variables; for example:

```
121995 squote=\` dquote=\" lbrace='{` rbrace='}'
121996 ${foo-$squote$rbrace$squote}
```

121997 rather than:

```
121998 ${foo-"'}'"}
```

121999 Some implementations have allowed the end of the word to terminate the backquoted command  
 122000 substitution, such as in:

```
122001 "`echo hello"
```

122002 This usage is undefined; the matching backquote is required by the Shell and Utilities volume of  
 122003 POSIX.1-200x. The other undefined usage can be illustrated by the example:

```
122004 sh -c '` echo "foo`'
```

122005 The description of the recursive actions involving command substitution can be illustrated with  
 122006 an example. Upon recognizing the introduction of command substitution, the shell parses input  
 122007 (in a new context), gathering the source for the command substitution until an unbalanced ')' or  
 122008 or '`' is located. For example, in the following:

```
122009 echo "$(date; echo "

122010 one")"
```

122011 the double-quote following the *echo* does not terminate the first double-quote; it is part of the  
 122012 command substitution script. Similarly, in:

```
122013 echo "$(echo *)"
```

122014 the asterisk is not quoted since it is inside command substitution; however:

```
122015 echo "$(echo "*")"
```

122016 is quoted (and represents the asterisk character itself).

122017 **C.2.3 Token Recognition**

122018 The "(" and ")" symbols are control operators in the KornShell, used for an alternative  
 122019 syntax of an arithmetic expression command. A conforming application cannot use "(" as a  
 122020 single token (with the exception of the "\$(" form for shell arithmetic).

122021 On some implementations, the symbol "(" is a control operator; its use produces unspecified  
 122022 results. Applications that wish to have nested subshells, such as:

```
122023 ((echo Hello);(echo World))
```

122024 must separate the "(" characters into two tokens by including white space between them.  
 122025 Some systems may treat these as invalid arithmetic expressions instead of subshells.

122026 Certain combinations of characters are invalid in portable scripts, as shown in the grammar.  
 122027 Implementations may use these combinations (such as "|&") as valid control operators. Portable  
 122028 scripts cannot rely on receiving errors in all cases where this volume of POSIX.1-200x indicates  
 122029 that a syntax is invalid.

122030 The (3) rule about combining characters to form operators is not meant to preclude systems from  
 122031 extending the shell language when characters are combined in otherwise invalid ways.  
 122032 Conforming applications cannot use invalid combinations, and test suites should not penalize  
 122033 systems that take advantage of this fact. For example, the unquoted combination "|&" is not  
 122034 valid in a POSIX script, but has a specific KornShell meaning.

122035 The (10) rule about '#' as the current character is the first in the sequence in which a new token  
 122036 is being assembled. The '#' starts a comment only when it is at the beginning of a token. This  
 122037 rule is also written to indicate that the search for the end-of-comment does not consider escaped  
 122038 <newline> specially, so that a comment cannot be continued to the next line.

122039 **C.2.3.1 Alias Substitution**

122040 The alias capability was added because it is widely used in historical implementations by  
 122041 interactive users.

122042 The definition of "alias name" precludes an alias name containing a slash character. Since the  
 122043 text applies to the command words of simple commands, reserved words (in their proper  
 122044 places) cannot be confused with aliases.

122045 The placement of alias substitution in token recognition makes it clear that it precedes all of the  
 122046 word expansion steps.

122047 An example concerning trailing <blank>s and reserved words follows. If the user types:

```
122048 $ alias foo="/bin/ls "

 122049 $ alias while="/ "
```

122050 The effect of executing:

```
122051 $ while true

 122052 > do

 122053 > echo "Hello, World"

 122054 > done
```

122055 is a never-ending sequence of "Hello, World" strings to the screen. However, if the user  
 122056 types:

```
122057 $ foo while
```

122058 the result is an *ls* listing of /. Since the alias substitution for **foo** ends in a <space>, the next word  
 122059 is checked for alias substitution. The next word, **while**, has also been aliased, so it is substituted

122060 as well. Since it is not in the proper position as a command word, it is not recognized as a  
122061 reserved word.

122062 If the user types:

122063 `$ foo; while`

122064 `while` retains its normal reserved-word properties.

## 122065 C.2.4 Reserved Words

122066 All reserved words are recognized syntactically as such in the contexts described. However, note  
122067 that `in` is the only meaningful reserved word after a `case` or `for`; similarly, `in` is not meaningful as  
122068 the first word of a simple command.

122069 Reserved words are recognized only when they are delimited (that is, meet the definition of XBD  
122070 Section 3.438, on page 94), whereas operators are themselves delimiters. For instance, `' ( ' and  
122071 ' ) '` are control operators, so that no `<space>` is needed in `(list)`. However, `' { ' and ' } '`  
122072 are reserved words in `{ list; }`, so that in this case the leading `<space>` and semicolon are required.

122073 The list of unspecified reserved words is from the KornShell, so conforming applications cannot  
122074 use them in places a reserved word would be recognized. This list contained `time` in early  
122075 proposals, but it was removed when the `time` utility was selected for the Shell and Utilities  
122076 volume of POSIX.1-200x.

122077 There was a strong argument for promoting braces to operators (instead of reserved words), so  
122078 they would be syntactically equivalent to subshell operators. Concerns about compatibility  
122079 outweighed the advantages of this approach. Nevertheless, conforming applications should  
122080 consider quoting `' { ' and ' } '` when they represent themselves.

122081 The restriction on ending a name with a colon is to allow future implementations that support  
122082 named labels for flow control; see the RATIONALE for the `break` built-in utility.

122083 It is possible that a future version of the Shell and Utilities volume of POSIX.1-200x may require  
122084 that `' { ' and ' } '` be treated individually as control operators, although the token `" { } "` will  
122085 probably be a special-case exemption from this because of the often-used `find{ }` construct.

## 122086 C.2.5 Parameters and Variables

### 122087 C.2.5.1 Positional Parameters

122088 There is no additional rationale provided for this section.

### 122089 C.2.5.2 Special Parameters

122090 Most historical implementations implement subshells by forking; thus, the special parameter  
122091 `' $ '`  does not necessarily represent the process ID of the shell process executing the commands  
122092 since the subshell execution environment preserves the value of `' $ '` .

122093 If a subshell were to execute a background command, the value of `" $! "` for the parent would  
122094 not change. For example:

```
122095 (
122096 date &
122097 echo $!
122098)
122099 echo $!
```

122100 would echo two different values for "\$!".

122101 The "\$-" special parameter can be used to save and restore *set* options:

```
122102 Save=$(echo $- | sed 's/[ics]//g')
122103 ...
122104 set +aCefnuvx
122105 if [-n "$Save"]; then
122106 set -$Save
122107 fi
```

122108 The three options are removed using *sed* in the example because they may appear in the value of  
122109 "\$-" (from the *sh* command line), but are not valid options to *set*.

122110 The descriptions of parameters '\*' and '@' assume the reader is familiar with the field |  
122111 splitting discussion in XCU Section 2.6.5 (on page 2258) and understands that portions of the |  
122112 word remain concatenated unless there is some reason to split them into separate fields.

122113 Some examples of the '\*' and '@' properties, including the concatenation aspects:

```
122114 set "abc" "def ghi" "jkl"
122115 echo $* => "abc" "def" "ghi" "jkl"
122116 echo "$*" => "abc def ghi jkl"
122117 echo $@ => "abc" "def" "ghi" "jkl"
```

122118 but:

```
122119 echo "$@" => "abc" "def ghi" "jkl"
122120 echo "xx$@yy" => "xxabc" "def ghi" "jkl"yy"
122121 echo "$@$@" => "abc" "def ghi" "jklabc" "def ghi" "jkl"
```

122122 In the preceding examples, the double-quote characters that appear after the "=>" do not  
122123 appear in the output and are used only to illustrate word boundaries.

122124 The following example illustrates the effect of setting *IFS* to a null string:

```
122125 $ IFS=
122126 $ set foo bar bam
122127 $ echo "$@"
122128 foo bar bam
122129 $ echo "$*"
122130 foobarbam
122131 $ unset IFS
122132 $ echo "$*"
122133 foo bar bam
```

### 122134 C.2.5.3 Shell Variables

122135 See the discussion of *IFS* in Section C.2.6.5 (on page 3557) and the RATIONALE for the *sh* utility.

122136 The prohibition on *LC\_CTYPE* changes affecting lexical processing protects the shell  
122137 implementor (and the shell programmer) from the ill effects of changing the definition of  
122138 <blank> or the set of alphabetic characters in the current environment. It would probably not be  
122139 feasible to write a compiled version of a shell script without this rule. The rule applies only to  
122140 the current invocation of the shell and its subshells—invoking a shell script or performing *exec*  
122141 *sh* would subject the new shell to the changes in *LC\_CTYPE*.

122142 Other common environment variables used by historical shells are not specified by the Shell and  
122143 Utilities volume of POSIX.1-200x, but they should be reserved for the historical uses.

|        |                |                                                                                                         |
|--------|----------------|---------------------------------------------------------------------------------------------------------|
| 122144 |                | Tilde expansion for components of <i>PATH</i> in an assignment such as:                                 |
| 122145 |                | <code>PATH=~hlj/bin:~dwc/bin:\$PATH</code>                                                              |
| 122146 |                | is a feature of some historical shells and is allowed by the wording of XCU Section 2.6.1 (on page      |
| 122147 |                | 2253). Note that the tildes are expanded during the assignment to <i>PATH</i> , not when <i>PATH</i> is |
| 122148 |                | accessed during command search.                                                                         |
| 122149 |                | The following entries represent additional information about variables included in the Shell and        |
| 122150 |                | Utilities volume of POSIX.1-200x, or rationale for common variables in use by shells that have          |
| 122151 |                | been excluded:                                                                                          |
| 122152 | —              | (Underscore.) While underscore is historical practice, its overloaded usage in                          |
| 122153 |                | the KornShell is confusing, and it has been omitted from the Shell and Utilities                        |
| 122154 |                | volume of POSIX.1-200x.                                                                                 |
| 122155 | <i>ENV</i>     | This variable can be used to set aliases and other items local to the invocation                        |
| 122156 |                | of a shell. The file referred to by <i>ENV</i> differs from <i>\$HOME/.profile</i> in that              |
| 122157 |                | <i>.profile</i> is typically executed at session start-up, whereas the <i>ENV</i> file is               |
| 122158 |                | executed at the beginning of each shell invocation. The <i>ENV</i> value is                             |
| 122159 |                | interpreted in a manner similar to a dot script, in that the commands are                               |
| 122160 |                | executed in the current environment and the file needs to be readable, but not                          |
| 122161 |                | executable. However, unlike dot scripts, no <i>PATH</i> searching is performed. This                    |
| 122162 |                | is used as a guard against Trojan Horse security breaches.                                              |
| 122163 | <i>ERRNO</i>   | This variable was omitted from the Shell and Utilities volume of POSIX.1-200x                           |
| 122164 |                | because the values of error numbers are not defined in POSIX.1-200x in a                                |
| 122165 |                | portable manner.                                                                                        |
| 122166 | <i>FCEDIT</i>  | Since this variable affects only the <i>fc</i> utility, it has been omitted from this more              |
| 122167 |                | global place. The value of <i>FCEDIT</i> does not affect the command-line editing                       |
| 122168 |                | mode in the shell; see the description of <i>set -o vi</i> in the <i>set</i> built-in utility.          |
| 122169 | <i>PS1</i>     | This variable is used for interactive prompts. Historically, the “superuser”                            |
| 122170 |                | has had a prompt of ‘#’. Since privileges are not required to be monolithic, it                         |
| 122171 |                | is difficult to define which privileges should cause the alternate prompt.                              |
| 122172 |                | However, a sufficiently powerful user should be reminded of that power by                               |
| 122173 |                | having an alternate prompt.                                                                             |
| 122174 | <i>PS3</i>     | This variable is used by the KornShell for the <i>select</i> command. Since the POSIX                   |
| 122175 |                | shell does not include <i>select</i> , <i>PS3</i> was omitted.                                          |
| 122176 | <i>PS4</i>     | This variable is used for shell debugging. For example, the following script:                           |
| 122177 |                | <code>PS4=' [ \${LINENO} ]+ '</code>                                                                    |
| 122178 |                | <code>set -x</code>                                                                                     |
| 122179 |                | <code>echo Hello</code>                                                                                 |
| 122180 |                | writes the following to standard error:                                                                 |
| 122181 |                | <code>[3]+ echo Hello</code>                                                                            |
| 122182 | <i>RANDOM</i>  | This pseudo-random number generator was not seen as being useful to                                     |
| 122183 |                | interactive users.                                                                                      |
| 122184 | <i>SECONDS</i> | Although this variable is sometimes used with <i>PS1</i> to allow the display of the                    |
| 122185 |                | current time in the prompt of the user, it is not one that would be manipulated                         |
| 122186 |                | frequently enough by an interactive user to include in the Shell and Utilities                          |
| 122187 |                | volume of POSIX.1-200x.                                                                                 |

## 122188 C.2.6 Word Expansions

122189 Step (2) refers to the “portions of fields generated by step (1)”. For example, if the word being  
 122190 expanded were "\$x+\$y" and IFS=+, the word would be split only if "\$x" or "\$y" contained  
 122191 '+'; the '+' in the original word was not generated by step (1).

122192 IFS is used for performing field splitting on the results of parameter and command substitution;  
 122193 it is not used for splitting all fields. Earlier versions of the shell used it for splitting all fields  
 122194 during field splitting, but this has severe problems because the shell can no longer parse its own  
 122195 script. There are also important security implications caused by this behavior. All useful  
 122196 applications of IFS use it for parsing input of the *read* utility and for splitting the results of  
 122197 parameter and command substitution.

122198 The rule concerning expansion to a single field requires that if **foo=abc** and **bar=def**, that:

```
122199 "$foo" "$bar"
```

122200 expands to the single field:

```
122201 abcdef
```

122202 The rule concerning empty fields can be illustrated by:

```
122203 $ unset foo
122204 $ set $foo bar ' ' xyz "$foo" abc
122205 $ for i
122206 > do
122207 > echo "-$i-"
122208 > done
122209 -bar-
122210 --
122211 -xyz-
122212 --
122213 -abc-
```

122214 Step (1) indicates that parameter expansion, command substitution, and arithmetic expansion  
 122215 are all processed simultaneously as they are scanned. For example, the following is valid  
 122216 arithmetic:

```
122217 x=1
122218 echo $(($(echo 3)+$x))
```

122219 An early proposal stated that tilde expansion preceded the other steps, but this is not the case in  
 122220 known historical implementations; if it were, and if a referenced home directory contained a '\$'  
 122221 character, expansions would result within the directory name.

### 122222 C.2.6.1 Tilde Expansion

122223 Tilde expansion generally occurs only at the beginning of words, but an exception based on  
 122224 historical practice has been included:

```
122225 PATH=/posix/bin:~djk/bin
```

122226 This is eligible for tilde expansion because tilde follows a colon and none of the relevant  
 122227 characters is quoted. Consideration was given to prohibiting this behavior because any of the  
 122228 following are reasonable substitutes:

```
122229 PATH=$(printf %s ~karels/bin : ~bostic/bin)
122230 for Dir in ~maat/bin ~srb/bin ...
122231 do
```



122232           PATH=\${PATH:+\$PATH:}\$Dir  
122233 done

122234 In the first command, explicit colons are used for each directory. In all cases, the shell performs  
122235 tilde expansion on each directory because all are separate words to the shell.

122236 Note that expressions in operands such as:

122237 make -k mumble LIBDIR=~chet/lib

122238 do not qualify as shell variable assignments, and tilde expansion is not performed (unless the  
122239 command does so itself, which *make* does not).

122240 Because of the requirement that the word is not quoted, the following are not equivalent; only  
122241 the last causes tilde expansion:

122242 \~hlj/   ~h\lj/   ~"hlj"/   ~hlj\   ~hlj/

122243 In an early proposal, tilde expansion occurred following any unquoted equals sign or colon, but  
122244 this was removed because of its complexity and to avoid breaking commands such as:

122245 rcp hostname:~marc/.profile .

122246 A suggestion was made that the special sequence "\$~" should be allowed to force tilde  
122247 expansion anywhere. Since this is not historical practice, it has been left for future  
122248 implementations to evaluate. (The description in XCU [Section 2.2](#) (on page 2246) requires that a  
122249 dollar sign be quoted to represent itself, so the "\$~" combination is already unspecified.)

122250 The results of giving tilde with an unknown login name are undefined because the KornShell  
122251 "~+" and "~-" constructs make use of this condition, but in general it is an error to give an  
122252 incorrect login name with tilde. The results of having *HOME* unset are unspecified because  
122253 some historical shells treat this as an error.

### 122254 C.2.6.2 Parameter Expansion

122255 The rule for finding the closing '}' in "\${...}" is the one used in the KornShell and is  
122256 upwardly-compatible with the Bourne shell, which does not determine the closing '}' until the  
122257 word is expanded. The advantage of this is that incomplete expansions, such as:

122258 \${foo

122259 can be determined during tokenization, rather than during expansion.

122260 The string length and substring capabilities were included because of the demonstrated need for  
122261 them, based on their usage in other shells, such as C shell and KornShell.

122262 Historical versions of the KornShell have not performed tilde expansion on the word part of  
122263 parameter expansion; however, it is more consistent to do so.

### 122264 C.2.6.3 Command Substitution

122265 The "\$()" form of command substitution solves a problem of inconsistent behavior when using  
122266 backquotes. For example:

| Command              | Output |
|----------------------|--------|
| echo '\\$x'          | \\$x   |
| echo `echo '\\$x'`   | \$x    |
| echo \$(echo '\\$x') | \\$x   |

122271 Additionally, the backquoted syntax has historical restrictions on the contents of the embedded  
122272 command. While the newer "\$()" form can process any kind of valid embedded script, the

122273 backquoted form cannot handle some valid scripts that include backquotes. For example, these  
 122274 otherwise valid embedded scripts do not work in the left column, but do work on the right:

|        |                             |                             |
|--------|-----------------------------|-----------------------------|
| 122275 | echo `                      | echo \$(                    |
| 122276 | cat <<\eof                  | cat <<\eof                  |
| 122277 | a here-doc with `           | a here-doc with )           |
| 122278 | eof                         | eof                         |
| 122279 | `                           | )                           |
| 122280 | echo `                      | echo \$(                    |
| 122281 | echo abc # a comment with ` | echo abc # a comment with ) |
| 122282 | `                           | )                           |
| 122283 | echo `                      | echo \$(                    |
| 122284 | echo ` `                    | echo ` )                    |
| 122285 | `                           | )                           |

122286 Because of these inconsistent behaviors, the backquoted variety of command substitution is not  
 122287 recommended for new applications that nest command substitutions or attempt to embed  
 122288 complex scripts.

122289 The KornShell feature:

122290 If *command* is of the form `<word>`, *word* is expanded to generate a pathname, and the value  
 122291 of the command substitution is the contents of this file with any trailing `<newline>`  
 122292 deleted.

122293 was omitted from the Shell and Utilities volume of POSIX.1-200x because `$(cat word)` is an  
 122294 appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it  
 122295 is unspecified to have a script within `"$( )"` that has only redirections.

122296 The requirement to separate `"$( "` and `'('` when a single subshell is command-substituted is to  
 122297 avoid any ambiguities with arithmetic expansion.

122298 IEEE Std 1003.1-2001/Cor 1-2002, item XCU/TC1/D6/4 is applied, changing the text from: "If a  
 122299 command substitution occurs inside double-quotes, it shall not be performed on the results of  
 122300 the substitution." to: "If a command substitution occurs inside double-quotes, field splitting and  
 122301 pathname expansion shall not be performed on the results of the substitution.". The  
 122302 replacement text taken from the ISO POSIX-2:1993 standard is clearer about the items that are  
 122303 not performed.

122304 SD5-XCU-ERN-84 is applied, clarifying how the search for the matching backquote is satisfied.

#### 122305 C.2.6.4 Arithmetic Expansion

122306 The standard developers agreed that there was a strong desire for some kind of arithmetic  
 122307 evaluator to provide functionality similar to *expr*, that relating it to `'$'` makes it work well with  
 122308 the standard shell language and provides access to arithmetic evaluation in places where  
 122309 accessing a utility would be inconvenient.

122310 The syntax and semantics for arithmetic were revised for the ISO/IEC 9945-2:1993 standard.  
 122311 The language represents a simple subset of the previous arithmetic language (which was  
 122312 derived from the KornShell `"( )"` construct). The syntax was changed from that of a  
 122313 command denoted by `((expression))` to an expansion denoted by `$(expression)`. The new form is  
 122314 a dollar expansion `'$'` that evaluates the expression and substitutes the resulting value.  
 122315 Objections to the previous style of arithmetic included that it was too complicated, did not fit in  
 122316 well with the use of variables in the shell, and its syntax conflicted with subshells. The  
 122317 justification for the new syntax is that the shell is traditionally a macro language, and if a new

122318 feature is to be added, it should be accomplished by extending the capabilities presented by the  
 122319 current model of the shell, rather than by inventing a new one outside the model; adding a new  
 122320 dollar expansion was perceived to be the most intuitive and least destructive way to add such a  
 122321 new capability.

122322 The standard requires assignment operators to be supported (as listed in XCU Section 1.1.2, on  
 122323 page 2231), and since arithmetic expansions are not specified to be evaluated in a subshell  
 122324 environment, changes to variables there have to be in effect after the arithmetic expansion, just  
 122325 as in the parameter expansion "\$ {x=value}".

122326 Note, however, that "\$ (( x=5 ))" need not be equivalent to "\$ (( \$x=5 ))". If the value of  
 122327 the environment variable *x* is the string "y=", the expansion of "\$ (( x=5 ))" would set *x* to 5  
 122328 and output 5, but "\$ (( \$x=5 ))" would output 0 if the value of the environment variable *y* is  
 122329 not 5 and would output 1 if the environment variable *y* is 5. Similarly, if the value of the  
 122330 environment variable is 4, the expansion of "\$ (( x=5 ))" would still set *x* to 5 and output 5,  
 122331 but "\$ (( \$x=5 ))" (which would be equivalent to "\$ (( 4=5 ))") would yield a syntax  
 122332 error.

122333 In early proposals, a form `$(expression)` was used. It was functionally equivalent to the "\$ (( ))"  
 122334 of the current text, but objections were lodged that the 1988 KornShell had already implemented  
 122335 "\$ (( ))" and there was no compelling reason to invent yet another syntax. Furthermore, the  
 122336 "\$ [ ]" syntax had a minor incompatibility involving the patterns in **case** statements.

122337 The portion of the ISO C standard arithmetic operations selected corresponds to the operations  
 122338 historically supported in the KornShell. In addition to the exceptions listed in XCU Section 2.6.4  
 122339 (on page 2257), the use of the following are explicitly outside the scope of the rules defined in  
 122340 XCU Section 1.1.2.1 (on page 2231):

- The prefix operator '&' and the "[ ]", "->", and '.' operators.
- Casts

122343 It was concluded that the *test* command (I) was sufficient for the majority of relational arithmetic  
 122344 tests, and that tests involving complicated relational expressions within the shell are rare, yet  
 122345 could still be accommodated by testing the value of "\$ (( ))" itself. For example:

```
122346 # a complicated relational expression
122347 while ["$(((($x + $y)/($a * $b)) < ($foo*$bar)))" -ne 0]
```

122348 or better yet, the rare script that has many complex relational expressions could define a  
 122349 function like this:

```
122350 val() {
122351 return "$(!$1)"
122352 }
122353
```

and complicated tests would be less intimidating:

```
122354 while val "$(((($x + $y)/($a * $b)) < ($foo*$bar)))"
122355 do
122356 # some calculations
122357 done
```

122358 A suggestion that was not adopted was to modify *true* and *false* to take an optional argument,  
 122359 and *true* would exit true only if the argument was non-zero, and *false* would exit false only if the  
 122360 argument was non-zero:

```
122361 while true "$(($x > 5 && $y <= 25))"
```

122362 There is a minor portability concern with the new syntax. The example "\$ (( (2+2) ))" could have  
 122363 been intended to mean a command substitution of a utility named "2+2" in a subshell. The

122364 standard developers considered this to be obscure and isolated to some KornShell scripts  
 122365 (because "\$()" command substitution existed previously only in the KornShell). The text on  
 122366 command substitution requires that the "\$(" and '((' be separate tokens if this usage is  
 122367 needed.

122368 An example such as:

122369 `echo $((echo hi);(echo there))`

122370 should not be misinterpreted by the shell as arithmetic because attempts to balance the  
 122371 parentheses pairs would indicate that they are subshells. However, as indicated by XBD [Section](#)  
 122372 [3.112](#) (on page 49), a conforming application must separate two adjacent parentheses with white  
 122373 space to indicate nested subshells.

122374 The standard is intentionally silent about how a variable's numeric value in an expression is  
 122375 determined from its normal "sequence of bytes" value. It could be done as a text substitution, as  
 122376 a conversion like that performed by `strtol()`, or even recursive evaluation. Therefore, the only  
 122377 cases for which the standard is clear are those for which both conversions produce the same  
 122378 result. The cases where they give the same result are those where the sequence of bytes form a  
 122379 valid integer constant. Therefore, if a variable does not contain a valid integer constant, the  
 122380 behavior is unspecified.

122381 For the commands:

122382 `x=010; echo $((x += 1))`

122383 the output must be 9.

122384 For the commands:

122385 `x=' 1'; echo $((x += 1))`

122386 the results are unspecified.

122387 For the commands:

122388 `x=1+1; echo $((x += 1))`

122389 the results are unspecified.

122390 Although the ISO/IEC 9899:1999 standard now requires support for **long long** and allows  
 122391 extended integer types with higher ranks, POSIX.1-200x only requires arithmetic expansions to  
 122392 support **signed long** integer arithmetic. Implementations are encouraged to support signed  
 122393 integer values at least as large as the size of the largest file allowed on the implementation.

122394 Implementations are also allowed to perform floating-point evaluations as long as an  
 122395 application won't see different results for expressions that would not overflow **signed long**  
 122396 integer expression evaluation. (This includes appropriate truncation of results to integer values.)

122397 Changes made in response to IEEE PASC Interpretation 1003.2 #208 removed the requirement  
 122398 that the integer constant suffixes `l` and `L` had to be recognized. The ISO POSIX-2:1993 standard  
 122399 did not require the `u`, `u1`, `uL`, `U`, `U1`, `UL`, `lu`, `lU`, `Lu`, and `LU` suffixes since only signed integer  
 122400 arithmetic was required. Since all arithmetic expressions were treated as handling **signed long**  
 122401 integer types anyway, the `l` and `L` suffixes were redundant. No known scripts used them and  
 122402 some historic shells did not support them. When the ISO/IEC 9899:1999 standard was used as  
 122403 the basis for the description of arithmetic processing, the `ll` and `LL` suffixes and combinations  
 122404 were also not required. Implementations are still free to accept any or all of these suffixes, but  
 122405 are not required to do so.

122406 There was also some confusion as to whether the shell was required to recognize character  
 122407 constants. Syntactically, character constants were required to be recognized, but the

122408 requirements for the handling of backslash ('\`\`') and quote ('`''`') characters (needed to specify  
 122409 character constants) within an arithmetic expansion were ambiguous. Furthermore, no known  
 122410 shells supported them. Changes made in response to IEEE PASC Interpretation 1003.2 #208  
 122411 removed the requirement to support them (if they were indeed required before). POSIX.1-200x  
 122412 clearly does not require support for character constants.

122413 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/3 is applied, clarifying arithmetic  
 122414 expressions.

#### 122415 C.2.6.5 Field Splitting

122416 The operation of field splitting using *IFS*, as described in early proposals, was based on the way  
 122417 the KornShell splits words, but it is incompatible with other common versions of the shell.  
 122418 However, each has merit, and so a decision was made to allow both. If the *IFS* variable is unset  
 122419 or is `<space><tab><newline>`, the operation is equivalent to the way the System V shell splits  
 122420 words. Using characters outside the `<space><tab><newline>` set yields the KornShell behavior,  
 122421 where each of the non-`<space><tab><newline>`s is significant. This behavior, which affords the  
 122422 most flexibility, was taken from the way the original *awk* handled field splitting.

122423 Rule (3) can be summarized as a pseudo-ERE:

122424  $(s^*ns^* | s^+)$

122425 where *s* is an *IFS* white space character and *n* is a character in the *IFS* that is not white space.  
 122426 Any string matching that ERE delimits a field, except that the *s*<sup>+</sup> form does not delimit fields at  
 122427 the beginning or the end of a line. For example, if *IFS* is `<space>/<comma>/<tab>`, the string:

122428 `<space><space>red<space><space>, <space>white<space>blue`

122429 yields the three colors as the delimited fields.

#### 122430 C.2.6.6 Pathname Expansion

122431 There is no additional rationale provided for this section.

#### 122432 C.2.6.7 Quote Removal

122433 There is no additional rationale provided for this section.

### 122434 C.2.7 Redirection

122435 In the System Interfaces volume of POSIX.1-200x, file descriptors are integers in the range  
 122436 0–(`OPEN_MAX`–1). The file descriptors discussed in XCU Section 2.7 (on page 2259) are that  
 122437 same set of small integers.

122438 Having multi-digit file descriptor numbers for I/O redirection can cause some obscure  
 122439 compatibility problems. Specifically, scripts that depend on an example command:

122440 `echo 22>/dev/null`

122441 echoing "2" to standard error or "22" to standard output are no longer portable. However, the  
 122442 file descriptor number must still be delimited from the preceding text. For example:

122443 `cat file2>foo`

122444 writes the contents of **file2**, not the contents of **file**.

122445 The "`>|`" format of output redirection was adopted from the KornShell. Along with the  
 122446 *noclobber* option, set `-C`, it provides a safety feature to prevent inadvertent overwriting of  
 122447 existing files. (See the RATIONALE for the *pathchk* utility for why this step was taken.) The  
 122448 restriction on regular files is historical practice.

122449 The System V shell and the KornShell have differed historically on pathname expansion of *word*;  
 122450 the former never performed it, the latter only when the result was a single field (file). As a  
 122451 compromise, it was decided that the KornShell functionality was useful, but only as a shorthand  
 122452 device for interactive users. No reasonable shell script would be written with a command such  
 122453 as:

```
122454 cat foo > a*
```

122455 Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with  
 122456 which they are most comfortable.

122457 The construct "2>&1" is often used to redirect standard error to the same file as standard  
 122458 output. Since the redirections take place beginning to end, the order of redirections is significant.  
 122459 For example:

```
122460 ls > foo 2>&1
```

122461 directs both standard output and standard error to file **foo**. However:

```
122462 ls 2>&1 > foo
```

122463 only directs standard output to file **foo** because standard error was duplicated as standard  
 122464 output before standard output was directed to file **foo**.

122465 The "<>" operator could be useful in writing an application that worked with several terminals,  
 122466 and occasionally wanted to start up a shell. That shell would in turn be unable to run  
 122467 applications that run from an ordinary controlling terminal unless it could make use of "<>"  
 122468 redirection. The specific example is a historical version of the pager *more*, which reads from  
 122469 standard error to get its commands, so standard input and standard output are both available  
 122470 for their usual usage. There is no way of saying the following in the shell without "<>":

```
122471 cat food | more - >/dev/tty03 2<>/dev/tty03
```

122472 Another example of "<>" is one that opens **/dev/tty** on file descriptor 3 for reading and writing:

```
122473 exec 3<> /dev/tty
```

122474 An example of creating a lock file for a critical code region:

```
122475 set -C
122476 until 2> /dev/null > lockfile
122477 do sleep 30
122478 done
122479 set +C
122480 perform critical function
122481 rm lockfile
```

122482 Since **/dev/null** is not a regular file, no error is generated by redirecting to it in *noclobber* mode.

122483 Tilde expansion is not performed on a here-document because the data is treated as if it were  
 122484 enclosed in double quotes.

### 122485 C.2.7.1 Redirecting Input

122486 There is no additional rationale provided for this section.

122487 C.2.7.2 *Redirecting Output*

122488 There is no additional rationale provided for this section.

122489 C.2.7.3 *Appending Redirected Output*

122490 Note that when a file is opened (even with the `O_APPEND` flag set), the initial file offset for that  
 122491 file is set to the beginning of the file. Some historic shells set the file offset to the current end-of-  
 122492 file when **append** mode shell redirection was used, but this is not allowed by POSIX.1-200x.

122493 C.2.7.4 *Here-Document*

122494 There is no additional rationale provided for this section.

122495 C.2.7.5 *Duplicating an Input File Descriptor*

122496 There is no additional rationale provided for this section.

122497 C.2.7.6 *Duplicating an Output File Descriptor*

122498 There is no additional rationale provided for this section.

122499 C.2.7.7 *Open File Descriptors for Reading and Writing*

122500 There is no additional rationale provided for this section.

122501 **C.2.8 Exit Status and Errors**122502 C.2.8.1 *Consequences of Shell Errors*

122503 There is no additional rationale provided for this section.

122504 C.2.8.2 *Exit Status for Commands*

122505 There is a historical difference in *sh* and *ksh* non-interactive error behavior. When a command  
 122506 named in a script is not found, some implementations of *sh* exit immediately, but *ksh* continues  
 122507 with the next command. Thus, the Shell and Utilities volume of POSIX.1-200x says that the shell  
 122508 “may” exit in this case. This puts a small burden on the programmer, who has to test for  
 122509 successful completion following a command if it is important that the next command not be  
 122510 executed if the previous command was not found. If it is important for the command to have  
 122511 been found, it was probably also important for it to complete successfully. The test for successful  
 122512 completion would not need to change.

122513 Historically, shells have returned an exit status of  $128+n$ , where  $n$  represents the signal number.  
 122514 Since signal numbers are not standardized, there is no portable way to determine which signal  
 122515 caused the termination. Also, it is possible for a command to exit with a status in the same range  
 122516 of numbers that the shell would use to report that the command was terminated by a signal.  
 122517 Implementations are encouraged to choose exit values greater than 256 to indicate programs that  
 122518 terminate by a signal so that the exit status cannot be confused with an exit status generated by a  
 122519 normal termination.

122520 Historical shells make the distinction between “utility not found” and “utility found but cannot  
 122521 execute” in their error messages. By specifying two seldomly used exit status values for these  
 122522 cases, 127 and 126 respectively, this gives an application the opportunity to make use of this  
 122523 distinction without having to parse an error message that would probably change from locale to  
 122524 locale. The *command*, *env*, *nohup*, and *xargs* utilities in the Shell and Utilities volume of

122525 POSIX.1-200x have also been specified to use this convention.

122526 When a command fails during word expansion or redirection, most historical implementations  
 122527 exit with a status of 1. However, there was some sentiment that this value should probably be  
 122528 much higher so that an application could distinguish this case from the more normal exit status  
 122529 values. Thus, the language “greater than zero” was selected to allow either method to be  
 122530 implemented.

## 122531 C.2.9 Shell Commands

122532 A description of an “empty command” was removed from an early proposal because it is only  
 122533 relevant in the cases of *sh -c " "*, *system(" ")*, or an empty shell-script file (such as the  
 122534 implementation of *true* on some historical systems). Since it is no longer mentioned in the Shell  
 122535 and Utilities volume of POSIX.1-200x, it falls into the silently unspecified category of behavior  
 122536 where implementations can continue to operate as they have historically, but conforming  
 122537 applications do not construct empty commands. (However, note that *sh* does explicitly state an  
 122538 exit status for an empty string or file.) In an interactive session or a script with other commands,  
 122539 extra <newline>s or semicolons, such as:

```
122540 $ false
122541 $
122542 $ echo $?
122543 1
```

122544 would not qualify as the empty command described here because they would be consumed by  
 122545 other parts of the grammar.

### 122546 C.2.9.1 Simple Commands

122547 The enumerated list is used only when the command is actually going to be executed. For  
 122548 example, in:

```
122549 true || $foo *
```

122550 no expansions are performed.

122551 The following example illustrates both how a variable assignment without a command name  
 122552 affects the current execution environment, and how an assignment with a command name only  
 122553 affects the execution environment of the command:

```
122554 $ x=red
122555 $ echo $x
122556 red
122557 $ export x
122558 $ sh -c 'echo $x'
122559 red
122560 $ x=blue sh -c 'echo $x'
122561 blue
122562 $ echo $x
122563 red
```

122564 This next example illustrates that redirections without a command name are still performed:

```
122565 $ ls foo
122566 ls: foo: no such file or directory
122567 $ > foo
122568 $ ls foo
122569 foo
```



122570 A command without a command name, but one that includes a command substitution, has an  
122571 exit status of the last command substitution that the shell performed. For example:

```
122572 if x=$(command)
122573 then ...
122574 fi
```

122575 An example of redirections without a command name being performed in a subshell shows that  
122576 the here-document does not disrupt the standard input of the **while** loop:

```
122577 IFS=:
122578 while read a b
122579 do echo $a
122580 <<-eof
122581 Hello
122582 eof
122583 done </etc/passwd
```

122584 Following are examples of commands without command names in AND-OR lists:

```
122585 > foo || {
122586 echo "error: foo cannot be created" >&2
122587 exit 1
122588 }
```

```
122589 # set saved if /vmunix.save exists
122590 test -f /vmunix.save && saved=1
```

122591 Command substitution and redirections without command names both occur in subshells, but  
122592 they are not necessarily the same ones. For example, in:

```
122593 exec 3> file
122594 var=$(echo foo >&3) 3>&1
```

122595 it is unspecified whether **foo** is echoed to the file or to standard output.

### 122596 Command Search and Execution

122597 This description requires that the shell can execute shell scripts directly, even if the underlying  
122598 system does not support the common "#!" interpreter convention. That is, if file **foo** contains  
122599 shell commands and is executable, the following executes **foo**:

```
122600 ./foo
```

122601 The command search shown here does not match all historical implementations. A more typical  
122602 sequence has been:

- 122603 • Any built-in (special or regular)
- 122604 • Functions
- 122605 • Path search for executable files

122606 But there are problems with this sequence. Since the programmer has no idea in advance which  
122607 utilities might have been built into the shell, a function cannot be used to override portably a  
122608 utility of the same name. (For example, a function named *cd* cannot be written for many  
122609 historical systems.) Furthermore, the *PATH* variable is partially ineffective in this case, and only  
122610 a pathname with a slash can be used to ensure a specific executable file is invoked.

122611 After the *execve()* failure described, the shell normally executes the file as a shell script. Some  
122612 implementations, however, attempt to detect whether the file is actually a script and not an

122613 executable from some other architecture. The method used by the KornShell is allowed by the  
122614 text that indicates non-text files may be bypassed.

122615 The sequence selected for the Shell and Utilities volume of POSIX.1-200x acknowledges that  
122616 special built-ins cannot be overridden, but gives the programmer full control over which  
122617 versions of other utilities are executed. It provides a means of suppressing function lookup (via  
122618 the *command* utility) for the user's own functions and ensures that any regular built-ins or  
122619 functions provided by the implementation are under the control of the path search. The  
122620 mechanisms for associating built-ins or functions with executable files in the path are not  
122621 specified by the Shell and Utilities volume of POSIX.1-200x, but the wording requires that if  
122622 either is implemented, the application is not able to distinguish a function or built-in from an  
122623 executable (other than in terms of performance, presumably). The implementation ensures that  
122624 all effects specified by the Shell and Utilities volume of POSIX.1-200x resulting from the  
122625 invocation of the regular built-in or function (interaction with the environment, variables, traps,  
122626 and so on) are identical to those resulting from the invocation of an executable file.

122627 IEEE Std 1003.1-2001/Cor 2-2004, item XCU/TC2/D6/4 is applied, updating the case where  
122628 *execve()* fails due to an error equivalent to the [ENOEXEC] error.

### 122629 Examples

122630 Consider three versions of the *ls* utility:

- 122631 1. The application includes a shell function named *ls*.
- 122632 2. The user writes a utility named *ls* and puts it in **/fred/bin**.
- 122633 3. The example implementation provides *ls* as a regular shell built-in that is invoked (either  
122634 by the shell or directly by *exec*) when the path search reaches the directory **/posix/bin**.

122635 If *PATH*=**/posix/bin**, various invocations yield different versions of *ls*:

| 122636 | Invocation                                             | Version of <i>ls</i> |
|--------|--------------------------------------------------------|----------------------|
| 122637 | <i>ls</i> (from within application script)             | (1) function         |
| 122638 | <i>command ls</i> (from within application script)     | (3) built-in         |
| 122639 | <i>ls</i> (from within makefile called by application) | (3) built-in         |
| 122640 | <i>system("ls")</i>                                    | (3) built-in         |
| 122641 | <i>PATH="/fred/bin:\$PATH" ls</i>                      | (2) user's version   |

### 122642 C.2.9.2 Pipelines

122643 Because pipeline assignment of standard input or standard output or both takes place before  
122644 redirection, it can be modified by redirection. For example:

```
122645 $ command1 2>&1 | command2
```

122646 sends both the standard output and standard error of *command1* to the standard input of  
122647 *command2*.

122648 The reserved word **!** allows more flexible testing using AND and OR lists.

122649 It was suggested that it would be better to return a non-zero value if any command in the  
122650 pipeline terminates with non-zero status (perhaps the bitwise-inclusive OR of all return values).  
122651 However, the choice of the last-specified command semantics are historical practice and would  
122652 cause applications to break if changed. An example of historical behavior:

```
122653 $ sleep 5 | (exit 4)
122654 $ echo $?
122655 4
```

```

122656 $ (exit 4) | sleep 5
122657 $ echo $?
122658 0

```

### 122659 C.2.9.3 Lists

122660 The equal precedence of "&&" and "||" is historical practice. The standard developers  
 122661 evaluated the model used more frequently in high-level programming languages, such as C, to  
 122662 allow the shell logical operators to be used for complex expressions in an unambiguous way, but  
 122663 they could not allow historical scripts to break in the subtle way unequal precedence might  
 122664 cause. Some arguments were posed concerning the "{" or "(" groupings that are required  
 122665 historically. There are some disadvantages to these groupings:

- 122666 • The "(" can be expensive, as they spawn other processes on some implementations. This  
 122667 performance concern is primarily an implementation issue.
- 122668 • The "{" braces are not operators (they are reserved words) and require a trailing space  
 122669 after each '{', and a semicolon before each}'. Most programmers (and certainly  
 122670 interactive users) have avoided braces as grouping constructs because of the problematic  
 122671 syntax required. Braces were not changed to operators because that would generate  
 122672 compatibility issues even greater than the precedence question; braces appear outside the  
 122673 context of a keyword in many shell scripts.

122674 IEEE PASC Interpretation 1003.2 #204 is applied, clarifying that the operators "&&" and "||"  
 122675 are evaluated with left associativity.

#### 122676 Asynchronous Lists

122677 The grammar treats a construct such as:

```
122678 foo & bar & bam &
```

122679 as one "asynchronous list", but since the status of each element is tracked by the shell, the term  
 122680 "element of an asynchronous list" was introduced to identify just one of the **foo**, **bar**, or **bam**  
 122681 portions of the overall list.

122682 Unless the implementation has an internal limit, such as {CHILD\_MAX}, on the retained process  
 122683 IDs, it would require unbounded memory for the following example:

```

122684 while true
122685 do foo & echo $!
122686 done

```

122687 The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in XCU |  
 122688 [Section 2.11](#) (on page 2277).

122689 Since the connection of the input to the equivalent of /dev/null is considered to occur before  
 122690 redirections, the following script would produce no output:

```

122691 exec < /etc/passwd
122692 cat <&0 &
122693 wait

```

122694 **Sequential Lists**

122695 There is no additional rationale provided for this section.

122696 **AND Lists**

122697 There is no additional rationale provided for this section.

122698 **OR Lists**

122699 There is no additional rationale provided for this section.

122700 C.2.9.4 *Compound Commands*122701 **Grouping Commands**

122702 The semicolon shown in `{compound-list};` is an example of a control operator delimiting the } |  
 122703 reserved word. Other delimiters are possible, as shown in XCU [Section 2.10](#) (on page 2271);  
 122704 `<newline>` is frequently used.

122705 A proposal was made to use the **<do-done>** construct in all cases where command grouping in  
 122706 the current process environment is performed, identifying it as a construct for the grouping  
 122707 commands, as well as for shell functions. This was not included because the shell already has a  
 122708 grouping construct for this purpose ("`{ }`"), and changing it would have been counter-  
 122709 productive.

122710 **For Loop**

122711 The format is shown with generous usage of `<newline>`s. See the grammar in XCU [Section 2.10](#) |  
 122712 (on page 2271) for a precise description of where `<newline>`s and semicolons can be  
 122713 interchanged.

122714 Some historical implementations support '`{`' and '`}`' as substitutes for **do** and **done**. The  
 122715 standard developers chose to omit them, even as an obsolescent feature. (Note that these  
 122716 substitutes were only for the **for** command; the **while** and **until** commands could not use them  
 122717 historically because they are followed by compound-lists that may contain "`{ . . . }`" grouping  
 122718 commands themselves.)

122719 The reserved word pair **do ... done** was selected rather than **do ... od** (which would have  
 122720 matched the spirit of **if ... fi** and **case ... esac**) because *od* is already the name of a standard  
 122721 utility.

122722 PASC Interpretation 1003.2 #169 has been applied changing the grammar.

122723 **Case Conditional Construct**

122724 An optional left parenthesis before *pattern* was added to allow numerous historical KornShell  
 122725 scripts to conform. At one time, using the leading parenthesis was required if the **case** statement  
 122726 was to be embedded within a "`$( )`" command substitution; this is no longer the case with the  
 122727 POSIX shell. Nevertheless, many historical scripts use the left parenthesis, if only because it  
 122728 makes matching-parenthesis searching easier in *vi* and other editors. This is a relatively simple  
 122729 implementation change that is upwards-compatible for all scripts.

122730 Consideration was given to requiring *break* inside the *compound-list* to prevent falling through to  
 122731 the next pattern action list. This was rejected as being nonexistent practice. An interesting  
 122732 undocumented feature of the KornShell is that using "`;&`" instead of "`;;`" as a terminator  
 122733 causes the exact opposite behavior—the flow of control continues with the next *compound-list*.

122734 The pattern '`*'`', given as the last pattern in a **case** construct, is equivalent to the default case in

122735 a C-language **switch** statement.

122736 The grammar shows that reserved words can be used as patterns, even if one is the first word on  
122737 a line. Obviously, the reserved word **esac** cannot be used in this manner.

### 122738 **If Conditional Construct**

122739 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2271). |

### 122740 **While Loop**

122741 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2271). |

### 122742 **Until Loop**

122743 The precise format for the command syntax is described in XCU [Section 2.10](#) (on page 2271). |

### 122744 C.2.9.5 *Function Definition Command*

122745 The description of functions in an early proposal was based on the notion that functions should  
122746 behave like miniature shell scripts; that is, except for sharing variables, most elements of an  
122747 execution environment should behave as if they were a new execution environment, and  
122748 changes to these should be local to the function. For example, traps and options should be reset  
122749 on entry to the function, and any changes to them do not affect the traps or options of the caller.  
122750 There were numerous objections to this basic idea, and the opponents asserted that functions  
122751 were intended to be a convenient mechanism for grouping common commands that were to be  
122752 executed in the current execution environment, similar to the execution of the *dot* special  
122753 built-in.

122754 It was also pointed out that the functions described in that early proposal did not provide a local  
122755 scope for everything a new shell script would, such as the current working directory, or *umask*,  
122756 but instead provided a local scope for only a few select properties. The basic argument was that  
122757 if a local scope is needed for the execution environment, the mechanism already existed: the  
122758 application can put the commands in a new shell script and call that script. All historical shells  
122759 that implemented functions, other than the KornShell, have implemented functions that operate  
122760 in the current execution environment. Because of this, traps and options have a global scope  
122761 within a shell script. Local variables within a function were considered and included in another  
122762 early proposal (controlled by the special built-in *local*), but were removed because they do not fit  
122763 the simple model developed for functions and because there was some opposition to adding yet  
122764 another new special built-in that was not part of historical practice. Implementations should  
122765 reserve the identifier *local* (as well as *typeset*, as used in the KornShell) in case this local variable  
122766 mechanism is adopted in a future version of this standard. |

122767 A separate issue from the execution environment of a function is the availability of that function  
122768 to child shells. A few objectors maintained that just as a variable can be shared with child shells  
122769 by exporting it, so should a function. In early proposals, the *export* command therefore had a *-f*  
122770 flag for exporting functions. Functions that were exported were to be put into the environment  
122771 as *name(=value* pairs, and upon invocation, the shell would scan the environment for these and  
122772 automatically define these functions. This facility was strongly opposed and was omitted. Some  
122773 of the arguments against exportable functions were as follows:

- 122774 • There was little historical practice. The Ninth Edition shell provided them, but there was  
122775 controversy over how well it worked.
- 122776 • There are numerous security problems associated with functions appearing in the  
122777 environment of a user and overriding standard utilities or the utilities owned by the  
122778 application.

- 122779
- 122780
- There was controversy over requiring *make* to import functions, where it has historically used an *exec* function for many of its command line executions.
- 122781
- Functions can be big and the environment is of a limited size. (The counter-argument was that functions are no different from variables in terms of size: there can be big ones, and there can be small ones—and just as one does not export huge variables, one does not export huge functions. However, this might not apply to the average shell-function writer, who typically writes much larger functions than variables.)
- 122782
- 122783
- 122784
- 122785

122786 As far as can be determined, the functions in the Shell and Utilities volume of POSIX.1-200x  
122787 match those in System V. Earlier versions of the KornShell had two methods of defining  
122788 functions:

122789 `function fname { compound-list }`

122790 and:

122791 `fname() { compound-list }`

122792 The latter used the same definition as the Shell and Utilities volume of POSIX.1-200x, but  
122793 differed in semantics, as described previously. The current edition of the KornShell aligns the  
122794 latter syntax with the Shell and Utilities volume of POSIX.1-200x and keeps the former as is.

122795 The name space for functions is limited to that of a *name* because of historical practice.  
122796 Complications in defining the syntactic rules for the function definition command and in  
122797 dealing with known extensions such as the "@()" usage in the KornShell prevented the name  
122798 space from being widened to a *word*. Using functions to support synonyms such as the "!!"  
122799 and '%' usage in the C shell is thus disallowed to conforming applications, but acceptable as an  
122800 extension. For interactive users, the aliasing facilities in the Shell and Utilities volume of  
122801 POSIX.1-200x should be adequate for this purpose. It is recognized that the name space for  
122802 utilities in the file system is wider than that currently supported for functions, if the portable  
122803 filename character set guidelines are ignored, but it did not seem useful to mandate extensions  
122804 in systems for so little benefit to conforming applications.

122805 The "()" in the function definition command consists of two operators. Therefore, intermixing  
122806 <blank>s with the *fname*, '( ', and ' )' is allowed, but unnecessary.

122807 An example of how a function definition can be used wherever a simple command is allowed:

```
122808 # If variable i is equal to "yes",
122809 # define function foo to be ls -l
122810 #
122811 ["$i" = yes] && foo() {
122812 ls -l
122813 }
```

## 122814 C.2.10 Shell Grammar

122815 There are several subtle aspects of this grammar where conventional usage implies rules about  
122816 the grammar that in fact are not true.

122817 For *compound\_list*, only the forms that end in a *separator* allow a reserved word to be recognized,  
122818 so usually only a *separator* can be used where a compound list precedes a reserved word (such as  
122819 **Then**, **Else**, **Do**, and **Rbrace**). Explicitly requiring a separator would disallow such valid (if rare)  
122820 statements as:

```
122821 if (false) then (echo x) else (echo y) fi
```

122822 See the Note under special grammar rule (1).

122823 Concerning the third sentence of rule (1) (“Also, if the parser ...”):

122824 • This sentence applies rather narrowly: when a compound list is terminated by some clear  
122825 delimiter (such as the closing **fi** of an inner **if\_clause**) then it would apply; where the  
122826 compound list might continue (as in after a ‘;’), rule (7a) (and consequently the first  
122827 sentence of rule (1)) would apply. In many instances the two conditions are identical, but  
122828 this part of rule (1) does not give license to treating a **WORD** as a reserved word unless it  
122829 is in a place where a reserved word has to appear.

122830 • The statement is equivalent to requiring that when the LR(1) lookahead set contains  
122831 exactly one reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the  
122832 theoretical concepts, not to any real parser generator.)

122833 For example, in the construct below, and when the parser is at the point marked with ‘^’,  
122834 the only next legal token is **then** (this follows directly from the grammar rules):

```
122835 if if...fi then ... fi
122836 ^
```

122837 At that point, the **then** must be recognized as a reserved word.

122838 (Depending on the parser generator actually used, “extra” reserved words may be in some  
122839 lookahead sets. It does not really matter if they are recognized, or even if any possible  
122840 reserved word is recognized in that state, because if it is recognized and is not in the  
122841 (theoretical) LR(1) lookahead set, an error is ultimately detected. In the example above, if  
122842 some other reserved word (for example, **while**) is also recognized, an error occurs later.

122843 This is approximately equivalent to saying that reserved words are recognized after other  
122844 reserved words (because it is after a reserved word that this condition occurs), but avoids  
122845 the “except for ...” list that would be required for **case**, **for**, and so on. (Reserved words  
122846 are of course recognized anywhere a *simple\_command* can appear, as well. Other rules take  
122847 care of the special cases of non-recognition, such as rule (4) for **case** statements.)

122848 Note that the body of here-documents are handled by token recognition (see XCU [Section 2.3](#), on  
122849 page 2247) and do not appear in the grammar directly. (However, the here-document I/O  
122850 redirection operator is handled as part of the grammar.)

122851 The start symbol of the grammar (**complete\_command**) represents either input from the  
122852 command line or a shell script. It is repeatedly applied by the interpreter to its input and  
122853 represents a single “chunk” of that input as seen by the interpreter.

#### 122854 C.2.10.1 Shell Grammar Lexical Conventions

122855 There is no additional rationale provided for this section.

#### 122856 C.2.10.2 Shell Grammar Rules

122857 There is no additional rationale provided for this section.

## 122858 C.2.11 Signals and Error Handling

122859 SD5-XCU-ERN-93 is applied, updating the first paragraph of XCU [Section 2.11](#) (on page 2277).

## 122860 C.2.12 Shell Execution Environment

122861 Some implementations have implemented the last stage of a pipeline in the current environment  
122862 so that commands such as:

```
122863 command | read foo
```

122864 set variable **foo** in the current environment. This extension is allowed, but not required;  
122865 therefore, a shell programmer should consider a pipeline to be in a subshell environment, but  
122866 not depend on it.

122867 In early proposals, the description of execution environment failed to mention that each  
122868 command in a multiple command pipeline could be in a subshell execution environment. For  
122869 compatibility with some historical shells, the wording was phrased to allow an implementation  
122870 to place any or all commands of a pipeline in the current environment. However, this means that  
122871 a POSIX application must assume each command is in a subshell environment, but not depend  
122872 on it.

122873 The wording about shell scripts is meant to convey the fact that describing “trap actions” can  
122874 only be understood in the context of the shell command language. Outside of this context, such  
122875 as in a C-language program, signals are the operative condition, not traps.

## 122876 C.2.13 Pattern Matching Notation

122877 Pattern matching is a simpler concept and has a simpler syntax than REs, as the former is  
122878 generally used for the manipulation of filenames, which are relatively simple collections of  
122879 characters, while the latter is generally used to manipulate arbitrary text strings of potentially  
122880 greater complexity. However, some of the basic concepts are the same, so this section points  
122881 liberally to the detailed descriptions in XBD [Chapter 9](#) (on page 167).

### 122882 C.2.13.1 Patterns Matching a Single Character

122883 Both quoting and escaping are described here because pattern matching must work in three  
122884 separate circumstances:

- 122885 1. Calling directly upon the shell, such as in pathname expansion or in a **case** statement. All  
122886 of the following match the string or file **abc**:

```
122887 abc "abc" a"b" c a\b c a[b] c a["b"] c a[\b] c a["\b"] c a?c a*c
```

122888 The following do not:

```
122889 "a?c" a*c a\[b]c
```

- 122890 2. Calling a utility or function without going through a shell, as described for *find* and the  
122891 *fnmatch()* function defined in the System Interfaces volume of POSIX.1-200x.
- 122892 3. Calling utilities such as *find*, *cpio*, *tar*, or *pax* through the shell command line. In this case,  
122893 shell quote removal is performed before the utility sees the argument. For example, in:

```
122894 find /bin -name "e\c[\h]o" -print
```

122895 after quote removal, the backslashes are presented to *find* and it treats them as escape  
122896 characters. Both precede ordinary characters, so the *c* and *h* represent themselves and *echo*  
122897 would be found on many historical systems (that have it in **/bin**). To find a filename that  
122898 contained shell special characters or pattern characters, both quoting and escaping are



122899 required, such as:

122900 `paX -r . . . "*"a\(\?"`

122901 to extract a filename ending with "a(?".

122902 Conforming applications are required to quote or escape the shell special characters (sometimes  
122903 called metacharacters). If used without this protection, syntax errors can result or  
122904 implementation extensions can be triggered. For example, the KornShell supports a series of  
122905 extensions based on parentheses in patterns.

122906 The restriction on a circumflex in a bracket expression is to allow implementations that support  
122907 pattern matching using the circumflex as the negation character in addition to the exclamation  
122908 mark. A conforming application must use something like "[\^!]" to match either character.

### 122909 C.2.13.2 Patterns Matching Multiple Characters

122910 Since each asterisk matches zero or more occurrences, the patterns "a\*b" and "a\*\*b" have  
122911 identical functionality.

#### 122912 Examples

122913 `a[bc]` Matches the strings "ab" and "ac".

122914 `a*d` Matches the strings "ad", "abd", and "abcd", but not the string "abc".

122915 `a*d*` Matches the strings "ad", "abcd", "abcdef", "aaaad", and "adddd".

122916 `*a*d` Matches the strings "ad", "abcd", "efabcd", "aaaad", and "adddd".

### 122917 C.2.13.3 Patterns Used for Filename Expansion

122918 The caveat about a slash within a bracket expression is derived from historical practice. The  
122919 pattern "a[b/c]d" does not match such pathnames as **abd** or **a/d**. On some implementations  
122920 (including those conforming to the Single UNIX Specification), it matched a pathname of  
122921 literally "a[b/c]d". On other systems, it produced an undefined condition (an unescaped '['  
122922 used outside a bracket expression). In this version, the XSI behavior is now required.

122923 Filenames beginning with a period historically have been specially protected from view on  
122924 UNIX systems. A proposal to allow an explicit period in a bracket expression to match a leading  
122925 period was considered; it is allowed as an implementation extension, but a conforming  
122926 application cannot make use of it. If this extension becomes popular in the future, it will be  
122927 considered for a future version of the Shell and Utilities volume of POSIX.1-200x.

122928 Historical systems have varied in their permissions requirements. To match **f\*/bar** has required  
122929 read permissions on the **f\*** directories in the System V shell, but the Shell and Utilities volume of  
122930 POSIX.1-200x, the C shell, and KornShell require only search permissions.

## 122931 C.2.14 Special Built-In Utilities

122932 See the RATIONALE sections on the individual reference pages.

## 122933 C.3 Batch Environment Services and Utilities

### 122934 Scope of the Batch Environment Services and Utilities Option

122935 This section summarizes the deliberations of the IEEE P1003.15 (Batch Environment) working  
 122936 group in the development of the Batch Environment Services and Utilities option, which covers  
 122937 a set of services and utilities defining a batch processing system.

122938 This informative section contains historical information concerning the contents of the  
 122939 amendment and describes why features were included or discarded by the working group.

### 122940 History of Batch Systems

122941 The supercomputing technical committee began as a “Birds Of a Feather” (BOF) at the January  
 122942 1987 Usenix meeting. There was enough general interest to form a supercomputing attachment  
 122943 to the /usr/group working groups. Several subgroups rapidly formed. Of those subgroups, the  
 122944 batch group was the most ambitious. The first early meetings were spent evaluating user needs  
 122945 and existing batch implementations.

122946 To evaluate user needs, individuals from the supercomputing community came and presented  
 122947 their needs. Common requests were flexibility, interoperability, control of resources, and ease-of-  
 122948 use. Backward-compatibility was not an issue. The working group then evaluated some existing  
 122949 systems. The following different systems were evaluated:

- 122950 • PROD
- 122951 • Convex Distributed Batch
- 122952 • NQS
- 122953 • CTSS
- 122954 • MDQS from Ballistics Research Laboratory (BRL)

122955 Finally, NQS was chosen as a model because it satisfied not only the most user requirements, but  
 122956 because it was public domain, already implemented on a variety of hardware platforms, and  
 122957 network-based.

### 122958 Historical Implementations of Batch Systems

122959 Deferred processing of work under the control of a scheduler has been a feature of most  
 122960 proprietary operating systems from the earliest days of multi-user systems in order to maximize  
 122961 utilization of the computer.

122962 The arrival of UNIX systems proved to be a dilemma to many hardware providers and users  
 122963 because it did not include the sophisticated batch facilities offered by the proprietary systems.  
 122964 This omission was rectified in 1986 by NASA Ames Research Center who developed the  
 122965 Network Queuing System (NQS) as a portable UNIX application that allowed the routing and  
 122966 processing of batch “jobs” in a network. To encourage its usage, the product was later put into  
 122967 the public domain. It was promptly picked up by UNIX hardware providers, and ported and  
 122968 developed for their respective hardware and UNIX implementations.

122969 Many major vendors, who traditionally offer a batch-dominated environment, ported the  
 122970 public-domain product to their systems, customized it to support the capabilities of their  
 122971 systems, and added many customer-requested features.

122972 Due to the strong hardware provider and customer acceptance of NQS, it was decided to use  
 122973 NQS as the basis for the POSIX Batch Environment amendment in 1987. Other batch systems  
 122974 considered at the time included CTSS, MDQS (a forerunner of NQS from the Ballistics Research  
 122975 Laboratory), and PROD (a Los Alamos Labs development). None were thought to have both the

122976 functionality and acceptability of NQS.

122977 **NQS Differences from the *at* utility**

122978 The base standard *at* and *batch* utilities are not sufficient to meet the batch processing needs in a  
122979 supercomputing environment and additional functionality in the areas of resource management,  
122980 job scheduling, system management, and control of output is required.

122981 **Batch Environment Services and Utilities Option Definitions**

122982 The concept of a batch job is closely related to a session with a session leader. The main  
122983 difference is that a batch job does not have a controlling terminal. There has been much debate  
122984 over whether to use the term “request” or “job”. Job was the final choice because of the  
122985 historical use of this term in the batch environment.

122986 The current definition for job identifiers is not sufficient with the model of destinations. The  
122987 current definition is:

122988 `sequence_number.originating_host`

122989 Using the model of destination, a host may include multiple batch nodes, the location of which  
122990 is identified uniquely by a name or directory service. If the current definition is used, batch  
122991 nodes running on the same host would have to coordinate their use of sequence numbers, as  
122992 sequence numbers are assigned by the originating host. The alternative is to use the originating  
122993 batch node name instead of the originating host name.

122994 The reasons for wishing to run more than one batch system per host could be the following.

122995 A test and production batch system are maintained on a single host. This is most likely in a  
122996 development facility, but could also arise when a site is moving from one version to another. The  
122997 new batch system could be installed as a test version that is completely separate from the  
122998 production batch system, so that problems can be isolated to the test system. Requiring the batch  
122999 nodes to coordinate their use of sequence numbers creates a dependency between the two  
123000 nodes, and that defeats the purpose of running two nodes.

123001 A site has multiple departments using a single host, with different management policies. An  
123002 example of contention might be in job selection algorithms. One group might want a FIFO type  
123003 of selection, while another group wishes to use a more complex algorithm based on resource  
123004 availability. Again, requiring the batch nodes to coordinate is an unnecessary binding.

123005 The proposal eventually accepted was to replace originating host with originating batch node.  
123006 This supplies sufficient granularity to ensure unique job identifiers. If more than one batch node  
123007 is on a particular host, they each have their own unique name.

123008 The queue portion of a destination is not part of the job identifier as these are not required to be  
123009 unique between batch nodes. For instance, two batch nodes may both have queues called small,  
123010 medium, and large. It is only the batch node name that is uniquely identifiable throughout the  
123011 batch system. The queue name has no additional function in this context.

123012 Assume there are three batch nodes, each of which has its own name server. On batch node one,  
123013 there are no queues. On batch node two, there are fifty queues. On batch node three, there are  
123014 forty queues. The system administrator for batch node one does not have to configure queues,  
123015 because there are none implemented. However, if a user wishes to send a job to either batch  
123016 node two or three, the system administrator for batch node one must configure a destination  
123017 that maps to the appropriate batch node and queue. If every queue is to be made accessible from  
123018 batch node one, the system administrator has to configure ninety destinations.

123019 To avoid requiring this, there should be a mechanism to allow a user to separate the destination  
123020 into a batch node name and a queue name. Then, an implementation that is configured to get to

123021 all the batch nodes does not need any more configuration to allow a user to get to all of the  
 123022 queues on all of the batch nodes. The node name is used to locate the batch node, while the  
 123023 queue name is sent unchanged to that batch node.

123024 The following are requirements that a destination identifier must be capable of providing:

- 123025 • The ability to direct a job to a queue in a particular batch node.
- 123026 • The ability to direct a job to a particular batch node.
- 123027 • The ability to group at a higher level than just one queue. This includes grouping similar  
 123028 queues across multiple batch nodes (this is a pipe queue).
- 123029 • The ability to group batch nodes. This allows a user to submit a job to a group name with  
 123030 no knowledge of the batch node configuration. This also provides aliasing as a special  
 123031 case. Aliasing is a group containing only one batch node name. The group name is the  
 123032 alias.

123033 In addition, the administrator has the following requirements:

- 123034 • The ability to control access to the queues.
- 123035 • The ability to control access to the batch nodes.
- 123036 • The ability to control access to groups of queues (pipe queues).
- 123037 • The ability to configure retry time intervals and durations.

123038 The requirements of the user are met by destination as explained in the following.

123039 The user has the ability to specify a queue name, which is known only to the batch node  
 123040 specified. There is no configuration of these queues required on the submitting node.

123041 The user has the ability to specify a batch node whose name is network-unique. The  
 123042 configuration required is that the batch node be defined as an application, just as other  
 123043 applications such as FTP are configured.

123044 Once a job reaches a queue, it can again become a user of the batch system. The batch node can  
 123045 choose to send the job to another batch node or queue or both. In other words, the routing is at  
 123046 an application level, and it is up to the batch system to choose where the job will be sent.  
 123047 Configuration is up to the batch node where the queue resides. This provides grouping of  
 123048 queues across batch nodes or within a batch node. The user submits the job to a queue, which by  
 123049 definition routes the job to other queues or nodes or both.

123050 A node name may be given to a naming service, which returns multiple addresses as opposed to  
 123051 just one. This provides grouping at a batch node level. This is a local issue, meaning that the  
 123052 batch node must choose only one of these addresses. The list of addresses is not sent with the  
 123053 job, and once the job is accepted on another node, there is no connection between the list and the  
 123054 job. The requirements of the administrator are met by destination as explained in the following.

123055 The control of queues is a batch system issue, and will be done using the batch administrative  
 123056 utilities.

123057 The control of nodes is a network issue, and will be done through whatever network facilities  
 123058 are available.

123059 The control of access to groups of queues (pipe queues) is covered by the control of any other  
 123060 queue. The fact that the job may then be sent to another destination is not relevant.

123061 The propagation of a job across more than one point-to-point connection was dropped because  
 123062 of its complexity and because all of the issues arising from this capability could not be resolved.  
 123063 It could be provided as additional functionality at some time in the future.

123064 The addition of *network* as a defined term was done to clarify the difference between a network  
 123065 of batch nodes as opposed to a network of hosts. A network of batch nodes is referred to as a  
 123066 batch system. The network refers to the actual host configuration. A single host may have  
 123067 multiple batch nodes.

123068 In the absence of a standard network naming convention, this option establishes its own  
 123069 convention for the sake of consistency and expediency. This is subject to change, should a future  
 123070 working group develop a standard naming convention for network pathnames.

### 123071 C.3.1 Batch General Concepts

123072 During the development of the Batch Environment Services and Utilities option, a number of  
 123073 topics were discussed at length which influenced the wording of the normative text but could  
 123074 not be included in the final text. The following items are some of the most significant terms and  
 123075 concepts of those discussed:

- 123076 • Small and Consistent Command Set

123077 Often, conventional utilities from UNIX systems have a very complicated utility syntax  
 123078 and usage. This can often result in confusion and errors when trying to use them. The  
 123079 Batch Environment Services and Utilities option utility set, on the other hand, has been  
 123080 paired to a small set of robust utilities with an orthogonal calling sequence.

- 123081 • Checkpoint/Restart

123082 This feature permits an already executing process to checkpoint or save its contents. Some  
 123083 implementations permit this at both the batch utility level (for example, checkpointing this  
 123084 job upon its abnormal termination) or from within the job itself via a system call. Support  
 123085 of checkpoint/restart is optional. A conscious, careful effort was made to make the *qsub*  
 123086 utility consistently refer to checkpoint/restart as optional functionality.

- 123087 • Rerunability

123088 When a user submits a job for batch processing, they can designate it “rerunnable” in that  
 123089 it will automatically resume execution from the start of the job if the machine on which it  
 123090 was executing crashes for some reason. The decision on whether the job will be rerun or  
 123091 not is entirely up to the submitter of the job and no decisions will be made within the batch  
 123092 system. A job that is rerunnable and has been submitted with the proper  
 123093 checkpoint/restart switch will first be checkpointed and execution begun from that point.  
 123094 Furthermore, use of the implementation-defined checkpoint/restart feature will not be  
 123095 defined in this context.

- 123096 • Error Codes

123097 All utilities exit with error status zero (0) if successful, one (1) if a user error occurred, and  
 123098 two (2) for an internal Batch Environment Services and Utilities option error.

- 123099 • Level of Portability

123100 Portability is specified at both the user, operator, and administrator levels. A conforming  
 123101 batch implementation prevents identical functionality and behavior at all these levels.  
 123102 Additionally, portable batch shell scripts with embedded Batch Environment Services and  
 123103 Utilities option utilities add an additional level of portability.

- 123104 • Resource Specification

123105 A small set of globally understood resources, such as memory and CPU time, is specified.  
 123106 All conforming batch implementations are able to process them in a manner consistent  
 123107 with the yet-to-be-developed resource management model. Resources not in this  
 123108 amendment set are ignored and passed along as part of the argument stream of the utility.

- 123109
- Queue Position
- 123110 Queue position is the place a job occupies in a queue. It is dependent on a variety of factors  
123111 such as submission time and priority. Since priority may be affected by the implementation  
123112 of fair share scheduling, the definition of queue position is implementation-defined.
- 123113
- Queue ID
- 123114 A numerical queue ID is an external requirement for purposes of accounting. The  
123115 identification number was chosen over queue name for processing convenience.
- 123116
- Job ID
- 123117 A common notion of “jobs” is a collection of processes whose process group cannot be  
123118 altered and is used for resource management and accounting. This concept is  
123119 implementation-defined and, as such, has been omitted from the batch amendment.
- 123120
- Bytes *versus* Words
- 123121 Except for one case, bytes are used as the standard unit for memory size. Furthermore, the  
123122 definition of a word varies from machine to machine. Therefore, bytes will be the default  
123123 unit of memory size.
- 123124
- Regular Expressions
- 123125 The standard definition of regular expressions is much too broad to be used in the batch  
123126 utility syntax. All that is needed is a simple concept of “all”; for example, delete all my jobs  
123127 from the named queue. For this reason, regular expressions have been eliminated from the  
123128 batch amendment.
- 123129
- Display Privacy
- 123130 How much data should be displayed locally through functions? Local policy dictates the  
123131 amount of privacy. Library functions must be used to create and enforce local policy.  
123132 Network and local *qstats* must reflect the policy of the server machine.
- 123133
- Remote Host Naming Convention
- 123134 It was decided that host names would be a maximum of 255 characters in length, with at  
123135 most 15 characters being shown in displays. The 255 character limit was chosen because it  
123136 is consistent with BSD. The 15-character limit was an arbitrary decision.
- 123137
- Network Administration
- 123138 Network administration is important, but is outside the scope of the batch amendment.  
123139 Network administration could be done with *rsh*. However, authentication becomes two-  
123140 sided.
- 123141
- Network Administration Philosophy
- 123142 Keep it simple. Centralized management should be possible. For example, Los Alamos  
123143 needs a dumb set of CPUs to be managed by a central system *versus* several  
123144 independently-managed systems as is the general case for the Batch Environment Services  
123145 and Utilities option.
- 123146
- Operator Utility Defaults (that is, Default Host, User, Account, and so on)
- 123147 It was decided that usability would override orthogonality and syntactic consistency.
- 123148
- The Batch System Manager and Operator Distinction
- 123149 The distinction between manager and operator is that operators can only control the flow  
123150 of jobs. A manager can alter the batch system configuration in addition to job flow. POSIX

123151 makes a distinction between user and system administrator but goes no further. The  
 123152 concepts of manager and operator privileges fall under local policy. The distinction  
 123153 between manager and operator is historical in batch environments, and the Batch  
 123154 Environment Services and Utilities option has continued that distinction.

123155 • The Batch System Administrator

123156 An administrator is equivalent to a batch system manager.

### 123157 C.3.2 Batch Services

123158 This rationale is provided as informative rather than normative text, to avoid placing  
 123159 requirements on implementors regarding the use of symbolic constants, but at the same time to  
 123160 give implementors a preferred practice for assigning values to these constants to promote  
 123161 interoperability.

123162 The *Checkpoint* and *Minimum\_Cpu\_Interval* attributes induce a variety of behavior depending  
 123163 upon their values. Some jobs cannot or should not be checkpointed. Other users will simply  
 123164 need to ensure job continuation across planned downtimes; for example, scheduled preventive  
 123165 maintenance. For users consuming expensive resources, or for jobs that run longer than the  
 123166 mean time between failures, however, periodic checkpointing may be essential. However,  
 123167 system administrators must be able to set minimum checkpoint intervals on a queue-by-queue  
 123168 basis to guard against, for example, naive users specifying interval values too small on memory-  
 123169 intensive jobs. Otherwise, system overhead would adversely affect performance.

123170 The use of symbolic constants, such as `NO_CHECKPOINT`, was introduced to lend a degree of  
 123171 formalism and portability to this option.

123172 Support for checkpointing is optional for servers. However, clients must provide for the `-c`  
 123173 option, since in a distributed environment the job may run on a server that does provide such  
 123174 support, even if the host of the client does not support the checkpoint feature.

123175 If the user does not specify the `-c` option, the default action is left unspecified by this option.  
 123176 Some implementations may wish to do checkpointing by default; others may wish to checkpoint  
 123177 only under an explicit request from the user.

123178 The *Priority* attribute has been made non-optional. All clients already had been required to  
 123179 support the `-p` option. The concept of prioritization is common in historical implementations.  
 123180 The default priority is left to the server to establish.

123181 The *Hold\_Types* attribute has been modified to allow for implementation-defined hold types to  
 123182 be passed to a batch server.

123183 It was the intent of the IEEE P1003.15 working group to mandate the support for the  
 123184 *Resource\_List* attribute in this option by referring to another amendment, specifically the  
 123185 IEEE P1003.1a draft standard. However, during the development of the IEEE P1003.1a draft  
 123186 standard this was excluded. As such this requirement has been removed from the normative  
 123187 text.

123188 The *Shell\_Path* attribute has been modified to accept a list of shell paths that are associated with  
 123189 a host. The name of the attribute has been changed to *Shell\_Path\_List*.

### 123190 C.3.3 Common Behavior for Batch Environment Utilities

123191 This section was defined to meet the goal of a “Small and Consistent Command Set” for this  
123192 option.

## 123193 C.4 Utilities

123194 For the utilities included in POSIX.1-200x, see the RATIONALE sections on the individual  
123195 reference pages.

### 123196 Exclusion of Utilities

123197 The set of utilities contained in POSIX.1-200x is drawn from the base documents, with one  
123198 addition: the *c99* utility. This section contains rationale for some of the deliberations that led to  
123199 this set of utilities, and why certain utilities were excluded.

123200 Many utilities were evaluated by the standard developers; more historical utilities were  
123201 excluded from the base documents than included. The following list contains many common  
123202 UNIX system utilities that were not included as mandatory utilities, in the User Portability  
123203 Utilities option, in the XSI option, or in one of the software development groups. It is logistically  
123204 difficult for this rationale to distribute correctly the reasons for not including a utility among the  
123205 various utility options. Therefore, this section covers the reasons for all utilities not included in  
123206 POSIX.1-200x.

123207 This rationale is limited to a discussion of only those utilities actively or indirectly evaluated by  
123208 the standard developers of the base documents, rather than the list of all known UNIX utilities  
123209 from all its variants.

123210 *adb* The intent of the various software development utilities was to assist in the  
123211 installation (rather than the actual development and debugging) of applications.  
123212 This utility is primarily a debugging tool. Furthermore, many useful aspects of *adb*  
123213 are very hardware-specific.

123214 *as* Assemblers are hardware-specific and are included implicitly as part of the  
123215 compilers in POSIX.1-200x.

123216 *banner* The only known use of this command is as part of the *lp* printer header pages. It  
123217 was decided that the format of the header is implementation-defined, so this utility  
123218 is superfluous to application portability.

123219 *calendar* This reminder service program is not useful to conforming applications.

123220 *cancel* The *lp* (line printer spooling) system specified is the most basic possible and did  
123221 not need this level of application control.

123222 *chroot* This is primarily of administrative use, requiring superuser privileges.

123223 *col* No utilities defined in POSIX.1-200x produce output requiring such a filter. The  
123224 *nroff* text formatter is present on many historical systems and will continue to  
123225 remain as an extension; *col* is expected to be shipped by all the systems that ship  
123226 *nroff*.

123227 *cpio* This has been replaced by *pax*, for reasons explained in the rationale for that utility.

123228 *cpp* This is subsumed by *c99*.

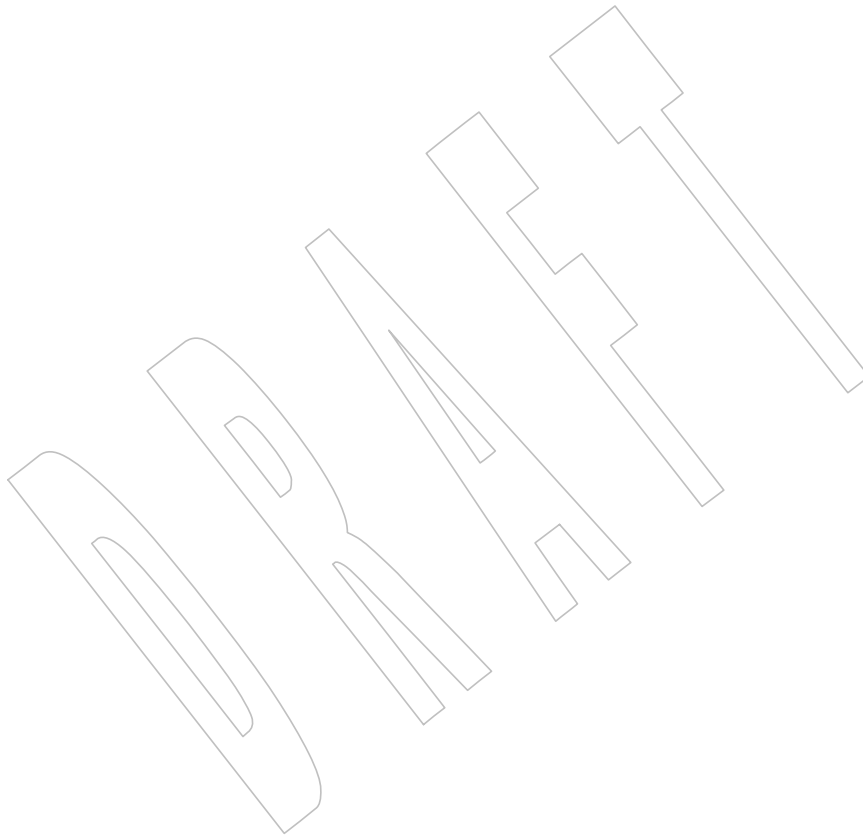
123229 *cu* This utility is terminal-oriented and is not useful from shell scripts or typical  
123230 application programs.

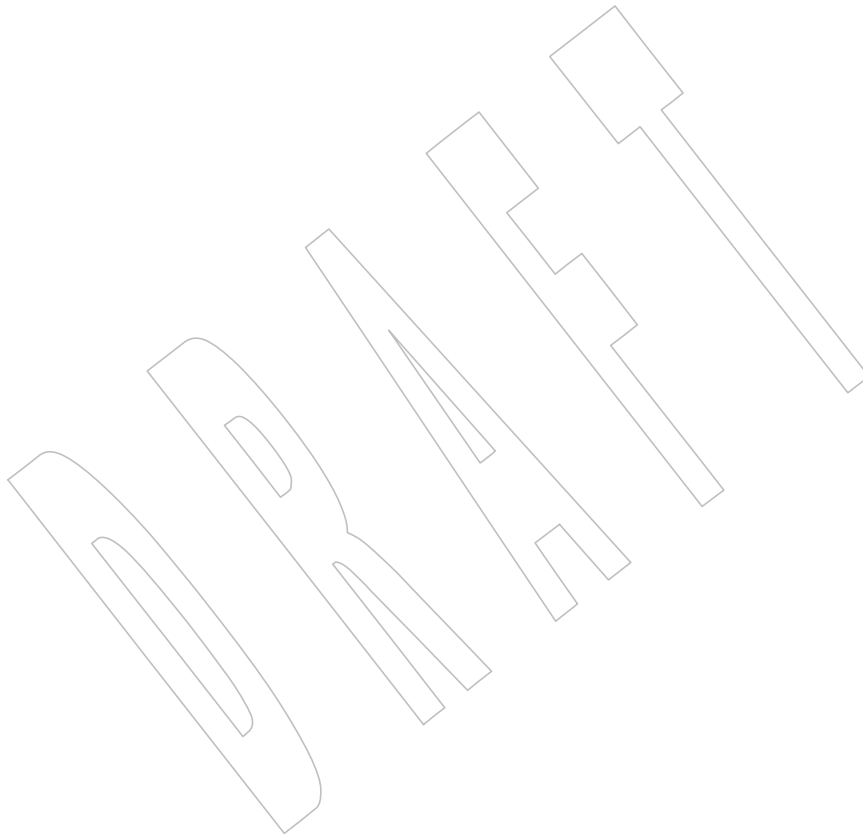


|        |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 123231 | <i>dc</i>     | The functionality of this utility can be provided by the <i>bc</i> utility; <i>bc</i> was selected because it was easier to use and had superior functionality. Although the historical versions of <i>bc</i> are implemented using <i>dc</i> as a base, POSIX.1-200x prescribes the interface and not the underlying mechanism used to implement it.                                                                                                                                                                                                                                                                                                                            |
| 123232 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123233 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123234 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123235 | <i>dircmp</i> | Although a useful concept, the historical output of this directory comparison program is not suitable for processing in application programs. Also, the <i>diff -r</i> command gives equivalent functionality.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 123236 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123237 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123238 | <i>dis</i>    | Disassemblers are hardware-specific.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 123239 | <i>emacs</i>  | The community of <i>emacs</i> editing enthusiasts was adamant that the full <i>emacs</i> editor not be included in the base documents because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original <i>emacs</i> program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include <i>emacs</i> , or included it without supporting it, but there were very few that did not include and support <i>vi</i> . |
| 123240 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123241 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123242 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123243 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123244 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123245 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123246 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123247 | <i>ld</i>     | This is subsumed by <i>c99</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 123248 | <i>line</i>   | The functionality of <i>line</i> can be provided with <i>read</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 123249 | <i>lint</i>   | This technology is partially subsumed by <i>c99</i> . It is also hard to specify the degree of checking for possible error conditions in programs in any compiler, and specifying what <i>lint</i> would do in these cases is equally difficult.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 123250 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123251 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123252 |               | It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, any description of <i>lint</i> is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require <i>lint</i> to detect all errors of any given class.                                                                                                                                                                                                   |
| 123253 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123254 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123255 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123256 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123257 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123258 |               | Additionally, some vendors complained that since many compilers are distributed in a binary form without a <i>lint</i> facility (because the ISO C standard does not require one), implementing the standard as a stand-alone product will be much harder. Rather than being able to build upon a standard compiler component (simply by providing <i>c99</i> as an interface), source to that compiler would most likely need to be modified to provide the <i>lint</i> functionality. This was considered a major burden on system providers for a very small gain to developers (users).                                                                                      |
| 123259 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123260 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123261 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123262 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123263 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123264 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123265 | <i>login</i>  | This utility is terminal-oriented and is not useful from shell scripts or typical application programs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| 123266 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123267 | <i>lorder</i> | This utility is an aid in creating an implementation-defined detail of object libraries that the standard developers did not feel required standardization.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 123268 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123269 | <i>lpstat</i> | The <i>lp</i> system specified is the most basic possible and did not need this level of application control.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 123270 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123271 | <i>mail</i>   | This utility was omitted in favor of <i>mailx</i> because there was a considerable functionality overlap between the two.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 123272 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 123273 | <i>mknod</i>  | This was omitted in favor of <i>mkfifo</i> , as <i>mknod</i> has too many implementation-defined functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 123274 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

|        |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 123275 | <i>news</i>   | This utility is terminal-oriented and is not useful from shell scripts or typical application programs.                                                                                                                                                                                                                                                                                                                                                       |
| 123276 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123277 | <i>pack</i>   | This compression program was considered inferior to <i>compress</i> .                                                                                                                                                                                                                                                                                                                                                                                         |
| 123278 | <i>passwd</i> | This utility was proposed in a historical draft of the base documents but met with too many objections to be included. There were various reasons:                                                                                                                                                                                                                                                                                                            |
| 123279 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123280 |               | <ul style="list-style-type: none"> <li>• Changing a password should not be viewed as a command, but as part of the login sequence. Changing a password should only be done while a trusted path is in effect.</li> </ul>                                                                                                                                                                                                                                      |
| 123281 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123282 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123283 |               | <ul style="list-style-type: none"> <li>• Even though the text in early drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a <i>passwd</i> utility that would conform to POSIX.1-200x for all sites using their system.</li> </ul> |
| 123284 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123285 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123286 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123287 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123288 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123289 |               | <ul style="list-style-type: none"> <li>• This is really a subject for a system administration working group or a security working group.</li> </ul>                                                                                                                                                                                                                                                                                                           |
| 123290 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123291 | <i>pcat</i>   | This compression program was considered inferior to <i>zcat</i> .                                                                                                                                                                                                                                                                                                                                                                                             |
| 123292 | <i>pg</i>     | This duplicated many of the features of the <i>more</i> pager, which was preferred by the standard developers.                                                                                                                                                                                                                                                                                                                                                |
| 123293 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123294 | <i>prof</i>   | The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.                                                                                                                                                                                                                                                    |
| 123295 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123296 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123297 | RCS           | RCS was originally considered as part of a version control utilities portion of the scope. However, this aspect was abandoned by the standard developers. SCCS is now included as an optional part of the XSI option.                                                                                                                                                                                                                                         |
| 123298 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123299 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123300 | <i>red</i>    | Restricted editor. This was not considered by the standard developers because it never provided the level of security restriction required.                                                                                                                                                                                                                                                                                                                   |
| 123301 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123302 | <i>rsh</i>    | Restricted shell. This was not considered by the standard developers because it does not provide the level of security restriction that is implied by historical documentation.                                                                                                                                                                                                                                                                               |
| 123303 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123304 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123305 | <i>sdb</i>    | The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool. Furthermore, some useful aspects of <i>sdb</i> are very hardware-specific.                                                                                                                                                                         |
| 123306 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123307 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123308 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123309 | <i>sdiff</i>  | The “side-by-side <i>diff</i> ” utility from System V was omitted because it is used infrequently, and even less so by conforming applications. Despite being in System V, it is not in the SVID or XPG.                                                                                                                                                                                                                                                      |
| 123310 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123311 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123312 | <i>shar</i>   | Any of the numerous “shell archivers” were excluded because they did not meet the requirement of existing practice.                                                                                                                                                                                                                                                                                                                                           |
| 123313 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123314 | <i>shl</i>    | This utility is terminal-oriented and is not useful from shell scripts or typical application programs. The job control aspects of the shell command language are generally more useful.                                                                                                                                                                                                                                                                      |
| 123315 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123316 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123317 | <i>size</i>   | The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.                                                                                                                                                                                                                                                    |
| 123318 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 123319 |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

|        |               |                                                                                                                                                                                                                                                                                                                                 |
|--------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 123320 | <i>spell</i>  | This utility is not useful from shell scripts or typical application programs. The <i>spell</i> utility was considered, but was omitted because there is no known technology that can be used to make it recognize general language for user-specified input without providing a complete dictionary along with the input file. |
| 123321 |               |                                                                                                                                                                                                                                                                                                                                 |
| 123322 |               |                                                                                                                                                                                                                                                                                                                                 |
| 123323 |               |                                                                                                                                                                                                                                                                                                                                 |
| 123324 | <i>su</i>     | This utility is not useful from shell scripts or typical application programs. (There was also sentiment to avoid security-related utilities.)                                                                                                                                                                                  |
| 123325 |               |                                                                                                                                                                                                                                                                                                                                 |
| 123326 | <i>sum</i>    | This utility was renamed <i>cksum</i> .                                                                                                                                                                                                                                                                                         |
| 123327 | <i>tar</i>    | This has been replaced by <i>pax</i> , for reasons explained in the rationale for that utility.                                                                                                                                                                                                                                 |
| 123328 | <i>unpack</i> | This compression program was considered inferior to <i>uncompress</i> .                                                                                                                                                                                                                                                         |
| 123329 | <i>wall</i>   | This utility is terminal-oriented and is not useful in shell scripts or typical applications. It is generally used only by system administrators.                                                                                                                                                                               |
| 123330 |               |                                                                                                                                                                                                                                                                                                                                 |





123331

# *Rationale (Informative)*

123332

## **Part D:**

123333

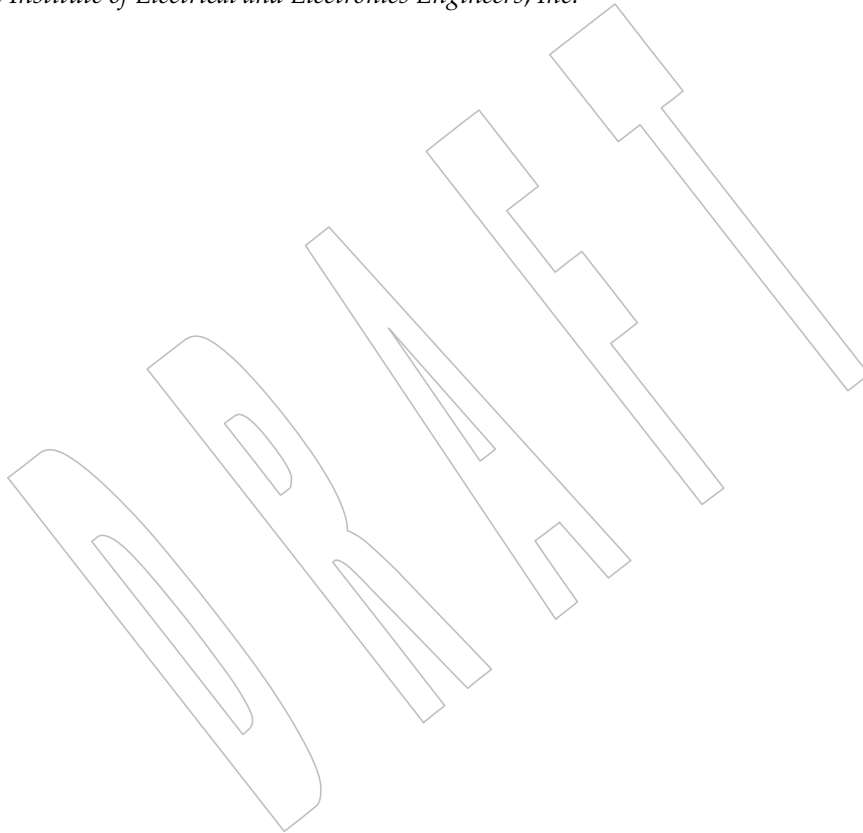
## **Portability Considerations**

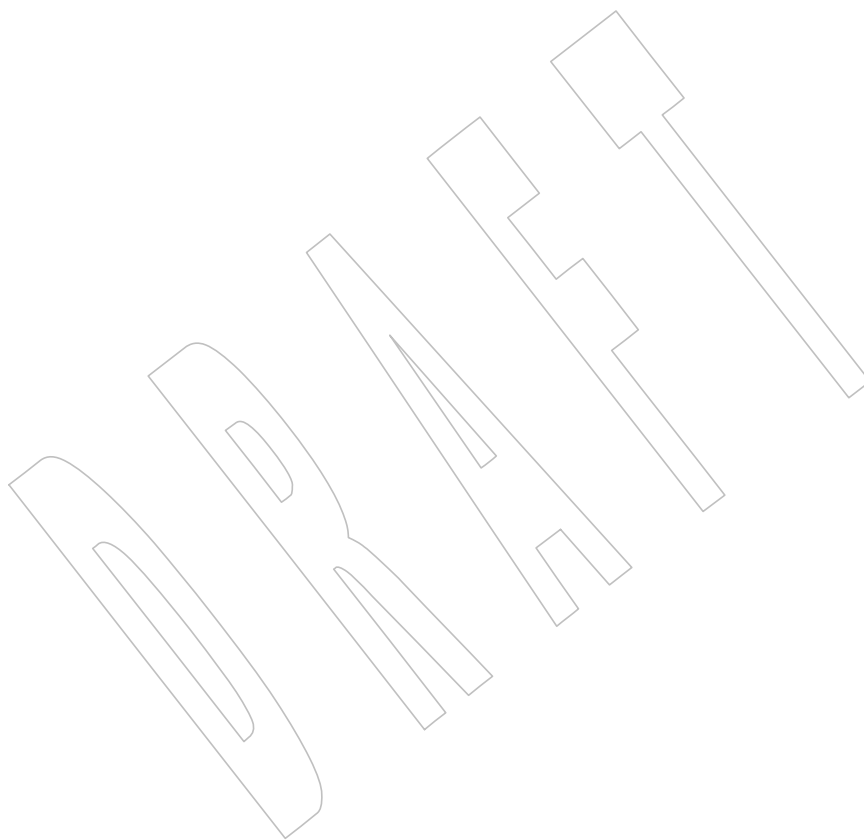
123334

*The Open Group*

123335

*The Institute of Electrical and Electronics Engineers, Inc.*





## Portability Considerations (Informative)

This section contains information to satisfy various international requirements:

- [Section D.1](#) describes perceived user requirements.
- [Section D.2](#) (on page 3586) indicates how the facilities of POSIX.1-200x satisfy those requirements.
- [Section D.3](#) (on page 3594) offers guidance to writers of profiles on how the configurable options, limits, and optional behavior of POSIX.1-200x should be cited in profiles.

### D.1 User Requirements

This section describes the user requirements that were perceived by the standard developers. The primary source for these requirements was an analysis of historical practice in widespread use, as typified by the base documents listed in [Section A.1.1](#) (on page 3313).

POSIX.1-200x addresses the needs of users requiring open systems solutions for source code portability of applications. It currently addresses users requiring open systems solutions for source-code portability of applications involving multi-programming and process management (creating processes, signaling, and so on); access to files and directories in a hierarchy of file systems (opening, reading, writing, deleting files, and so on); access to asynchronous communications ports and other special devices; access to information about other users of the system; facilities supporting applications requiring bounded (realtime) response.

The following users are identified for POSIX.1-200x:

- Those employing applications written in high-level languages, such as C, Ada, or FORTRAN.
- Users who desire conforming applications that do not necessarily require the characteristics of high-level languages (for example, the speed of execution of compiled languages or the relative security of source code intellectual property inherent in the compilation process).
- Users who desire conforming applications that can be developed quickly and can be modified readily without the use of compilers and other system components that may be unavailable on small systems or those without special application development capabilities.
- Users who interact with a system to achieve general-purpose time-sharing capabilities common to most business or government offices or academic environments: editing, filing, inter-user communications, printing, and so on.
- Users who develop applications for POSIX-conformant systems.
- Users who develop applications for UNIX systems.

An acknowledged restriction on applicable users is that they are limited to the group of individuals who are familiar with the style of interaction characteristic of historically-derived systems based on one of the UNIX operating systems (as opposed to other historical systems with different models, such as MS/DOS, Macintosh, VMS, MVS, and so on). Typical users

123375 would include program developers, engineers, or general-purpose time-sharing users.

123376 The requirements of users of POSIX.1-200x can be summarized as a single goal: *application source*  
 123377 *portability*. The requirements of the user are stated in terms of the requirements of portability of  
 123378 applications. This in turn becomes a requirement for a standardized set of syntax and semantics  
 123379 for operations commonly found on many operating systems.

123380 The following sections list the perceived requirements for application portability.

### 123381 **D.1.1 Configuration Interrogation**

123382 An application must be able to determine whether and how certain optional features are  
 123383 provided and to identify the system upon which it is running, so that it may appropriately adapt  
 123384 to its environment.

123385 Applications must have sufficient information to adapt to varying behaviors of the system.

### 123386 **D.1.2 Process Management**

123387 An application must be able to manage itself, either as a single process or as multiple processes.  
 123388 Applications must be able to manage other processes when appropriate.

123389 Applications must be able to identify, control, create, and delete processes, and there must be  
 123390 communication of information between processes and to and from the system.

123391 Applications must be able to use multiple flows of control with a process (threads) and  
 123392 synchronize operations between these flows of control.

### 123393 **D.1.3 Access to Data**

123394 Applications must be able to operate on the data stored on the system, access it, and transmit it  
 123395 to other applications. Information must have protection from unauthorized or accidental access  
 123396 or modification.

### 123397 **D.1.4 Access to the Environment**

123398 Applications must be able to access the external environment to communicate their input and  
 123399 results.

### 123400 **D.1.5 Access to Determinism and Performance Enhancements**

123401 Applications must have sufficient control of resource allocation to ensure the timeliness of  
 123402 interactions with external objects.

### 123403 **D.1.6 Operating System-Dependent Profile**

123404 The capabilities of the operating system may make certain optional characteristics of the base  
 123405 language in effect no longer optional, and this should be specified.



- 123406 **D.1.7 I/O Interaction**
- 123407 The interaction between the C language I/O subsystem (*stdio*) and the I/O subsystem of  
123408 POSIX.1-200x must be specified.
- 123409 **D.1.8 Internationalization Interaction**
- 123410 The effects of the environment of POSIX.1-200x on the internationalization facilities of the C  
123411 language must be specified.
- 123412 **D.1.9 C-Language Extensions**
- 123413 Certain functions in the C language must be extended to support the additional capabilities  
123414 provided by POSIX.1-200x.
- 123415 **D.1.10 Command Language**
- 123416 Users should be able to define procedures that combine simple tools and/or applications into  
123417 higher-level components that perform to the specific needs of the user. The user should be able  
123418 to store, recall, use, and modify these procedures. These procedures should employ a powerful  
123419 command language that is used for recurring tasks in conforming applications (scripts) in the  
123420 same way that it is used interactively to accomplish one-time tasks. The language and the  
123421 utilities that it uses must be consistent between systems to reduce errors and retraining.
- 123422 **D.1.11 Interactive Facilities**
- 123423 Use the system to accomplish individual tasks at an interactive terminal. The interface should be  
123424 consistent, intuitive, and offer usability enhancements to increase the productivity of terminal  
123425 users, reduce errors, and minimize retraining costs. Online documentation or usage assistance  
123426 should be available.
- 123427 **D.1.12 Accomplish Multiple Tasks Simultaneously**
- 123428 Access applications and interactive facilities from a single terminal without requiring serial  
123429 execution: switch between multiple interactive tasks; schedule one-time or periodic background  
123430 work; display the status of all work in progress or scheduled; influence the priority scheduling  
123431 of work, when authorized.
- 123432 **D.1.13 Complex Data Manipulation**
- 123433 Manipulate data in files in complex ways: sort, merge, compare, translate, edit, format, pattern  
123434 match, select subsets (strings, columns, fields, rows, and so on). These facilities should be  
123435 available to both conforming applications and interactive users.
- 123436 **D.1.14 File Hierarchy Manipulation**
- 123437 Create, delete, move/rename, copy, backup/archive, and display files and directories. These  
123438 facilities should be available to both conforming applications and interactive users.

123439 **D.1.15 Locale Configuration**

123440 Customize applications and interactive sessions for the cultural and language conventions of the  
 123441 user. Employ a wide variety of standard character encodings. These facilities should be available  
 123442 to both conforming applications and interactive users.

123443 **D.1.16 Inter-User Communication**

123444 Send messages or transfer files to other users on the same system or other systems on a network.  
 123445 These facilities should be available to both conforming applications and interactive users.

123446 **D.1.17 System Environment**

123447 Display information about the status of the system (activities of users and their interactive and  
 123448 background work, file system utilization, system time, configuration, and presence of optional  
 123449 facilities) and the environment of the user (terminal characteristics, and so on). Inform the  
 123450 system operator/administrator of problems. Control access to user files and other resources.

123451 **D.1.18 Printing**

123452 Output files on a variety of output device classes, accessing devices on local or network-  
 123453 connected systems. Control (or influence) the formatting, priority scheduling, and output  
 123454 distribution of work. These facilities should be available to both conforming applications and  
 123455 interactive users.

123456 **D.1.19 Software Development**

123457 Develop (create and manage source files, compile/interpret, debug) portable open systems  
 123458 applications and package them for distribution to, and updating of, other systems.

123459 **D.2 Portability Capabilities**

123460 This section describes the significant portability capabilities of POSIX.1-200x and indicates how  
 123461 the user requirements listed in [Section D.1](#) (on page 3583) are addressed. The capabilities are  
 123462 listed in the same format as the preceding user requirements; they are summarized below:

- 123463 • Configuration Interrogation
- 123464 • Process Management
- 123465 • Access to Data
- 123466 • Access to the Environment
- 123467 • Access to Determinism and Performance Enhancements
- 123468 • Operating System-Dependent Profile
- 123469 • I/O Interaction
- 123470 • Internationalization Interaction
- 123471 • C-Language Extensions
- 123472 • Command Language
- 123473 • Interactive Facilities

- 123474 • Accomplish Multiple Tasks Simultaneously
- 123475 • Complex Data Manipulation
- 123476 • File Hierarchy Manipulation
- 123477 • Locale Configuration
- 123478 • Inter-User Communication
- 123479 • System Environment
- 123480 • Printing
- 123481 • Software Development

### 123482 D.2.1 Configuration Interrogation

123483 The *uname()* operation provides basic identification of the system. The *sysconf()*, *pathconf()*, and  
 123484 *fpathconf()* functions and the *getconf* utility provide means to interrogate the implementation to  
 123485 determine how to adapt to the environment in which it is running. These values can be either  
 123486 static (indicating that all instances of the implementation have the same value) or dynamic  
 123487 (indicating that different instances of the implementation have the different values, or that the  
 123488 value may vary for other reasons, such as reconfiguration).

#### 123489 Unsatisfied Requirements

123490 None directly. However, as new areas are added, there will be a need for additional capability in  
 123491 this area.

### 123492 D.2.2 Process Management

123493 The *fork()*, *exec* family, *posix\_spawn()*, and *posix\_spawnp()* functions provide for the creation of  
 123494 new processes or the insertion of new applications into existing processes. The *\_Exit()*, *\_exit()*,  
 123495 *exit()*, and *abort()* functions allow for the termination of a process by itself. The *wait()*, *waitid()*,  
 123496 and *waitpid()* functions allow one process to deal with the termination of another.

123497 The *times()* function allows for basic measurement of times used by a process. Various  
 123498 functions, including *fstat()*, *getegid()*, *geteuid()*, *getgid()*, *getgrgid()*, *getgrnam()*, *getlogin()*,  
 123499 *getpid()*, *getppid()*, *getpwnam()*, *getpwuid()*, *getuid()*, *lstat()*, and *stat()*, provide for access to the  
 123500 identifiers of processes and the identifiers and names of owners of processes (and files).

123501 The various functions operating on environment variables provide for communication of  
 123502 information (primarily user-configurable defaults) from a parent to child processes.

123503 The operations on the current working directory control and interrogate the directory from  
 123504 which relative filename searches start. The *umask()* function controls the default protections  
 123505 applied to files created by the process.

123506 The *alarm()*, *pause()*, *sleep()*, *ualarm()*, and *usleep()* operations allow the process to suspend until  
 123507 a timer has expired or to be notified when a period of time has elapsed. The *time()* operation  
 123508 interrogates the current time and date.

123509 The signal mechanism provides for communication of events either from other processes or  
 123510 from the environment to the application, and the means for the application to control the effect  
 123511 of these events. The mechanism provides for external termination of a process and for a process  
 123512 to suspend until an event occurs. The mechanism also provides for a value to be associated with  
 123513 an event.

123514 Job control provides a means to group processes and control them as groups, and to control their

123515 access to the function between the user and the system (the “controlling terminal”). It also  
123516 provides the means to suspend and resume processes.

123517 The Process Scheduling option provides control of the scheduling and priority of a process.

123518 The Message Passing option provides a means for interprocess communication involving small  
123519 amounts of data.

123520 The Memory Management facilities provide control of memory resources and for the sharing of  
123521 memory. This functionality is mandatory on POSIX-conforming systems.

123522 The Threads facilities provide multiple flows of control with a process (threads),  
123523 synchronization between threads (including mutexes, barriers, and spin locks), association of  
123524 data with threads, and controlled cancellation of threads.

123525 The XSI interprocess communications functionality provide an alternate set of facilities to  
123526 manipulate semaphores, message queues, and shared memory. These are provided on XSI-  
123527 conformant systems to support conforming applications developed to run on UNIX systems.

### 123528 D.2.3 Access to Data

123529 The *open()*, *close()*, *fclose()*, *fopen()*, and *pipe()* functions provide for access to files and data.  
123530 Such files may be regular files, interprocess data channels (pipes), or devices. Additional types  
123531 of objects in the file system are permitted and are being contemplated for standardization.

123532 The *access()*, *chmod()*, *chown()*, *dup()*, *dup2()*, *fchmod()*, *fcntl()*, *fstat()*, *ftruncate()*, *lstat()*,  
123533 *readlink()*, *realpath()*, *stat()*, and *utime()* functions allow for control and interrogation of file and  
123534 file-related objects (including symbolic links), and their ownership, protections, and timestamps.

123535 The *fgetc()*, *fputc()*, *fread()*, *fseek()*, *fsetpos()*, *fwrite()*, *getc()*, *getchar()*, *lseek()*, *putchar()*, *putc()*,  
123536 *read()*, and *write()* functions provide for data transfer from the application to files (in all their  
123537 forms).

123538 The *closedir()*, *link()*, *mkdir()*, *opendir()*, *readdir()*, *rename()*, *rmdir()*, *rewinddir()*, and *unlink()*  
123539 functions provide for a complete set of operations on directories. Directories can arbitrarily  
123540 contain other directories, and a single file can be mentioned in more than one directory.

123541 The *faccessat()*, *openat()*, *fchmodat()*, *fchownat()*, *fstatat()*, *linkat()*, *renameat()*, *readlinkat()*,  
123542 *symlinkat()*, and *unlinkat()* functions allow for race-free and thread-safe file access. The  
123543 motivation for the introduction of these functions was as follows:

- 123544 • Interfaces taking a pathname are limited by the maximum length of a pathname  
123545 (`_SC_PATH_MAX`). The absolute path of files can far exceed this length. The alternative  
123546 solution of changing the working directory and using relative pathnames is not thread-  
123547 safe.
- 123548 • A second motivation is that files accessed outside the current working directory are subject  
123549 to attacks caused by the race condition created by changing any of the elements of the  
123550 pathnames used.
- 123551 • A third motivation is to allow application code which makes use of a virtual current  
123552 working directory for each individual thread. In the alternative model there is only one  
123553 current working directory for all threads.

123554 The file-locking mechanism provides for advisory locking (protection during transactions) of  
123555 ranges of bytes (in effect, records) in a file.

123556 The *confstr()*, *fpathconf()*, *pathconf()*, and *sysconf()* functions provide for enquiry as to the  
123557 behavior of the system where variability is permitted.

123558 The asynchronous input and output functions *aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*,

123559  *aio\_return()*,  *aio\_suspend()*,  *aio\_write()*, and  *lio\_listio()* provide for initiation and control of  
 123560 asynchronous data transfers.

123561 The Synchronized Input and Output option provides for assured commitment of data to media.

#### 123562 **D.2.4 Access to the Environment**

123563 The operations and types in XBD are provided for access to asynchronous serial devices. The  
 123564 primary intended use for these is the controlling terminal for the application (the interaction  
 123565 point between the user and the system). They are general enough to be used to control any  
 123566 asynchronous serial device. The functions are also general enough to be used with many other  
 123567 device types as a user interface when some emulation is provided.

123568 Less detailed access is provided for other device types, but in many instances an application  
 123569 need not know whether an object in the file system is a device or a regular file to operate  
 123570 correctly.

#### 123571 **Unsatisfied Requirements**

123572 Detailed control of common device classes, specifically magnetic tape, is not provided.

#### 123573 **D.2.5 Bounded (Realtime) Response**

123574 The realtime signal functions  *sigqueue()*,  *sigtimedwait()*, and  *sigwaitinfo()* provide queued signals  
 123575 and the prioritization of the handling of signals.

123576 The SCHED\_FIFO, SCHED\_SPORADIC, and SCHED\_RR scheduling policies provide control  
 123577 over processor allocation.

123578 The semaphore functions  *sem\_close()*,  *sem\_destroy()*,  *sem\_getvalue()*,  *sem\_init()*,  *sem\_open()*,  
 123579  *sem\_post()*,  *sem\_timedwait()*,  *sem\_trywait()*,  *sem\_unlink()*, and  *sem\_wait()* provide high-  
 123580 performance synchronization.

123581 The memory management functions provide memory locking for control of memory allocation,  
 123582 file mapping for high performance, and shared memory for high-performance interprocess  
 123583 communication. The Message Passing option provides for interprocess communication without  
 123584 being dependent on shared memory.

123585 The timers functions  *clock\_getres()*,  *clock\_gettime()*,  *clock\_settime()*,  *nanosleep()*,  *timer\_create()*,  
 123586  *timer\_delete()*,  *timer\_getoverrun()*,  *timer\_gettime()*, and  *timer\_settime()* provide functionality to  
 123587 manipulate clocks and timers and include a high resolution function called  *nanosleep()* with a  
 123588 finer resolution than the  *sleep()* function.

123589 The timeout functions —  *pthread\_mutex\_timedlock()*,  *pthread\_rwlock\_timedrdlock()*,  
 123590  *pthread\_rwlock\_timedwrlock()*, and  *sem\_timedwait()* — the Typed Memory Objects option and the  
 123591 Monotonic Clock option provide further facilities for applications to use to obtain predictable  
 123592 bounded response.

## 123593 D.2.6 Operating System-Dependent Profile

123594 POSIX.1-200x makes no distinction between text and binary files. The values of EXIT\_SUCCESS  
123595 and EXIT\_FAILURE are further defined.

### 123596 Unsatisfied Requirements

123597 None known, but the ISO C standard may contain some additional options that could be  
123598 specified.

## 123599 D.2.7 I/O Interaction

123600 POSIX.1-200x defines how each of the ISO C standard *stdio* functions interact with the POSIX.1  
123601 operations, typically specifying the behavior in terms of POSIX.1 operations.

### 123602 Unsatisfied Requirements

123603 None.

## 123604 D.2.8 Internationalization Interaction

123605 The POSIX.1-200x environment operations provide a means to define the environment for  
123606 *setlocale()* and time functions such as *ctime()*. The *tzset()* function is provided to set time  
123607 conversion information.

123608 The *nl\_langinfo()* function is provided to query locale-specific cultural settings.

123609 The multiple concurrent locale functions *duplocale()*, *freelocale()*, *is\*\_l()*, *newlocale()*,  
123610 *strcasecmp\_l()*, *strcoll\_l()*, *strfmon\_l()*, *strncasecmp\_l()*, *strxfrm\_l()*, *tolower\_l()*, *toupper\_l()*,  
123611 *towctrans\_l()*, *towlower\_l()*, *towupper\_l()*, *uselocale()*, *wscasecmp\_l()*, *wscoll\_l()*, *wscnscmp\_l()*,  
123612 *wcsxfrm\_l()*, *wctrans\_l()*, and *wctype\_l()* are provide to support per-thread locale information.

### 123613 Unsatisfied Requirements

123614 None.

## 123615 D.2.9 C-Language Extensions

123616 The *setjmp()* and *longjmp()* functions are not defined to be cognizant of the signal masks defined  
123617 for POSIX.1. The *sigsetjmp()* and *siglongjmp()* functions are provided to fill this gap.

123618 The *\_setjmp()* and *\_longjmp()* functions are provided as XSI options to support historic practice.

### 123619 Unsatisfied Requirements

123620 None.

## 123621 D.2.10 Command Language

123622 The shell command language, as described in XCU [Chapter 2](#) (on page 2245), is a common  
123623 language useful in batch scripts, through an API to high-level languages (for the C-Language  
123624 Binding option, *system()* and *popen()*) and through an interactive terminal (see the *sh* utility).  
123625 The shell language has many of the characteristics of a high-level language, but it has been  
123626 designed to be more suitable for user terminal entry and includes interactive debugging  
123627 facilities. Through the use of pipelining, many complex commands can be constructed from  
123628 combinations of data filters and other common components. Shell scripts can be created, stored,  
123629 recalled, and modified by the user with simple editors.

123630 In addition to the basic shell language, the following utilities offer features that simplify and  
 123631 enhance programmatic access to the utilities and provide features normally found only in high-  
 123632 level languages: *basename*, *bc*, *command*, *dirname*, *echo*, *env*, *expr*, *false*, *printf*, *read*, *sleep*, *tee*, *test*,  
 123633 *time\**,<sup>8</sup> *true*, *wait*, *xargs*, and all of the special built-in utilities in XCU Section 2.14 (on page 2280).

#### 123634 Unsatisfied Requirements

123635 None.

### 123636 D.2.11 Interactive Facilities

123637 The utilities offer a common style of command-line interface through conformance to the Utility  
 123638 Syntax Guidelines (see XBD Section 12.2, on page 201) and the common utility defaults (see XCU  
 123639 Section 1.4, on page 2235). The *sh* utility offers an interactive command-line history and editing  
 123640 facility.

123641 The following utilities can be used interactively as well as by scripts; *alias*, *fc*, *mailx*, *unalias*, and  
 123642 *write*.

123643 The following utilities in the User Portability Utilities option provide for interactive use: *ex*, *more*,  
 123644 and *vi*; the *man* utility offers online access to system documentation.

#### 123645 Unsatisfied Requirements

123646 The command line interface to individual utilities is as intuitive and consistent as historical  
 123647 practice allows. Work underway based on graphical user interfaces may be more suitable for  
 123648 novice or occasional users of the system.

### 123649 D.2.12 Accomplish Multiple Tasks Simultaneously

123650 The shell command language offers background processing through the asynchronous list  
 123651 command form; see XCU Section 2.9 (on page 2263).

123652 The *nohup* utility makes background processing more robust and usable.

123653 The *kill* utility can terminate background jobs.

123654 The following utilities support periodic job scheduling, control, and display: *at*, *batch*, *crontab*,  
 123655 *nice*, *ps*, and *renice*.

123656 When the User Portability Utilities option is supported, the following utilities allow  
 123657 manipulation of jobs: *bg*, *fg*, and *jobs*.

#### 123658 Unsatisfied Requirements

123659 Terminals with multiple windows may be more suitable for some multi-tasking interactive uses  
 123660 than the job control approach in POSIX.1-200x. See the comments on graphical user interfaces in  
 123661 Section D.2.11. The *nice* and *renice* utilities do not necessarily take advantage of complex system  
 123662 scheduling algorithms that are supported by the realtime options within POSIX.1-200x.

---

123663 8. The utilities listed with an asterisk here and later in this section are present only on systems which support the User Portability Utilities  
 123664 option. There may be further restrictions on the utilities offered with various configuration option combinations; see the individual utility  
 123665 descriptions.

123666 **D.2.13 Complex Data Manipulation**

123667 The following utilities address user requirements in this area: *asa*, *awk*, *bc*, *cmp*, *comm*, *csplit*, *cut*,  
 123668 *dd*, *diff*, *ed*, *ex\**, *expand*, *expr*, *find*, *fold*, *grep*, *head*, *join*, *od*, *paste*, *pr*, *printf*, *sed*, *sort*, *split*, *tabs*, *tail*, *tr*,  
 123669 *unexpand*, *uniq*, *uudecode*, *uuencode*, and *wc*.

123670 **Unsatisfied Requirements**

123671 Sophisticated text formatting utilities, such as *troff* or *TeX*, are not included. Standards work in  
 123672 the area of SGML may satisfy this.

123673 **D.2.14 File Hierarchy Manipulation**

123674 The following utilities address user requirements in this area: *basename*, *cd*, *chgrp*, *chmod*, *chown*,  
 123675 *cksum*, *cp*, *dd*, *df*, *diff*, *dirname*, *du*, *find*, *ls*, *ln*, *mkdir*, *mkfifo*, *mv*, *patch*, *pathchk*, *pax*, *pwd*, *rm*, *rmdir*,  
 123676 *test*, and *touch*.

123677 **Unsatisfied Requirements**

123678 Some graphical user interfaces offer more intuitive file manager components that allow file  
 123679 manipulation through the use of icons for novice users.

123680 **D.2.15 Locale Configuration**

123681 The standard utilities are affected by the various *LC\_* variables to achieve locale-dependent  
 123682 operation: character classification, collation sequences, regular expressions and shell pattern  
 123683 matching, date and time formats, numeric formatting, and monetary formatting. When the  
 123684 *POSIX2\_LOCALEDEF* option is supported, applications can provide their own locale definition  
 123685 files.

123686 The following utilities address user requirements in this area: *date*, *ed*, *ex\**, *find*, *grep*, *locale*,  
 123687 *localedef*, *more\**, *sed*, *sh*, *sort*, *tr*, *uniq*, and *vi\**.

123688 The *iconv()*, *iconv\_close()*, and *iconv\_open()* functions are available to allow an application to  
 123689 convert character data between supported character sets.

123690 The *genccat* utility and the *catopen()*, *catclose()*, and *catgets()* functions provide for message  
 123691 catalog manipulation.

123692 **Unsatisfied Requirements**

123693 Some aspects of multi-byte character and state-encoded character encodings have not yet been  
 123694 addressed. The C-language functions, such as *getopt()*, are generally limited to single-byte  
 123695 characters. The effect of the *LC\_MESSAGES* variable on message formats is only suggested at  
 123696 this time.

123697 **D.2.16 Inter-User Communication**

123698 The following utilities address user requirements in this area: *cksum*, *mailx*, *mesg*, *patch*, *pax*, *talk*,  
 123699 *uudecode*, *uuencode*, *who*, and *write*.

123700 The historical UUCP utilities are included as a separate UUCP Utilities option.



123701 **Unsatisfied Requirements**

123702 None.

123703 **D.2.17 System Environment**123704 The following utilities address user requirements in this area: *chgrp*, *chmod*, *chown*, *df*, *du*, *env*,  
123705 *getconf*, *id*, *logger*, *logname*, *mesg*, *newgrp*, *ps*, *stty*, *tput*, *tty*, *umask*, *uname*, and *who*.123706 The *closelog()*, *openlog()*, *setlogmask()*, and *syslog()* functions provide system logging facilities on  
123707 XSI-conformant systems; these are analogous to the *logger* utility.123708 **Unsatisfied Requirements**

123709 None.

123710 **D.2.18 Printing**123711 The following utilities address user requirements in this area: *pr* and *lp*.123712 **Unsatisfied Requirements**

123713 There are no features to control the formatting or scheduling of the print jobs.

123714 **D.2.19 Software Development**123715 The following utilities address user requirements in this area: *ar*, *asa*, *awk*, *c99*, *ctags*, *fort77*,  
123716 *getconf*, *getopts*, *lex*, *localedef*, *make*, *nm*, *od*, *patch*, *pax*, *strings*, *strip*, *time*, and *yacc*.123717 The *system()*, *popen()*, *pclose()*, *regcomp()*, *regex()*, *regerror()*, *regfree()*, *fnmatch()*, *getopt()*,  
123718 *glob()*, *globfree()*, *wordexp()*, and *wordfree()* functions allow C-language programmers to access  
123719 some of the interfaces used by the utilities, such as argument processing, regular expressions,  
123720 and pattern matching.123721 The SCCS source-code control system utilities are available on systems supporting the XSI  
123722 Development option.123723 **Unsatisfied Requirements**123724 There are no language-specific development tools related to languages other than C and  
123725 FORTRAN. The C tools are more complete and varied than the FORTRAN tools. There is no  
123726 data dictionary or other CASE-like development tools.123727 **D.2.20 Future Growth**123728 It is arguable whether or not all functionality to support applications is potentially within the  
123729 scope of POSIX.1-200x. As a simple matter of practicality, it cannot be. Areas such as graphics,  
123730 application domain-specific functionality, windowing, and so on, should be in unique standards.  
123731 As such, they are properly “Unsatisfied Requirements” in terms of providing fully conforming  
123732 applications, but ones which are outside the scope of POSIX.1-200x.123733 However, as the standards evolve, certain functionality once considered “exotic” enough to be  
123734 part of a separate standard become common enough to be included in a core standard such as  
123735 this. Realtime and networking, for example, have both moved from separate standards (with  
123736 much difficult cross-referencing) into this standard over time, and although no specific areas  
123737 have been identified for inclusion in a future version, such inclusions seem likely.

### 123738 D.3 Profiling Considerations

123739 This section offers guidance to writers of profiles on how the configurable options, limits, and  
 123740 optional behavior of POSIX.1-200x should be cited in profiles. Profile writers should consult the  
 123741 general guidance in POSIX.0 when writing POSIX Standardized Profiles.

123742 The information in this section is an inclusive list of features that should be considered by profile  
 123743 writers. Subsetting of POSIX.1-200x should follow XBD [Section 2.1.5.1](#) (on page 20). A set of  
 123744 profiling options is described in [Appendix E](#) (on page 3607).

#### 123745 D.3.1 Configuration Options

123746 There are two set of options suggested by POSIX.1-200x: those for POSIX-conforming systems  
 123747 and those for X/Open System Interface (XSI) conformance. The requirements for XSI  
 123748 conformance are documented in the Base Definitions volume of POSIX.1-200x and not discussed  
 123749 further here, as they superset the POSIX conformance requirements.

#### 123750 D.3.2 Configuration Options (Shell and Utilities)

123751 There are three broad optional configurations for the Shell and Utilities volume of POSIX.1-200x:  
 123752 basic execution system, development system, and user portability interactive system. The  
 123753 options to support these, and other minor configuration options, are listed in XBD [Chapter 2](#) (on  
 123754 page 15). Profile writers should consult the following list and the comments concerning user  
 123755 requirements addressed by various components in [Section D.2](#) (on page 3586).

##### 123756 POSIX2\_UPE

123757 The system supports the User Portability Utilities option.

123758 This option is a requirement for a user portability interactive system. It is required  
 123759 frequently except for those systems, such as embedded realtime or dedicated application  
 123760 systems, that support little or no interactive time-sharing work by users or operators. XSI-  
 123761 conformant systems support this option.

##### 123762 POSIX2\_SW\_DEV

123763 The system supports the Software Development Utilities option.

123764 This option is required by many systems, even those in which actual software development  
 123765 does not occur. The *make* utility, in particular, is required by many application software  
 123766 packages as they are installed onto the system. If POSIX2\_C\_DEV is supported,  
 123767 POSIX2\_SW\_DEV is almost a mandatory requirement because of *ar* and *make*.

##### 123768 POSIX2\_C\_BIND

123769 The system supports the C-Language Bindings option.

123770 This option is required on some implementations developing complex C applications or on  
 123771 any system installing C applications in source form that require the functions in this option.  
 123772 The *system()* and *popen()* functions, in particular, are widely used by applications; the  
 123773 others are rather more specialized.

##### 123774 POSIX2\_C\_DEV

123775 The system supports the C-Language Development Utilities option.

123776 This option is required by many systems, even those in which actual C-language software  
 123777 development does not occur. The *c99* utility, in particular, is required by many application  
 123778 software packages as they are installed onto the system. The *lex* and *yacc* utilities are used  
 123779 less frequently.

- 123780 POSIX2\_FORT\_DEV  
 123781 The system supports the FORTRAN Development Utilities option
- 123782 As with C, this option is needed on any system developing or installing FORTRAN  
 123783 applications in source form.
- 123784 POSIX2\_FORT\_RUN  
 123785 The system supports the FORTRAN Runtime Utilities option.
- 123786 This option is required for some FORTRAN applications that need the *asa* utility to convert  
 123787 Hollerith printing statement output. It is unknown how frequently this occurs.
- 123788 POSIX2\_LOCALEDEF  
 123789 The system supports the creation of locales.
- 123790 This option is needed if applications require their own customized locale definitions to  
 123791 operate. It is presently unknown whether many applications are dependent on this.  
 123792 However, the option is virtually mandatory for systems in which internationalized  
 123793 applications are developed.
- 123794 XSI-conformant systems support this option.
- 123795 POSIX2\_PBS  
 123796 The system supports the Batch Environment Services and Utilities option.
- 123797 POSIX2\_PBS\_ACCOUNTING  
 123798 The system supports the optional feature of accounting within the Batch Environment  
 123799 Services and Utilities option. It will be required in servers that implement the optional  
 123800 feature of accounting.
- 123801 POSIX2\_PBS\_CHECKPOINT  
 123802 The system supports the optional feature of checkpoint/restart within the Batch  
 123803 Environment Services and Utilities option.
- 123804 POSIX2\_PBS\_LOCATE  
 123805 The system supports the optional feature of locating batch jobs within the Batch  
 123806 Environment Services and Utilities option.
- 123807 POSIX2\_PBS\_MESSAGE  
 123808 The system supports the optional feature of sending messages to batch jobs within the Batch  
 123809 Environment Services and Utilities option.
- 123810 POSIX2\_PBS\_TRACK  
 123811 The system supports the optional feature of tracking batch jobs within the Batch  
 123812 Environment Services and Utilities option.
- 123813 POSIX2\_CHAR\_TERM  
 123814 The system supports at least one terminal type capable of all operations described in  
 123815 POSIX.1-200x.
- 123816 On systems with POSIX2\_UPE, this option is almost always required. It was developed  
 123817 solely to allow certain specialized vendors and user applications to bypass the requirement  
 123818 for general-purpose asynchronous terminal support. For example, an application and  
 123819 system that was suitable for block-mode terminals, such as IBM 3270s, would not need this  
 123820 option.
- 123821 XSI-conformant systems support this option.

**D.3.3 Configurable Limits**

Very few of the limits need to be increased for profiles. No profile can cite lower values.

{POSIX2\_BC\_BASE\_MAX}

{POSIX2\_BC\_DIM\_MAX}

{POSIX2\_BC\_SCALE\_MAX}

{POSIX2\_BC\_STRING\_MAX}

No increase is anticipated for any of these *bc* values, except for very specialized applications involving huge numbers.

{POSIX2\_COLL\_WEIGHTS\_MAX}

Some natural languages with complex collation requirements require an increase from the default 2 to 4; no higher numbers are anticipated.

{POSIX2\_EXPR\_NEST\_MAX}

No increase is anticipated.

{POSIX2\_LINE\_MAX}

This number is much larger than most historical applications have been able to use. At some future time, applications may be rewritten to take advantage of even larger values.

{POSIX2\_RE\_DUP\_MAX}

No increase is anticipated.

{POSIX2\_VERSION}

This is actually not a limit, but a standard version stamp. Generally, a profile should specify XCU [Chapter 2](#) (on page 2245) by name in the normative references section, not this value.

**D.3.4 Configuration Options (System Interfaces)**

{NGROUPS\_MAX}

A non-zero value indicates that the implementation supports supplementary groups.

This option is needed where there is a large amount of shared use of files, but where a certain amount of protection is needed. Many profiles<sup>9</sup> are known to require this option; it should only be required if needed, but it should never be prohibited.

\_POSIX\_ADVISORY\_INFO

The system provides advisory information for file management.

This option allows the application to specify advisory information that can be used to achieve better or even deterministic response time in file manager or input and output operations.

\_POSIX\_ASYNCHRONOUS\_IO

Support for asynchronous input and output is mandatory in POSIX.1-200x.

\_POSIX\_BARRIERS

Support for barrier synchronization is mandatory in POSIX.1-200x.

This facility allows efficient synchronization of multiple parallel threads in multi-processor systems in which the operation is supported in part by the hardware architecture.

9. There are no formally approved profiles of POSIX.1-200x at the time of publication; the reference here is to various profiles generated by private bodies or governments.

- 123862 `_POSIX_CHOWN_RESTRICTED`  
 123863 The system restricts the right to “give away” files to other users. It is mandatory that an  
 123864 implementation be able to support this facility in POSIX.1-200x; however, it is recognized  
 123865 that implementations need not enable the functionality by default.
- 123866 Some applications expect that they can change the ownership of files in this way. It is  
 123867 provided where either security or system account requirements cause this ability to be a  
 123868 problem. It is also known to be specified in many profiles.
- 123869 `_POSIX_CLOCK_SELECTION`  
 123870 Support for clock selection is mandatory in POSIX.1-200x.
- 123871 This facility allows applications to request a high resolution sleep in order to suspend a  
 123872 thread during a relative time interval, or until an absolute time value, using the desired  
 123873 clock. It also allows the application to select the clock used in a `pthread_cond_timedwait()`  
 123874 function call.
- 123875 `_POSIX_CPUTIME`  
 123876 The system supports the Process CPU-Time Clocks option.
- 123877 This option allows applications to use a new clock that measures the execution times of  
 123878 processes or threads, and the possibility to create timers based upon these clocks, for  
 123879 runtime detection (and treatment) of execution time overruns.
- 123880 `_POSIX_FSYNC`  
 123881 The system supports file synchronization requests.
- 123882 This option was created to support historical systems that did not provide the feature.  
 123883 Applications that are expecting guaranteed completion of their input and output operations  
 123884 should require the `_POSIX_SYNC_IO` option. This option should never be prohibited.
- 123885 XSI-conformant systems support this option.
- 123886 `_POSIX_IPV6`  
 123887 The system supports facilities related to Internet Protocol Version 6 (IPv6).
- 123888 This option was created to allow systems to transition to IPv6.
- 123889 `_POSIX_JOB_CONTROL`  
 123890 Support for job control is mandatory in POSIX.1-200x.
- 123891 Most applications that use it can run when it is not present, although with a degraded level  
 123892 of user convenience.
- 123893 `_POSIX_MAPPED_FILES`  
 123894 Support for memory mapped files is mandatory in POSIX.1-200x.
- 123895 This facility provides for the mapping of regular files into the process address space.
- 123896 Both this facility and the Shared Memory Objects option provide shared access to memory  
 123897 objects in the process address space. The `mmap()` and `munmap()` functions provide the  
 123898 functionality of existing practice for mapping regular files. This functionality was deemed  
 123899 unnecessary, if not inappropriate, for embedded systems applications and is expected to be  
 123900 optional in subprofiles.
- 123901 `_POSIX_MEMLOCK`  
 123902 The system supports the locking of the address space.
- 123903 This option was created to support historical systems that did not provide the feature. It  
 123904 should only be required if needed, but it should never be prohibited.

- 123905 `_POSIX_MEMLOCK_RANGE`  
 123906 The system supports the locking of specific ranges of the address space.
- 123907 For applications that have well-defined sections that need to be locked and others that do  
 123908 not, POSIX.1-200x supports an optional set of functions to lock or unlock a range of process  
 123909 addresses. The following are two reasons for having a means to lock down a specific range:
- 123910 1. An asynchronous event handler function that must respond to external events in a  
 123911 deterministic manner such that page faults cannot be tolerated
  - 123912 2. An input/output “buffer” area that is the target for direct-to-process I/O, and the  
 123913 overhead of implicit locking and unlocking for each I/O call cannot be tolerated
- 123914 It should only be required if needed, but it should never be prohibited.
- 123915 `_POSIX_MEMORY_PROTECTION`  
 123916 Support for memory protection is mandatory in POSIX.1-200x.
- 123917 The provision of this facility typically imposes additional hardware requirements.
- 123918 `_POSIX_PRIORITIZED_IO`  
 123919 The system provides prioritization for input and output operations.
- 123920 The use of this option may interfere with the ability of the system to optimize input and  
 123921 output throughput. It should only be required if needed, but it should never be prohibited.
- 123922 `_POSIX_MESSAGE_PASSING`  
 123923 The system supports the passing of messages between processes.
- 123924 This option was created to support historical systems that did not provide the feature. The  
 123925 functionality adds a high-performance XSI interprocess communication facility for local  
 123926 communication. It should only be required if needed, but it should never be prohibited.
- 123927 `_POSIX_MONOTONIC_CLOCK`  
 123928 The system supports the Monotonic Clock option.
- 123929 This option allows realtime applications to rely on a monotonically increasing clock that  
 123930 does not jump backwards, and whose value does not change except for the regular ticking  
 123931 of the clock.
- 123932 `_POSIX_PRIORITY_SCHEDULING`  
 123933 The system provides priority-based process scheduling.
- 123934 Support of this option provides predictable scheduling behavior, allowing applications to  
 123935 determine the order in which processes that are ready to run are granted access to a  
 123936 processor. It should only be required if needed, but it should never be prohibited.
- 123937 `_POSIX_REALTIME_SIGNALS`  
 123938 Support for realtime signals is mandatory in POSIX.1-200x.
- 123939 This facility provides prioritized, queued signals with associated data values.
- 123940 `_POSIX_REGEX`  
 123941 Support for regular expression facilities is mandatory in POSIX.1-200x.
- 123942 `_POSIX_SAVED_IDS`  
 123943 Support for this feature is mandatory in POSIX.1-200x.
- 123944 Certain classes of applications rely on it for proper operation, and there is no alternative  
 123945 short of giving the application root privileges on most implementations that did not provide  
 123946 `_POSIX_SAVED_IDS`.

- 123947 `_POSIX_SEMAPHORES`  
 123948 Support for counting semaphores is mandatory in POSIX.1-200x.
- 123949 `_POSIX_SHARED_MEMORY_OBJECTS`  
 123950 The system supports the mapping of shared memory objects into the process address space.  
 123951 Both this option and the Memory Mapped Files option provide shared access to memory  
 123952 objects in the process address space. The functions defined under this option provide the  
 123953 functionality of existing practice for shared memory objects. This functionality was deemed  
 123954 appropriate for embedded systems applications and, hence, is provided under this option.  
 123955 It should only be required if needed, but it should never be prohibited.
- 123956 `_POSIX_SHELL`  
 123957 Support for the *sh* utility command line interpreter is mandatory in POSIX.1-200x.
- 123958 `_POSIX_SPAWN`  
 123959 The system supports the spawn option.  
 123960 This option provides applications with an efficient mechanism to spawn execution of a new  
 123961 process.
- 123962 `_POSIX_SPINLOCKS`  
 123963 Support for spin locks is mandatory in POSIX.1-200x.  
 123964 This facility provides a simple and efficient synchronization mechanism for threads  
 123965 executing in multi-processor systems.
- 123966 `_POSIX_SPORADIC_SERVER`  
 123967 The system supports the sporadic server scheduling policy.  
 123968 This option provides applications with a new scheduling policy for scheduling aperiodic  
 123969 processes or threads in hard realtime applications.
- 123970 `_POSIX_SYNCHRONIZED_IO`  
 123971 The system supports guaranteed file synchronization.  
 123972 This option was created to support historical systems that did not provide the feature.  
 123973 Applications that are expecting guaranteed completion of their input and output operations  
 123974 should require this option, rather than the File Synchronization option. It should only be  
 123975 required if needed, but it should never be prohibited.
- 123976 `_POSIX_THREADS`  
 123977 Support for multiple threads of control within a single process is mandatory in  
 123978 POSIX.1-200x.
- 123979 `_POSIX_THREAD_ATTR_STACKADDR`  
 123980 The system supports specification of the stack address for a created thread.  
 123981 Applications may take advantage of support of this option for performance benefits, but  
 123982 dependence on this feature should be minimized. This option should never be prohibited.  
 123983 XSI-conformant systems support this option.
- 123984 `_POSIX_THREAD_ATTR_STACKSIZE`  
 123985 The system supports specification of the stack size for a created thread.  
 123986 Applications may require this option in order to ensure proper execution, but such usage  
 123987 limits portability and dependence on this feature should be minimized. It should only be  
 123988 required if needed, but it should never be prohibited.  
 123989 XSI-conformant systems support this option.

- 123990        \_posix\_thread\_priority\_scheduling
- 123991        The system provides priority-based thread scheduling.
- 123992        Support of this option provides predictable scheduling behavior, allowing applications to
- 123993        determine the order in which threads that are ready to run are granted access to a processor.
- 123994        It should only be required if needed, but it should never be prohibited.
- 123995        \_posix\_thread\_prio\_inherit
- 123996        The system provides mutual-exclusion operations with priority inheritance.
- 123997        Support of this option provides predictable scheduling behavior, allowing applications to
- 123998        determine the order in which threads that are ready to run are granted access to a processor.
- 123999        It should only be required if needed, but it should never be prohibited.
- 124000        \_posix\_thread\_prio\_protect
- 124001        The system supports a priority ceiling emulation protocol for mutual-exclusion operations.
- 124002        Support of this option provides predictable scheduling behavior, allowing applications to
- 124003        determine the order in which threads that are ready to run are granted access to a processor.
- 124004        It should only be required if needed, but it should never be prohibited.
- 124005        \_posix\_thread\_process\_shared
- 124006        The system provides shared access among multiple processes to synchronization objects.
- 124007        This option was created to support historical systems that did not provide the feature. It
- 124008        should only be required if needed, but it should never be prohibited.
- 124009        XSI-conformant systems support this option.
- 124010        \_posix\_thread\_safe\_functions
- 124011        Support for thread-safe functions is mandatory in POSIX.1-200x.
- 124012        \_posix\_thread\_sporadic\_server
- 124013        The system supports the thread sporadic server scheduling policy.
- 124014        Support for this option provides applications with a new scheduling policy for scheduling
- 124015        aperiodic threads in hard realtime applications.
- 124016        \_posix\_timeouts
- 124017        Support for timeouts for some blocking services is mandatory in POSIX.1-200x.
- 124018        \_posix\_timers
- 124019        Support for higher resolution clocks with multiple timers per process is mandatory in
- 124020        POSIX.1-200x.
- 124021        This facility is appropriate for applications requiring higher resolution timestamps or
- 124022        needing to control the timing of multiple activities.
- 124023        \_posix\_trace
- 124024        The system supports the Trace option.
- 124025        This option was created to allow applications to perform tracing.
- 124026        \_posix\_trace\_event\_filter
- 124027        The system supports the Trace Event Filter option.
- 124028        This option is dependent on support of the Trace option.
- 124029        \_posix\_trace\_inherit
- 124030        The system supports the Trace Inherit option.
- 124031        This option is dependent on support of the Trace option.



- 124032        \_POSIX\_TRACE\_LOG  
124033            The system supports the Trace Log option.
- 124034            This option is dependent on support of the Trace option.
- 124035        \_POSIX\_TYPED\_MEMORY\_OBJECTS  
124036            The system supports the Typed Memory Objects option.
- 124037            This option was created to allow realtime applications to access different kinds of physical  
124038            memory, and allow processes in these applications to share portions of this memory.

### 124039    D.3.5   Configurable Limits

124040    In general, the configurable limits in the `<limits.h>` header defined in the Base Definitions  
124041    volume of POSIX.1-200x have been set to minimal values; many applications or  
124042    implementations may require larger values. No profile can cite lower values.

124043    {AIO\_LISTIO\_MAX}  
124044            The current minimum is likely to be inadequate for most applications. It is expected that  
124045            this value will be increased by profiles requiring support for list input and output  
124046            operations.

124047    {AIO\_MAX}  
124048            The current minimum is likely to be inadequate for most applications. It is expected that  
124049            this value will be increased by profiles requiring support for asynchronous input and  
124050            output operations.

124051    {AIO\_PRIO\_DELTA\_MAX}  
124052            The functionality associated with this limit is needed only by sophisticated applications. It  
124053            is not expected that this limit would need to be increased under a general-purpose profile.

124054    {ARG\_MAX}  
124055            The current minimum is likely to need to be increased for profiles, particularly as larger  
124056            amounts of information are passed through the environment. Many implementations are  
124057            believed to support larger values.

124058    {CHILD\_MAX}  
124059            The current minimum is suitable only for systems where a single user is not running  
124060            applications in parallel. It is significantly too low for any system also requiring windows,  
124061            and if `_POSIX_JOB_CONTROL` is specified, it should be raised.

124062    {CLOCKRES\_MIN}  
124063            It is expected that profiles will require a finer granularity clock, perhaps as fine as 1  $\mu$ s,  
124064            represented by a value of 1 000 for this limit.

124065    {DELAYTIMER\_MAX}  
124066            It is believed that most implementations will provide larger values.

124067    {LINK\_MAX}  
124068            For most applications and usage, the current minimum is adequate. Many implementations  
124069            have a much larger value, but this should not be used as a basis for raising the value unless  
124070            the applications to be used require it.

124071    {LOGIN\_NAME\_MAX}  
124072            This is not actually a limit, but an implementation parameter. No profile should impose a  
124073            requirement on this value.

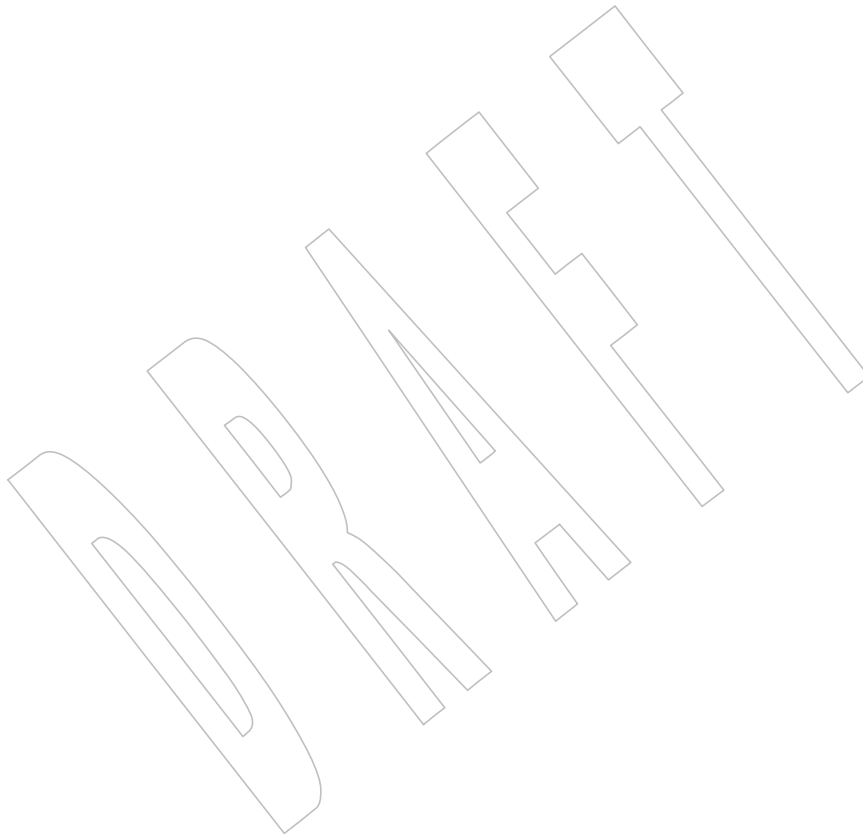
124074    {MAX\_CANON}  
124075            For most purposes, the current minimum is adequate. Unless high-speed burst serial  
124076            devices are used, it should be left as is.

- 124077 {MAX\_INPUT}  
124078 See {MAX\_CANON}.
- 124079 {MQ\_OPEN\_MAX}  
124080 The current minimum should be adequate for most profiles.
- 124081 {MQ\_PRIO\_MAX}  
124082 The current minimum corresponds to the required number of process scheduling priorities.  
124083 Many realtime practitioners believe that the number of message priority levels ought to be  
124084 the same as the number of execution scheduling priorities.
- 124085 {NAME\_MAX}  
124086 Many implementations now support larger values, and many applications and users  
124087 assume that larger names can be used. Many existing profiles also specify a larger value.  
124088 Specifying this value will reduce the number of conforming implementations, although this  
124089 might not be a significant consideration over time. Values greater than 255 should not be  
124090 required.
- 124091 {NGROUPS\_MAX}  
124092 The value selected will typically be 8 or larger.
- 124093 {OPEN\_MAX}  
124094 The historically common value for this has been 20. Many implementations support larger  
124095 values. If applications that use larger values are anticipated, an appropriate value should be  
124096 specified.
- 124097 {PAGESIZE}  
124098 This is not actually a limit, but an implementation parameter. No profile should impose a  
124099 requirement on this value.
- 124100 {PATH\_MAX}  
124101 Historically, the minimum has been either 1024 or indefinite, depending on the  
124102 implementation. Few applications actually require values larger than 256, but some users  
124103 may create file hierarchies that must be accessed with longer paths. This value should only  
124104 be changed if there is a clear requirement.
- 124105 {PIPE\_BUF}  
124106 The current minimum is adequate for most applications. Historically, it has been larger. If  
124107 applications that write single transactions larger than this are anticipated, it should be  
124108 increased. Applications that write lines of text larger than this probably do not need it  
124109 increased, as the text line is delimited by a <newline>.
- 124110 {POSIX\_VERSION}  
124111 This is actually not a limit, but a standard version stamp. Generally, a profile should specify  
124112 POSIX.1-200x by a name in the normative references section, not this value.
- 124113 {PTHREAD\_DESTRUCTOR\_ITERATIONS}  
124114 It is unlikely that applications will need larger values to avoid loss of memory resources.
- 124115 {PTHREAD\_KEYS\_MAX}  
124116 The current value should be adequate for most profiles.
- 124117 {PTHREAD\_STACK\_MIN}  
124118 This should not be treated as an actual limit, but as an implementation parameter. No  
124119 profile should impose a requirement on this value.
- 124120 {PTHREAD\_THREADS\_MAX}  
124121 It is believed that most implementations will provide larger values.

- 124122 {RTSIG\_MAX}  
 124123 The current limit was chosen so that the set of POSIX.1 signal numbers can fit within a  
 124124 32-bit field. It is recognized that most existing implementations define many more signals  
 124125 than are specified in POSIX.1 and, in fact, many implementations have already exceeded 32  
 124126 signals (including the “null signal”). Support of {\_POSIX\_RTSIG\_MAX} additional signals  
 124127 may push some implementations over the single 32-bit word line, but is unlikely to push  
 124128 any implementations that are already over that line beyond the 64 signal line.
- 124129 {SEM\_NSEMS\_MAX}  
 124130 The current value should be adequate for most profiles.
- 124131 {SEM\_VALUE\_MAX}  
 124132 The current value should be adequate for most profiles.
- 124133 {SSIZE\_MAX}  
 124134 This limit reflects fundamental hardware characteristics (the size of an integer), and should  
 124135 not be specified unless it is clearly required. Extreme care should be taken to assure that  
 124136 any value that might be specified does not unnecessarily eliminate implementations  
 124137 because of accidents of hardware design.
- 124138 {STREAM\_MAX}  
 124139 This limit is very closely related to {OPEN\_MAX}. It should never be larger than  
 124140 {OPEN\_MAX}, but could reasonably be smaller for application areas where most files are  
 124141 not accessed through *stdio*. Some implementations may limit {STREAM\_MAX} to 20 but  
 124142 allow {OPEN\_MAX} to be considerably larger. Such implementations should be allowed for  
 124143 if the applications permit.
- 124144 {TIMER\_MAX}  
 124145 The current limit should be adequate for most profiles, but it may need to be larger for  
 124146 applications with a large number of asynchronous operations.
- 124147 {TTY\_NAME\_MAX}  
 124148 This is not actually a limit, but an implementation parameter. No profile should impose a  
 124149 requirement on this value.
- 124150 {TZNAME\_MAX}  
 124151 The minimum has been historically adequate, but if longer timezone names are anticipated  
 124152 (particularly such values as UTC-1), this should be increased.

### 124153 D.3.6 Optional Behavior

- 124154 In POSIX.1-200x, there are no instances of the terms unspecified, undefined, implementation-  
 124155 defined, or with the verbs “may” or “need not”, that the standard developers anticipate or  
 124156 sanction as suitable for profile or test method citation. All of these are merely warnings to  
 124157 conforming applications to avoid certain areas that can vary from system to system, and even  
 124158 over time on the same system. In many cases, these terms are used explicitly to support  
 124159 extensions, but profiles should not anticipate and require such extensions; future versions of this  
 124160 standard may do so.



124161

**/** *Rationale (Informative)*

124162

**Part E:**

124163

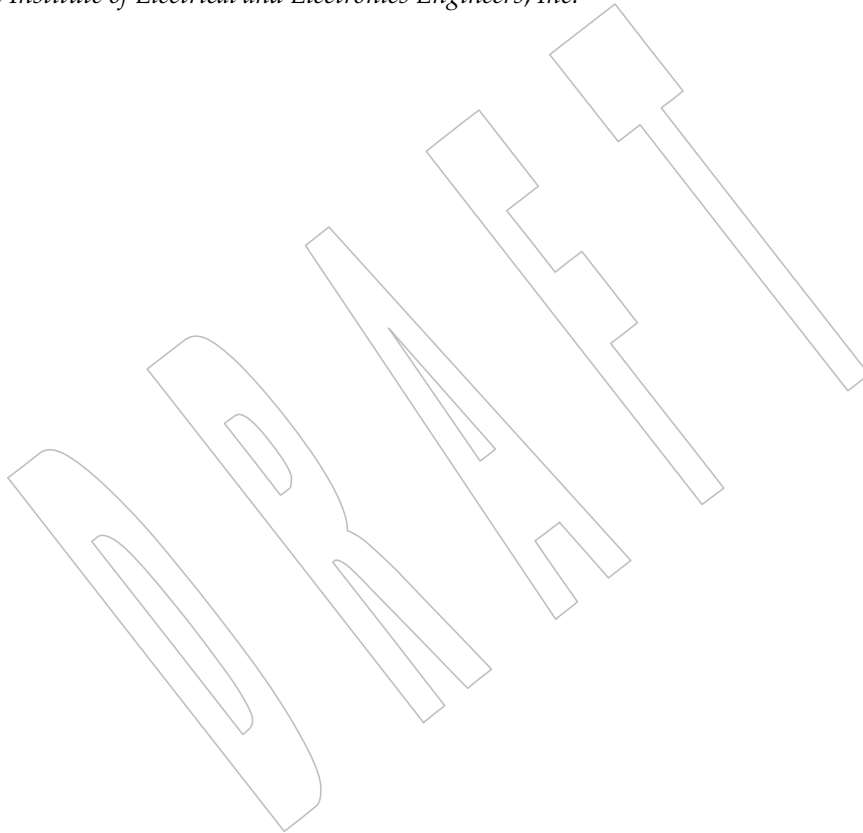
**Subprofiling Considerations**

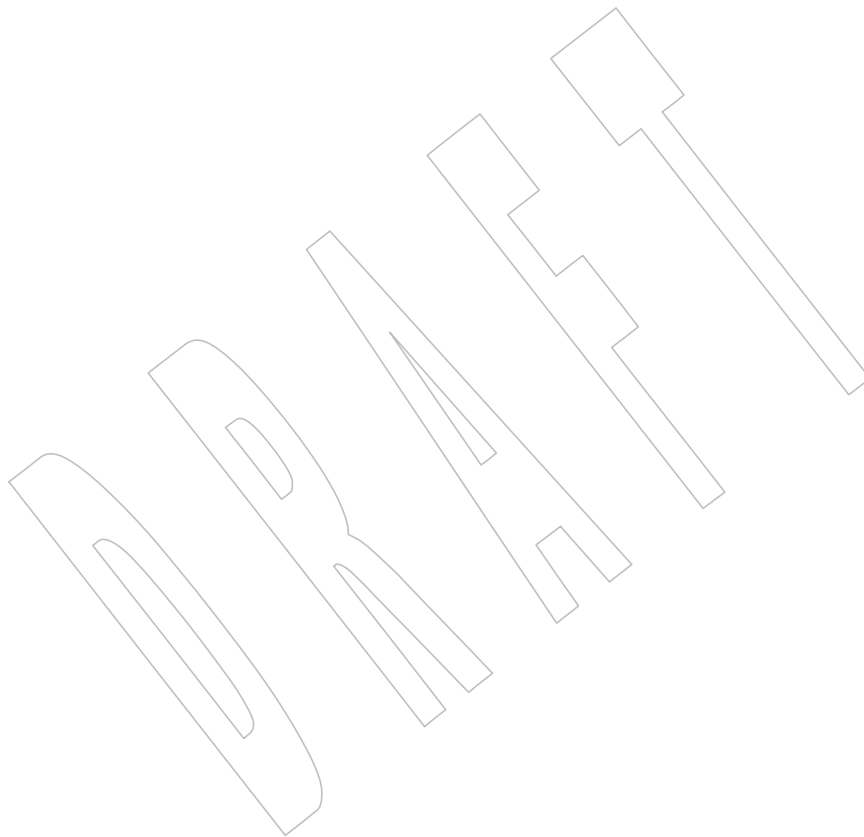
124164

*The Open Group*

124165

*The Institute of Electrical and Electronics Engineers, Inc.*





## Subprofiling Considerations (Informative)

124166

124167

124168

124169

124170

124171

This section contains further information to satisfy the requirement that the project scope enable subprofiling of POSIX.1-200x. The approach taken is to include a general requirement in normative text regarding subprofiling and to include an informative section (here) containing a proposed set of subprofiling options.

124172

### E.1 Subprofiling Option Groups

124173

124174

124175

124176

The following Option Groups<sup>10</sup> are defined to support profiling. Systems claiming support to POSIX.1-200x need not implement these options apart from the requirements stated in XBD Section 2.1.3 (on page 16). These Option Groups allow profiles to subset the System Interfaces volume of POSIX.1-200x by collecting sets of related functions.

124177

POSIX\_ASYNC\_HRONOUS\_IO: Asynchronous Input and Output Functions

124178

*aio\_cancel()*, *aio\_error()*, *aio\_fsync()*, *aio\_read()*, *aio\_return()*, *aio\_suspend()*, *aio\_write()*,

124179

*lio\_listio()*

124180

POSIX\_BARRIERS: Barriers

124181

*pthread\_barrier\_destroy()*, *pthread\_barrier\_init()*, *pthread\_barrier\_wait()*, *pthread\_barrierattr()*

124182

POSIX\_C\_LANG\_JUMP: Jump Functions

124183

*longjmp()*, *setjmp()*

124184

POSIX\_C\_LANG\_MATH: Maths Library

124185

*acos()*, *acosf()*, *acosh()*, *acoshf()*, *acoshl()*, *acosl()*, *asin()*, *asinf()*, *asinh()*, *asinhf()*, *asinhl()*,

124186

*asinl()*, *atan()*, *atan2()*, *atan2f()*, *atan2l()*, *atanf()*, *atanh()*, *atanhf()*, *atanhl()*, *atanl()*, *cabs()*,

124187

*cabsf()*, *cabsl()*, *cacos()*, *cacosf()*, *cacosh()*, *cacoshf()*, *cacoshl()*, *cacosl()*, *carg()*, *cargf()*, *cargl()*,

124188

*casin()*, *casinf()*, *casinh()*, *casinhf()*, *casinhl()*, *casinl()*, *catan()*, *catanf()*, *catanh()*, *catanhf()*,

124189

*catanhl()*, *catanl()*, *cbrt()*, *cbrtf()*, *cbrtl()*, *ccos()*, *ccosf()*, *ccosh()*, *ccoshf()*, *ccoshl()*, *ccosl()*,

124190

*ceil()*, *ceilf()*, *ceilL()*, *cexp()*, *cexpf()*, *cexpl()*, *cimag()*, *cimagf()*, *cimagl()*, *clog()*, *clogf()*, *clogl()*,

124191

*conj()*, *conjf()*, *conjl()*, *copysign()*, *copysignf()*, *copysignl()*, *cos()*, *cosf()*, *cosh()*, *coshf()*,

124192

*coshl()*, *cosl()*, *cpow()*, *cpowf()*, *cpowl()*, *cproj()*, *cprojf()*, *cprojl()*, *creal()*, *crealf()*, *creall()*,

124193

*csin()*, *csinf()*, *csinh()*, *csinhf()*, *csinhl()*, *csinl()*, *csqrt()*, *csqrtf()*, *csqrtl()*, *ctan()*, *ctanf()*,

124194

*ctanh()*, *ctanhf()*, *ctanhl()*, *ctanl()*, *erf()*, *erfc()*, *erfcf()*, *erfcl()*, *erff()*, *erfl()*, *exp()*, *exp2()*,

124195

*exp2f()*, *exp2l()*, *expf()*, *expl()*, *expm1()*, *expm1f()*, *expm1l()*, *fabs()*, *fabsf()*, *fabsl()*, *fdim()*,

124196

*fdimf()*, *fdiml()*, *floor()*, *floorf()*, *floorl()*, *fma()*, *fmaf()*, *fmal()*, *fmax()*, *fmaxf()*, *fmaxl()*, *fmin()*,

124197

*fminf()*, *fminl()*, *fmod()*, *fmodf()*, *fmodl()*, *fpclassify()*, *frexp()*, *frexpf()*, *frexpl()*, *hypot()*,

124198

*hypotf()*, *hypotl()*, *ilogb()*, *ilogbf()*, *ilogbl()*, *isfinite()*, *isgreater()*, *isgreaterequal()*, *isinf()*,

124199

*isless()*, *islessequal()*, *islessgreater()*, *isnan()*, *isnormal()*, *isunordered()*, *ldexp()*, *ldexpf()*,

124200

*ldexpl()*, *lgamma()*, *lgammaf()*, *lgammal()*, *llrint()*, *llrintf()*, *llrintl()*, *llround()*, *llroundf()*,

124201

*llroundl()*, *log()*, *log10()*, *log10f()*, *log10l()*, *log1p()*, *log1pf()*, *log1pl()*, *log2()*, *log2f()*, *log2l()*,

124202

*logb()*, *logbf()*, *logbl()*, *logf()*, *logl()*, *lrint()*, *lrintf()*, *lrintl()*, *lround()*, *lroundf()*, *lroundl()*,

124203

*modf()*, *modff()*, *modfl()*, *nan()*, *nanf()*, *nanl()*, *nearbyint()*, *nearbyintf()*, *nearbyintl()*,

124204

*nextafter()*, *nextafterf()*, *nextafterl()*, *nexttoward()*, *nexttowardf()*, *nexttowardl()*, *pow()*, *powf()*,

124205

*powl()*, *remainder()*, *remainderf()*, *remainderl()*, *remquo()*, *remquoof()*, *remquol()*, *rint()*, *rintf()*,

124206

10. These are modelled on the Units of Functionality from IEEE Std 1003.13-1998.

|        |                                                                                                                                                                                                         |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 124207 | <i>rintl()</i> , <i>round()</i> , <i>roundf()</i> , <i>roundl()</i> , <i>scalbln()</i> , <i>scalblnf()</i> , <i>scalblnl()</i> , <i>scalbn()</i> , <i>scalbnf()</i> ,                                   |
| 124208 | <i>scalbnl()</i> , <i>signbit()</i> , <i>sin()</i> , <i>sinf()</i> , <i>sinh()</i> , <i>sinhf()</i> , <i>sinhl()</i> , <i>sinl()</i> , <i>sqrt()</i> , <i>sqrtf()</i> , <i>sqrtl()</i> , <i>tan()</i> , |
| 124209 | <i>tanf()</i> , <i>tanh()</i> , <i>tanhf()</i> , <i>tanhL()</i> , <i>tanl()</i> , <i>tgamma()</i> , <i>tgammaf()</i> , <i>tgammaL()</i> , <i>trunc()</i> , <i>truncf()</i> ,                            |
| 124210 | <i>truncl()</i>                                                                                                                                                                                         |
| 124211 | POSIX_C_LANG_SUPPORT: General ISO C Library                                                                                                                                                             |
| 124212 | <i>abs()</i> , <i>asctime()</i> , <i>atof()</i> , <i>atoi()</i> , <i>atol()</i> , <i>atoll()</i> , <i>bsearch()</i> , <i>calloc()</i> , <i>ctime()</i> , <i>difftime()</i> , <i>div()</i> ,             |
| 124213 | <i>feclearexcept()</i> , <i>fegetenv()</i> , <i>fegetexceptflag()</i> , <i>fegetround()</i> , <i>fehldexcept()</i> , <i>feraiseexcept()</i> ,                                                           |
| 124214 | <i>fesetenv()</i> , <i>fesetexceptflag()</i> , <i>fesetround()</i> , <i>fetestexcept()</i> , <i>feupdateenv()</i> , <i>free()</i> , <i>gmtime()</i> ,                                                   |
| 124215 | <i>imaxabs()</i> , <i>imaxdiv()</i> , <i>isalnum()</i> , <i>isalpha()</i> , <i>isblank()</i> , <i>iscntrl()</i> , <i>isdigit()</i> , <i>isgraph()</i> , <i>islower()</i> ,                              |
| 124216 | <i>isprint()</i> , <i>ispunct()</i> , <i>isspace()</i> , <i>isupper()</i> , <i>isxdigit()</i> , <i>labs()</i> , <i>ldiv()</i> , <i>llabs()</i> , <i>lldiv()</i> , <i>localeconv()</i> ,                 |
| 124217 | <i>localtime()</i> , <i>malloc()</i> , <i>memchr()</i> , <i>memcmp()</i> , <i>memcpy()</i> , <i>memmove()</i> , <i>memset()</i> , <i>mktime()</i> ,                                                     |
| 124218 | <i>qsort()</i> , <i>rand()</i> , <i>realloc()</i> , <i>setlocale()</i> , <i>snprintf()</i> , <i>sprintf()</i> , <i>srand()</i> , <i>sscanf()</i> , <i>strcat()</i> , <i>strchr()</i> ,                  |
| 124219 | <i>strcmp()</i> , <i>strcoll()</i> , <i>strcpy()</i> , <i>strcspn()</i> , <i>strerror()</i> , <i>strftime()</i> , <i>strlen()</i> , <i>strncat()</i> , <i>strncmp()</i> ,                               |
| 124220 | <i>strncpy()</i> , <i>strpbrk()</i> , <i>strrchr()</i> , <i>strspn()</i> , <i>strstr()</i> , <i>strtod()</i> , <i>strtof()</i> , <i>strtoimax()</i> , <i>strtok()</i> , <i>strtol()</i> ,               |
| 124221 | <i>strtold()</i> , <i>strtoll()</i> , <i>strtoul()</i> , <i>strtoull()</i> , <i>strtoumax()</i> , <i>strxfrm()</i> , <i>time()</i> , <i>tolower()</i> , <i>toupper()</i> ,                              |
| 124222 | <i>tzname</i> , <i>tzset()</i> , <i>va_arg()</i> , <i>va_copy()</i> , <i>va_end()</i> , <i>va_start()</i> , <i>vsprintf()</i> , <i>vsscanf()</i>                                                        |
| 124223 | POSIX_C_LANG_SUPPORT_R: Thread-Safe General ISO C Library                                                                                                                                               |
| 124224 | <i>asctime_r()</i> , <i>ctime_r()</i> , <i>gmtime_r()</i> , <i>localtime_r()</i> , <i>rand_r()</i> , <i>strerror_r()</i> , <i>strtok_r()</i>                                                            |
| 124225 | POSIX_C_LANG_WIDE_CHAR: Wide-Character ISO C Library                                                                                                                                                    |
| 124226 | <i>btowc()</i> , <i>iswalnum()</i> , <i>iswalpunct()</i> , <i>iswblank()</i> , <i>iswcntrl()</i> , <i>iswctype()</i> , <i>iswdigit()</i> , <i>iswgraph()</i> ,                                          |
| 124227 | <i>iswlower()</i> , <i>iswprint()</i> , <i>iswpunct()</i> , <i>iswspace()</i> , <i>iswupper()</i> , <i>iswxdigit()</i> , <i>mblen()</i> , <i>mbrlen()</i> ,                                             |
| 124228 | <i>mbrtowc()</i> , <i>mbsinit()</i> , <i>mbsrtowcs()</i> , <i>mbstowcs()</i> , <i>mbtowc()</i> , <i>swprintf()</i> , <i>swscanf()</i> , <i>towctrans()</i> ,                                            |
| 124229 | <i>towlower()</i> , <i>towupper()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>wcrtomb()</i> , <i>wcscat()</i> , <i>wcschr()</i> , <i>wcscmp()</i> ,                                                 |
| 124230 | <i>wcscoll()</i> , <i>wcscpy()</i> , <i>wcscspn()</i> , <i>wcsftime()</i> , <i>wcslen()</i> , <i>wcsncat()</i> , <i>wcsncmp()</i> , <i>wcsncpy()</i> ,                                                  |
| 124231 | <i>wcspbrk()</i> , <i>wcsrchr()</i> , <i>wcstombs()</i> , <i>wcsspn()</i> , <i>wcsstr()</i> , <i>wcstod()</i> , <i>wcstof()</i> , <i>wcstoimax()</i> ,                                                  |
| 124232 | <i>wcstok()</i> , <i>wcstol()</i> , <i>wcstold()</i> , <i>wcstoll()</i> , <i>wcstombs()</i> , <i>wcstoul()</i> , <i>wcstoull()</i> , <i>wcstoumax()</i> ,                                               |
| 124233 | <i>wcsxfrm()</i> , <i>wctob()</i> , <i>wctomb()</i> , <i>wctrans()</i> , <i>wctype()</i> , <i>wmemchr()</i> , <i>wmemcmp()</i> , <i>wmemcpy()</i> ,                                                     |
| 124234 | <i>wmemmove()</i> , <i>wmemset()</i>                                                                                                                                                                    |
| 124235 | POSIX_C_LANG_WIDE_CHAR_EXT: Extended Wide-Character ISO C Library                                                                                                                                       |
| 124236 | <i>mbsnrtowcs()</i> , <i>wcpcpy()</i> , <i>wcpncpy()</i> , <i>wcscasecmp()</i> , <i>wcsdup()</i> , <i>wcsncasecmp()</i> , <i>wcsnlen()</i> ,                                                            |
| 124237 | <i>wcsnrtombs()</i>                                                                                                                                                                                     |
| 124238 | POSIX_C_LIB_EXT: General C Library Extension                                                                                                                                                            |
| 124239 | <i>fnmatch()</i> , <i>getopt()</i> , <i>getsubopt()</i> , <i>optarg</i> , <i>opterr</i> , <i>optind</i> , <i>optopt</i> , <i>stpcpy()</i> , <i>stpncpy()</i> , <i>strcasecmp()</i> ,                    |
| 124240 | <i>strdup()</i> , <i>strfmon()</i> , <i>strncasecmp()</i> , <i>strndup()</i> , <i>strnlen()</i>                                                                                                         |
| 124241 | POSIX_CLOCK_SELECTION: Clock Selection                                                                                                                                                                  |
| 124242 | <i>clock_nanosleep()</i> , <i>pthread_condattr_getclock()</i> , <i>pthread_condattr_setclock()</i>                                                                                                      |
| 124243 | POSIX_DEVICE_IO: Device Input and Output                                                                                                                                                                |
| 124244 | <i>FD_CLR()</i> , <i>FD_ISSET()</i> , <i>FD_SET()</i> , <i>FD_ZERO()</i> , <i>clearerr()</i> , <i>close()</i> , <i>fclose()</i> , <i>fdopen()</i> , <i>feof()</i> ,                                     |
| 124245 | <i>ferror()</i> , <i>fflush()</i> , <i>fgetc()</i> , <i>fgets()</i> , <i>fileno()</i> , <i>fopen()</i> , <i>fprintf()</i> , <i>fputc()</i> , <i>fputs()</i> , <i>fread()</i> , <i>freopen()</i> ,       |
| 124246 | <i>fscanf()</i> , <i>fwrite()</i> , <i>getc()</i> , <i>getchar()</i> , <i>gets()</i> , <i>open()</i> , <i>perror()</i> , <i>poll()</i> , <i>printf()</i> , <i>pread()</i> , <i>pselect()</i> ,          |
| 124247 | <i>putc()</i> , <i>putchar()</i> , <i>puts()</i> , <i>pwrite()</i> , <i>read()</i> , <i>scanf()</i> , <i>select()</i> , <i>setbuf()</i> , <i>setvbuf()</i> , <i>stderr</i> , <i>stdin</i> ,             |
| 124248 | <i>stdout</i> , <i>ungetc()</i> , <i>vfprintf()</i> , <i>vscanf()</i> , <i>vprintf()</i> , <i>vscanf()</i> , <i>write()</i>                                                                             |
| 124249 | POSIX_DEVICE_IO_EXT: Extended Device Input and Output                                                                                                                                                   |
| 124250 | <i>dprintf()</i> , <i>fmemopen()</i> , <i>open_memstream()</i>                                                                                                                                          |
| 124251 | POSIX_DEVICE_SPECIFIC: General Terminal                                                                                                                                                                 |
| 124252 | <i>cfgetispeed()</i> , <i>cfgetospeed()</i> , <i>cfsetispeed()</i> , <i>cfsetospeed()</i> , <i>ctermid()</i> , <i>isatty()</i> , <i>tcdrain()</i> , <i>tcflow()</i> ,                                   |
| 124253 | <i>tcflush()</i> , <i>tcgetattr()</i> , <i>tcsendbreak()</i> , <i>tcsetattr()</i> , <i>ttyname()</i>                                                                                                    |



|        |                                                                                                                  |
|--------|------------------------------------------------------------------------------------------------------------------|
| 124254 | POSIX_DEVICE_SPECIFIC_R: Thread-Safe General Terminal                                                            |
| 124255 | <i>ttyname_r()</i>                                                                                               |
| 124256 | POSIX_DYNAMIC_LINKING: Dynamic Linking                                                                           |
| 124257 | <i>dlclose(), dlerror(), dlopen(), dlsym()</i>                                                                   |
| 124258 | POSIX_FD_MGMT: File Descriptor Management                                                                        |
| 124259 | <i>dup(), dup2(), fcntl(), fgetpos(), fseek(), fseeko(), fsetpos(), ftell(), ftello(), ftruncate(), lseek(),</i> |
| 124260 | <i>rewind()</i>                                                                                                  |
| 124261 | POSIX_FIFO: FIFO                                                                                                 |
| 124262 | <i>mkfifo()</i>                                                                                                  |
| 124263 | POSIX_FIFO_FD: FIFO File Descriptor Routines                                                                     |
| 124264 | <i>mkfifoat(), mknodat()</i>                                                                                     |
| 124265 | POSIX_FILE_ATTRIBUTES: File Attributes                                                                           |
| 124266 | <i>chmod(), chown(), fchmod(), fchown(), umask()</i>                                                             |
| 124267 | POSIX_FILE_ATTRIBUTES_FD: File Attributes File Descriptor Routines                                               |
| 124268 | <i>fchmodat(), fchownat()</i>                                                                                    |
| 124269 | POSIX_FILE_LOCKING: Thread-Safe Stdio Locking                                                                    |
| 124270 | <i>flockfile(), frylockfile(), funlockfile(), getc_unlocked(), getchar_unlocked(), putc_unlocked(),</i>          |
| 124271 | <i>putchar_unlocked()</i>                                                                                        |
| 124272 | POSIX_FILE_SYSTEM: File System                                                                                   |
| 124273 | <i>access(), chdir(), closedir(), creat(), fchdir(), fpathconf(), fstat(), fstatvfs(), getcwd(), link(),</i>     |
| 124274 | <i>mkdir(), mkstemp(), opendir(), pathconf(), readdir(), remove(), rename(), rewinddir(), rmdir(),</i>           |
| 124275 | <i>stat(), statvfs(), tmpfile(), tmpnam(), truncate(), unlink(), utime()</i>                                     |
| 124276 | POSIX_FILE_SYSTEM_EXT: File System Extensions                                                                    |
| 124277 | <i>alphasort(), dirfd(), getdelim(), getline(), mkdtemp(), scandir()</i>                                         |
| 124278 | POSIX_FILE_SYSTEM_FD: File System File Descriptor Routines                                                       |
| 124279 | <i>faccessat(), fdopendir(), fstatat(), futimesat(), linkat(), mkdirat(), openat(), renameat(),</i>              |
| 124280 | <i>unlinkat()</i>                                                                                                |
| 124281 | POSIX_FILE_SYSTEM_GLOB: File System Glob Expansion                                                               |
| 124282 | <i>glob(), globfree()</i>                                                                                        |
| 124283 | POSIX_FILE_SYSTEM_R: Thread-Safe File System                                                                     |
| 124284 | <i>readdir_r()</i>                                                                                               |
| 124285 | POSIX_I18N: Internationalization                                                                                 |
| 124286 | <i>catclose(), catgets(), catopen(), iconv(), iconv_close(), iconv_open(), nl_langinfo()</i>                     |
| 124287 | POSIX_JOB_CONTROL: Job Control                                                                                   |
| 124288 | <i>setpgid(), tcgetpgrp(), tcsetpgrp(), tcgetsid()</i>                                                           |
| 124289 | POSIX_MAPPED_FILES: Memory Mapped Files                                                                          |
| 124290 | <i>mmap(), munmap()</i>                                                                                          |
| 124291 | POSIX_MEMORY_PROTECTION: Memory Protection                                                                       |
| 124292 | <i>mprotect()</i>                                                                                                |
| 124293 | POSIX_MULTI_CONCURRENT_LOCALES: Multiple Concurrent Locales                                                      |
| 124294 | <i>duplocale(), freelocale(), isalnum_l(), isalpha_l(), isblank_l(), iscntrl_l(), isdigit_l(), isgraph_l(),</i>  |
| 124295 | <i>islower_l(), isprint_l(), ispunct_l(), isspace_l(), isupper_l(), iswalnum_l(), iswalpha_l(),</i>              |
| 124296 | <i>iswblank_l(), iswcntrl_l(), iswctype_l(), iswdigit_l(), iswgraph_l(), iswlower_l(), iswprint_l(),</i>         |
| 124297 | <i>iswpunct_l(), iswspace_l(), iswupper_l(), iswxdigit_l(), isxdigit_l(), newlocale(), strcasecmp_l(),</i>       |

124298 *strcoll\_l()*, *strfmon\_l()*, *strncasecmp\_l()*, *strxfrm\_l()*, *tolower\_l()*, *toupper\_l()*, *towctrans\_l()*,  
 124299 *towlower()*, *towupper()*, *uselocale()*, *wcscasecmp\_l()*, *wcscoll\_l()*, *wcscncasecmp\_l()*, *wcsxfrm\_l()*,  
 124300 *wctrans\_l()*, *wctype\_l()*

124301 POSIX\_MULTI\_PROCESS: Multiple Processes  
 124302 *\_Exit()*, *\_exit()*, *assert()*, *atexit()*, *clock()*, *execl()*, *execle()*, *execlp()*, *execv()*, *execve()*, *execvp()*,  
 124303 *exit()*, *fork()*, *getpgrp()*, *getpgid()*, *getpid()*, *getppid()*, *getsid()*, *setsid()*, *sleep()*, *times()*, *wait()*,  
 124304 *waitid()*, *waitpid()*

124305 POSIX\_MULTI\_PROCESS\_FD: Multiple Processes File Descriptor Routines  
 124306 *fexecve()*

124307 POSIX\_NETWORKING: Networking  
 124308 *accept()*, *bind()*, *connect()*, *endhostent()*, *endnetent()*, *endprotoent()*, *endservent()*,  
 124309 *freeaddrinfo()*, *gai\_strerror()*, *getaddrinfo()*, *gethostent()*, *gethostname()*, *getnameinfo()*,  
 124310 *getnetbyaddr()*, *getnetbyname()*, *getnetent()*, *getpeername()*, *getprotobyname()*,  
 124311 *getprotobynumber()*, *getprotoent()*, *getserbyname()*, *getservbyname()*, *getservbyport()*, *getservent()*,  
 124312 *getsockname()*, *getsockopt()*, *htonl()*, *htons()*, *if\_freenameindex()*, *if\_indextoname()*,  
 124313 *if\_nameindex()*, *if\_nametoindex()*, *inet\_addr()*, *inet\_ntoa()*, *inet\_ntop()*, *inet\_pton()*, *listen()*,  
 124314 *ntohl()*, *ntohs()*, *recv()*, *recvfrom()*, *recvmsg()*, *send()*, *sendmsg()*, *sendto()*, *sethostent()*,  
 124315 *setnetent()*, *setprotoent()*, *setservent()*, *setsockopt()*, *shutdown()*, *socket()*, *socketatmark()*,  
 124316 *socketpair()*

124317 POSIX\_PIPE: Pipe  
 124318 *pipe()*

124319 POSIX\_ROBUST\_MUTEXES: Robust Mutexes  
 124320 *pthread\_mutex\_consistent()*, *pthread\_mutexattr\_getrobust()*, *pthread\_mutexattr\_setrobust()*

124321 POSIX\_REALTIME\_SIGNALS: Realtime Signals  
 124322 *sigqueue()*, *sigtimedwait()*, *sigwaitinfo()*

124323 POSIX\_REGEX: Regular Expressions  
 124324 *regcomp()*, *regerror()*, *regexec()*, *regfree()*

124325 POSIX\_RW\_LOCKS: Reader Writer Locks  
 124326 *pthread\_rwlock\_destroy()*, *pthread\_rwlock\_init()*, *pthread\_rwlock\_rdlock()*,  
 124327 *pthread\_rwlock\_timedrdlock()*, *pthread\_rwlock\_timedwrlock()*, *pthread\_rwlock\_tryrdlock()*,  
 124328 *pthread\_rwlock\_trywrlock()*, *pthread\_rwlock\_unlock()*, *pthread\_rwlock\_wrlock()*,  
 124329 *pthread\_rwlockattr\_destroy()*, *pthread\_rwlockattr\_init()*, *pthread\_rwlockattr\_getpshared()*,  
 124330 *pthread\_rwlockattr\_setpshared()*

124331 POSIX\_SEMAPHORES: Semaphores  
 124332 *sem\_close()*, *sem\_destroy()*, *sem\_getvalue()*, *sem\_init()*, *sem\_open()*, *sem\_post()*,  
 124333 *sem\_timedwait()*, *sem\_trywait()*, *sem\_unlink()*, *sem\_wait()*

124334 POSIX\_SHELL\_FUNC: Shell and Utilities  
 124335 *pclose()*, *popen()*, *system()*, *wordexp()*, *wordfree()*

124336 POSIX\_SIGNAL\_JUMP: Signal Jump Functions  
 124337 *siglongjmp()*, *sigsetjmp()*

124338 POSIX\_SIGNALS: Signals  
 124339 *abort()*, *alarm()*, *kill()*, *pause()*, *raise()*, *sigaction()*, *sigaddset()*, *sigdelset()*, *sigemptyset()*,  
 124340 *sigfillset()*, *sigismember()*, *signal()*, *sigpending()*, *sigprocmask()*, *sigsuspend()*, *sigwait()*

124341 POSIX\_SIGNALS\_EXT: Extended Signals  
 124342 *psignal()*, *psiginfo()*, *strsignal()*

|        |                                                                                                                                                                            |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 124343 | POSIX_SINGLE_PROCESS: Single Process                                                                                                                                       |
| 124344 | <i>confstr()</i> , <i>environ</i> , <i>errno</i> , <i>getenv()</i> , <i>setenv()</i> , <i>sysconf()</i> , <i>uname()</i> , <i>unsetenv()</i>                               |
| 124345 | POSIX_SPIN_LOCKS: Spin Locks                                                                                                                                               |
| 124346 | <i>pthread_spin_destroy()</i> , <i>pthread_spin_init()</i> , <i>pthread_spin_lock()</i> , <i>pthread_spin_trylock()</i> ,                                                  |
| 124347 | <i>pthread_spin_unlock()</i>                                                                                                                                               |
| 124348 | POSIX_SYMBOLIC_LINKS: Symbolic Links                                                                                                                                       |
| 124349 | <i>lchown()</i> , <sup>11</sup> <i>lstat()</i> , <i>readlink()</i> , <i>symlink()</i>                                                                                      |
| 124350 | POSIX_SYMBOLIC_LINKS_FD: Symbolic Links File Descriptor Routines                                                                                                           |
| 124351 | <i>readlinkat()</i> , <i>symlinkat()</i>                                                                                                                                   |
| 124352 | POSIX_SYSTEM_DATABASE: System Database                                                                                                                                     |
| 124353 | <i>getgrgid()</i> , <i>getgrnam()</i> , <i>getpwnam()</i> , <i>getpwuid()</i>                                                                                              |
| 124354 | POSIX_SYSTEM_DATABASE_R: Thread-Safe System Database                                                                                                                       |
| 124355 | <i>getgrgid_r()</i> , <i>getgrnam_r()</i> , <i>getpwnam_r()</i> , <i>getpwuid_r()</i>                                                                                      |
| 124356 | POSIX_THREADS_BASE: Base Threads                                                                                                                                           |
| 124357 | <i>pthread_atfork()</i> , <i>pthread_attr_destroy()</i> , <i>pthread_attr_getdetachstate()</i> ,                                                                           |
| 124358 | <i>pthread_attr_getschedparam()</i> , <i>pthread_attr_init()</i> , <i>pthread_attr_setdetachstate()</i> ,                                                                  |
| 124359 | <i>pthread_attr_setschedparam()</i> , <i>pthread_cancel()</i> , <i>pthread_cleanup_pop()</i> , <i>pthread_cleanup_push()</i> ,                                             |
| 124360 | <i>pthread_cond_broadcast()</i> , <i>pthread_cond_destroy()</i> , <i>pthread_cond_init()</i> , <i>pthread_cond_signal()</i> ,                                              |
| 124361 | <i>pthread_cond_timedwait()</i> , <i>pthread_cond_wait()</i> , <i>pthread_condattr_destroy()</i> ,                                                                         |
| 124362 | <i>pthread_condattr_init()</i> , <i>pthread_create()</i> , <i>pthread_detach()</i> , <i>pthread_equal()</i> , <i>pthread_exit()</i> ,                                      |
| 124363 | <i>pthread_getspecific()</i> , <i>pthread_join()</i> , <i>pthread_key_create()</i> , <i>pthread_key_delete()</i> , <i>pthread_kill()</i> ,                                 |
| 124364 | <i>pthread_mutex_destroy()</i> , <i>pthread_mutex_init()</i> , <i>pthread_mutex_lock()</i> ,                                                                               |
| 124365 | <i>pthread_mutex_timedlock()</i> , <i>pthread_mutex_trylock()</i> , <i>pthread_mutex_unlock()</i> ,                                                                        |
| 124366 | <i>pthread_mutexattr_destroy()</i> , <i>pthread_mutexattr_init()</i> , <i>pthread_once()</i> , <i>pthread_self()</i> ,                                                     |
| 124367 | <i>pthread_setcancelstate()</i> , <i>pthread_setcanceltype()</i> , <i>pthread_setspecific()</i> , <i>pthread_sigmask()</i> ,                                               |
| 124368 | <i>pthread_testcancel()</i>                                                                                                                                                |
| 124369 | POSIX_THREADS_EXT: Extended Threads                                                                                                                                        |
| 124370 | <i>pthread_attr_getguardsize()</i> , <i>pthread_attr_setguardsize()</i> , <i>pthread_mutexattr_gettype()</i> ,                                                             |
| 124371 | <i>pthread_mutexattr_settype()</i>                                                                                                                                         |
| 124372 | POSIX_TIMERS: Timers                                                                                                                                                       |
| 124373 | <i>clock_getres()</i> , <i>clock_gettime()</i> , <i>clock_settime()</i> , <i>nanosleep()</i> , <i>timer_create()</i> , <i>timer_delete()</i> ,                             |
| 124374 | <i>timer_getoverrun()</i> , <i>timer_gettime()</i> , <i>timer_settime()</i>                                                                                                |
| 124375 | POSIX_USER_GROUPS: User and Group                                                                                                                                          |
| 124376 | <i>getegid()</i> , <i>geteuid()</i> , <i>getgid()</i> , <i>getgroups()</i> , <i>getlogin()</i> , <i>getuid()</i> , <i>setegid()</i> , <i>seteuid()</i> , <i>setgid()</i> , |
| 124377 | <i>setuid()</i>                                                                                                                                                            |
| 124378 | POSIX_USER_GROUPS_R: Thread-Safe User and Group                                                                                                                            |
| 124379 | <i>getlogin_r()</i>                                                                                                                                                        |
| 124380 | POSIX_WIDE_CHAR_DEVICE_IO: Device Input and Output                                                                                                                         |
| 124381 | <i>fgetwc()</i> , <i>fgetws()</i> , <i>fputwc()</i> , <i>fputws()</i> , <i>fwide()</i> , <i>fwprintf()</i> , <i>fwscanf()</i> , <i>getwc()</i> , <i>getwchar()</i> ,       |
| 124382 | <i>putwc()</i> , <i>putwchar()</i> , <i>ungetwc()</i> , <i>vfwprintf()</i> , <i>vfwscanf()</i> , <i>vwprintf()</i> , <i>vwscanf()</i> , <i>wprintf()</i> ,                 |
| 124383 | <i>wscanf()</i>                                                                                                                                                            |
| 124384 | XSI_C_LANG_SUPPORT: XSI General C Library                                                                                                                                  |
| 124385 | <i>_tolower()</i> , <i>_toupper()</i> , <i>a64l()</i> , <i>daylight()</i> , <i>drand48()</i> , <i>erand48()</i> , <i>ffs()</i> , <i>getdate()</i> , <i>hcreate()</i> ,     |
| 124386 | <i>hdestroy()</i> , <i>hsearch()</i> , <i>initstate()</i> , <i>insque()</i> , <i>isascii()</i> , <i>jrand48()</i> , <i>l64a()</i> , <i>lcong48()</i> , <i>lfind()</i> ,    |
| 124387 | <i>lrand48()</i> , <i>lsearch()</i> , <i>memccpy()</i> , <i>mrnd48()</i> , <i>nrnd48()</i> , <i>random()</i> , <i>remque()</i> , <i>seed48()</i> ,                         |

11. The *lchown()* function also depends on POSIX\_FILE\_ATTRIBUTES.

|        |                                                                                                                                                                                   |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 124389 | <i>setstate()</i> , <i>siggam</i> , <i>srand48()</i> , <i>srandom()</i> , <i>strptime()</i> , <i>swab()</i> , <i>tdelete()</i> , <i>tfind()</i> , <i>timezone()</i> ,             |
| 124390 | <i>toascii()</i> , <i>tsearch()</i> , <i>twalk()</i>                                                                                                                              |
| 124391 | XSI_DBM: XSI Database Management                                                                                                                                                  |
| 124392 | <i>dbm_clearerr()</i> , <i>dbm_close()</i> , <i>dbm_delete()</i> , <i>dbm_error()</i> , <i>dbm_fetch()</i> , <i>dbm_firstkey()</i> ,                                              |
| 124393 | <i>dbm_nextkey()</i> , <i>dbm_open()</i> , <i>dbm_store()</i>                                                                                                                     |
| 124394 | XSI_DEVICE_IO: XSI Device Input and Output                                                                                                                                        |
| 124395 | <i>fntmsg()</i> , <i>readv()</i> , <i>writew()</i>                                                                                                                                |
| 124396 | XSI_DEVICE_SPECIFIC: XSI General Terminal                                                                                                                                         |
| 124397 | <i>grantpt()</i> , <i>posix_openpt()</i> , <i>ptsname()</i> , <i>unlockpt()</i>                                                                                                   |
| 124398 | XSI_FILE_SYSTEM: XSI File System                                                                                                                                                  |
| 124399 | <i>basename()</i> , <i>dirname()</i> , <i>ftw()</i> , <i>lockf()</i> , <i>mknod()</i> , <i>nftw()</i> , <i>realpath()</i> , <i>seekdir()</i> , <i>sync()</i> , <i>telldir()</i> , |
| 124400 | <i>tempnam()</i>                                                                                                                                                                  |
| 124401 | XSI_IPC: XSI Interprocess Communication                                                                                                                                           |
| 124402 | <i>ftok()</i> , <i>msgctl()</i> , <i>msgget()</i> , <i>msgrcv()</i> , <i>msgsnd()</i> , <i>semctl()</i> , <i>semget()</i> , <i>semop()</i> , <i>shmat()</i> , <i>shmctl()</i> ,   |
| 124403 | <i>shmdt()</i> , <i>shmget()</i>                                                                                                                                                  |
| 124404 | XSI_JUMP: XSI Jump Functions                                                                                                                                                      |
| 124405 | <i>_longjmp()</i> , <i>_setjmp()</i>                                                                                                                                              |
| 124406 | XSI_MATH: XSI Maths Library                                                                                                                                                       |
| 124407 | <i>j0()</i> , <i>j1()</i> , <i>jn()</i> , <i>y0()</i> , <i>y1()</i> , <i>yn()</i>                                                                                                 |
| 124408 | XSI_MULTI_PROCESS: XSI Multiple Process                                                                                                                                           |
| 124409 | <i>getpriority()</i> , <i>getrlimit()</i> , <i>getrusage()</i> , <i>nice()</i> , <i>setpgrp()</i> , <i>setpriority()</i> , <i>setrlimit()</i> , <i>ulimit()</i> ,                 |
| 124410 | XSI_SIGNALS: XSI Signal                                                                                                                                                           |
| 124411 | <i>killpg()</i> , <i>sigaltstack()</i> , <i>sighold()</i> , <i>sigignore()</i> , <i>siginterrupt()</i> , <i>sigpause()</i> , <i>sigrelse()</i> , <i>sigset()</i> ,                |
| 124412 | XSI_SINGLE_PROCESS: XSI Single Process                                                                                                                                            |
| 124413 | <i>gethostid()</i> , <i>gettimeofday()</i> , <i>putenv()</i>                                                                                                                      |
| 124414 | XSI_SYSTEM_DATABASE: XSI System Database                                                                                                                                          |
| 124415 | <i>endpwent()</i> , <i>getpwent()</i> , <i>setpwent()</i>                                                                                                                         |
| 124416 | XSI_SYSTEM_LOGGING: XSI System Logging                                                                                                                                            |
| 124417 | <i>closelog()</i> , <i>openlog()</i> , <i>setlogmask()</i> , <i>syslog()</i>                                                                                                      |
| 124418 | XSI_THREADS_EXT: XSI Threads Extensions                                                                                                                                           |
| 124419 | <i>pthread_attr_getstack()</i> , <i>pthread_attr_setstack()</i> , <i>pthread_getconcurrency()</i> ,                                                                               |
| 124420 | <i>pthread_setconcurrency()</i>                                                                                                                                                   |
| 124421 | XSI_TIMERS: XSI Timers                                                                                                                                                            |
| 124422 | <i>getitimer()</i> , <i>setitimer()</i>                                                                                                                                           |
| 124423 | XSI_USER_GROUPS: XSI User and Group                                                                                                                                               |
| 124424 | <i>endgrent()</i> , <i>endutxent()</i> , <i>getgrent()</i> , <i>getutxent()</i> , <i>getutxid()</i> , <i>getutxline()</i> , <i>pututxline()</i> ,                                 |
| 124425 | <i>setgrent()</i> , <i>setregid()</i> , <i>setreuid()</i> , <i>setutxent()</i>                                                                                                    |
| 124426 | XSI_WIDE_CHAR: XSI Wide-Character Library                                                                                                                                         |
| 124427 | <i>wcswidth()</i> , <i>wcwidth()</i>                                                                                                                                              |

# Index

|                         |           |                        |           |
|-------------------------|-----------|------------------------|-----------|
| (time) resolution ..... | 77        | <netinet/tcp.h> .....  | 291       |
| / .....                 | 183       | <newline> .....        | 66        |
| /dev .....              | 183       | <nl_types.h> .....     | 292       |
| /dev/console .....      | 183       | <poll.h> .....         | 293       |
| /dev/null .....         | 183       | <pthread.h> .....      | 295, 3474 |
| /dev/tty .....          | 183, 3332 | <pwd.h> .....          | 301       |
| /etc/passwd .....       | 3346      | <regex.h> .....        | 303       |
| /tmp .....              | 183       | <sched.h> .....        | 305       |
| <aio.h> .....           | 206       | <search.h> .....       | 307       |
| <alert> .....           | 34        | <semaphore.h> .....    | 309       |
| <arpa/inet.h> .....     | 208       | <setjmp.h> .....       | 311       |
| <assert.h> .....        | 209       | <signal.h> .....       | 312       |
| <backspace> .....       | 38        | <space> .....          | 82        |
| <blank> .....           | 43        | <spawn.h> .....        | 320       |
| <carriage-return> ..... | 45        | <stdarg.h> .....       | 322       |
| <complex.h> .....       | 210       | <stdbool.h> .....      | 324       |
| <control>-V .....       | 2585      | <stddef.h> .....       | 325       |
| <control>-W .....       | 2586      | <stdint.h> .....       | 327       |
| <cpio.h> .....          | 213       | <stdio.h> .....        | 334       |
| <ctype.h> .....         | 215       | <stdlib.h> .....       | 338       |
| <dirent.h> .....        | 217       | <string.h> .....       | 342       |
| <dlfcn.h> .....         | 219       | <strings.h> .....      | 344       |
| <errno.h> .....         | 220       | <stropts.h> .....      | 345       |
| <fcntl.h> .....         | 224       | <sys/dir.h> .....      | 217       |
| <fenv.h> .....          | 228       | <sys/ipc.h> .....      | 350       |
| <float.h> .....         | 232       | <sys/mman.h> .....     | 352       |
| <fmtmsg.h> .....        | 236       | <sys/msg.h> .....      | 355       |
| <fnmatch.h> .....       | 238       | <sys/resource.h> ..... | 357       |
| <form-feed> .....       | 59        | <sys/select.h> .....   | 359       |
| <ftw.h> .....           | 239       | <sys/sem.h> .....      | 361       |
| <glob.h> .....          | 241       | <sys/shm.h> .....      | 363       |
| <grp.h> .....           | 243       | <sys/socket.h> .....   | 365       |
| <iconv.h> .....         | 245       | <sys/stat.h> .....     | 370       |
| <inttypes.h> .....      | 246       | <sys/statvfs.h> .....  | 375       |
| <iso646.h> .....        | 248       | <sys/time.h> .....     | 377       |
| <langinfo.h> .....      | 249       | <sys/times.h> .....    | 379       |
| <libgen.h> .....        | 252       | <sys/types.h> .....    | 380       |
| <limits.h> .....        | 253       | <sys/uio.h> .....      | 384       |
| <locale.h> .....        | 267       | <sys/un.h> .....       | 385       |
| <math.h> .....          | 270       | <sys/utsname.h> .....  | 386       |
| <monetary.h> .....      | 277       | <sys/wait.h> .....     | 387       |
| <mqueue.h> .....        | 278       | <syslog.h> .....       | 389       |
| <ndbm.h> .....          | 280       | <tab> .....            | 87        |
| <net/if.h> .....        | 282       | <tar.h> .....          | 391       |
| <netdb.h> .....         | 283       | <termios.h> .....      | 393       |
| <netinet/in.h> .....    | 287       | <tgmath.h> .....       | 399       |
|                         |           | <time.h> .....         | 403       |

|                                         |                 |                                  |                 |
|-----------------------------------------|-----------------|----------------------------------|-----------------|
| <trace.h> .....                         | 407             | _IOLBF .....                     | 334, 832, 1851  |
| <ulimit.h> .....                        | 411             | _IONBF .....                     | 334, 1810, 1851 |
| <unistd.h> .....                        | 412             | _LDLFLAGS .....                  | 2432            |
| <utime.h> .....                         | 432             | _LIBS .....                      | 2432            |
| <utmpx.h> .....                         | 433             | _LINE .....                      | 575             |
| <vertical-tab> .....                    | 93              | _longjmp() .....                 | 528, 3590       |
| <wchar.h> .....                         | 435             | _LVL .....                       | 451             |
| <wctype.h> .....                        | 439             | _MAX .....                       | 450             |
| <wordexp.h> .....                       | 441             | _MIN .....                       | 253, 450        |
| ±0 .....                                | 94              | _PC constants                    |                 |
| _asm_builtin_atoi() .....               | 3397            | defined in <unistd.h> .....      | 421             |
| _CFLAGS .....                           | 2432            | used in pathconf .....           | 858             |
| _Complex_I .....                        | 210             | _PC_2_SYMLINKS .....             | 858             |
| _CS_PATH .....                          | 419             | _PC_ALLOC_SIZE_MIN .....         | 858             |
| _CS_POSIX_V6_ILP32_OFF32_CFLAGS .....   | 421             | _PC_ASYNC_IO .....               | 858             |
| _CS_POSIX_V6_ILP32_OFF32_LDFLAGS .....  | 421             | _PC_CHOWN_RESTRICTED .....       | 858             |
| _CS_POSIX_V6_ILP32_OFF32_LIBS .....     | 421             | _PC_FILESIZEBITS .....           | 858             |
| _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS .....  | 421             | _PC_LINK_MAX .....               | 858             |
| _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS ..... | 421             | _PC_MAX_CANON .....              | 858             |
| _CS_POSIX_V6_ILP32_OFFBIG_LIBS .....    | 421             | _PC_MAX_INPUT .....              | 858             |
| _CS_POSIX_V6_LP64_OFF64_CFLAGS .....    | 421             | _PC_NAME_MAX .....               | 858             |
| _CS_POSIX_V6_LP64_OFF64_LDFLAGS .....   | 421             | _PC_NO_TRUNC .....               | 858             |
| _CS_POSIX_V6_LP64_OFF64_LIBS .....      | 421             | _PC_PATH_MAX .....               | 858             |
| _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS .....  | 421             | _PC_PIPE_BUF .....               | 858             |
| _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS ..... | 421             | _PC_PRIO_IO .....                | 858             |
| _CS_POSIX_V6_LPBIG_OFFBIG_LIBS .....    | 421             | _PC_REC_INCR_XFER_SIZE .....     | 858             |
| _CS_POSIX_V6_WIDTH_RESTRICTED .....     | 421             | _PC_REC_MAX_XFER_SIZE .....      | 858             |
| ENV5 .....                              | 421             | _PC_REC_MIN_XFER_SIZE .....      | 858             |
| _CS_POSIX_V7_ILP32_OFF32_CFLAGS .....   | 419             | _PC_REC_XFER_ALIGN .....         | 858             |
| _CS_POSIX_V7_ILP32_OFF32_LDFLAGS .....  | 419             | _PC_SYMLINK_MAX .....            | 858             |
| _CS_POSIX_V7_ILP32_OFF32_LIBS .....     | 419             | _PC_SYNC_IO .....                | 858             |
| _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS .....  | 419             | _PC_TIMESTAMP_RESOLUTION .....   | 858             |
| _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS ..... | 419             | _PC_VDISABLE .....               | 858             |
| _CS_POSIX_V7_ILP32_OFFBIG_LIBS .....    | 420             | _POSIX .....                     | 253             |
| _CS_POSIX_V7_LP64_OFF64_CFLAGS .....    | 420             | _POSIX maximum values            |                 |
| _CS_POSIX_V7_LP64_OFF64_LDFLAGS .....   | 420             | in <limits.h> .....              | 258             |
| _CS_POSIX_V7_LP64_OFF64_LIBS .....      | 420             | _POSIX minimum values            |                 |
| _CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS .....  | 420             | in <limits.h> .....              | 258             |
| _CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS ..... | 420             | _POSIX2 constants                |                 |
| _CS_POSIX_V7_LPBIG_OFFBIG_LIBS .....    | 420             | in sysconf .....                 | 2010            |
| _CS_POSIX_V7_WIDTH_RESTRICTED .....     | 420             | _POSIX2_BC_BASE_MAX .....        | 257, 261        |
| ENV5 .....                              | 420             | _POSIX2_BC_DIM_MAX .....         | 257, 261        |
| _CS_V6_ENV .....                        | 421             | _POSIX2_BC_SCALE_MAX .....       | 257, 261        |
| _CS_V7_ENV .....                        | 420             | _POSIX2_BC_STRING_MAX .....      | 258, 261        |
| _exit .....                             | 523, 2136       | _POSIX2_CHARCLASS_NAME_MAX ..... | 258, 261        |
| _Exit() .....                           | 523             | _POSIX2_CHAR_TERM .....          | 417, 2012       |
| _exit() .....                           | 3414, 3432      | _POSIX2_COLL_WEIGHTS_MAX .....   | 258, 261        |
| _Exit() .....                           | 3587            | _POSIX2_C_BIND .....             | 417, 2012       |
| _exit() .....                           | 3587            | _POSIX2_C_DEV .....              | 417, 2012       |
| _FILE .....                             | 575             | _POSIX2_EXPR_NEST_MAX .....      | 258, 261        |
| _Imaginary_I .....                      | 210             | _POSIX2_FORT_DEV .....           | 417, 2012       |
| _IOBF .....                             | 334, 1810, 1851 | _POSIX2_FORT_RUN .....           | 417, 2012       |
|                                         |                 | _POSIX2_LINE_MAX .....           | 258, 262, 264   |

## Index

|                               |                             |                                    |                              |
|-------------------------------|-----------------------------|------------------------------------|------------------------------|
| _POSIX2_LOCALEDEF .....       | 417, 2012                   | _POSIX_MQ_PRIO_MAX .....           | 254, 259                     |
| _POSIX2_PBS .....             | 417, 2012                   | _POSIX_NAME_MAX .....              | 257, 259, 1294               |
| _POSIX2_PBS_ACCOUNTING.....   | 417, 2012                   | .....                              | 1304, 1780, 1788, 1858       |
| _POSIX2_PBS_CHECKPOINT .....  | 417, 2012                   | _POSIX_NGROUPS_MAX .....           | 258-259                      |
| _POSIX2_PBS_LOCATE.....       | 417, 2012                   | _POSIX_NO_TRUNC .....              | 16, 99, 414, 858, 3315       |
| _POSIX2_PBS_MESSAGE .....     | 417, 2012                   | _POSIX_OPEN_MAX .....              | 254, 259, 1031               |
| _POSIX2_PBS_TRACK .....       | 418, 2012                   | _POSIX_PATH_MAX.....               | 257, 259, 385                |
| _POSIX2_RE_DUP_MAX.....       | 255, 258, 262               | .....                              | 1294, 1304, 1780, 1788, 1858 |
| _POSIX2_SW_DEV.....           | 418, 2012                   | _POSIX_PIPE_BUF.....               | 257, 260                     |
| _POSIX2_SYMLINKS .....        | 419                         | _POSIX_PRIORITIZED_IO .....        | 17, 22                       |
| _POSIX2_UPE .....             | 418, 2012                   | .....                              | 414, 477, 2011, 3598         |
| _POSIX2_VERSION .....         | 412, 2012                   | _POSIX_PRIORITY_SCHEDULING .....   | 17, 22-23                    |
| _POSIX .....                  | 449                         | .....                              | 414, 477, 2011, 3598         |
| _POSIX_ADVISORY_INFO .....    | 17, 23, 412                 | _POSIX_PRIO_IO .....               | 419, 858                     |
| .....                         | 860, 2010, 3596             | _POSIX_RAW_SOCKETS.....            | 17, 414, 2011                |
| _POSIX_AIO_LISTIO_MAX .....   | 254, 258                    | _POSIX_READER_WRITER_LOCKS.....    | 17, 414                      |
| _POSIX_AIO_MAX.....           | 254, 259                    | .....                              | 2011, 3320                   |
| _POSIX_ARG_MAX.....           | 254, 259                    | _POSIX_REALTIME_SIGNALS .....      | 17, 414                      |
| _POSIX_ASYNCHRONOUS_IO.....   | 17, 412                     | .....                              | 2011, 3320, 3598             |
| .....                         | 2010, 3320, 3596            | _POSIX_REGEX.....                  | 414, 2011, 3598              |
| _POSIX_ASYNC_IO.....          | 418, 858                    | _POSIX_RE_DUP_MAX.....             | 260                          |
| _POSIX_BARRIERS.....          | 17, 413, 2010, 3320, 3596   | _POSIX_RTSIG_MAX.....              | 255, 260, 3408, 3603         |
| _POSIX_CHILD_MAX.....         | 254, 259                    | _POSIX_SAVED_IDS.....              | 17, 414, 2011, 3315, 3598    |
| _POSIX_CHOWN_RESTRICTED ..... | 16, 413                     | _POSIX_SEMAPHORES.....             | 17, 414, 2011, 3320, 3599    |
| .....                         | 635, 858, 860, 3315, 3597   | _POSIX_SEM_NSEMS_MAX .....         | 255, 260                     |
| _POSIX_CLOCKRES_MIN .....     | 258                         | _POSIX_SEM_VALUE_MAX.....          | 255, 260                     |
| _POSIX_CLOCK_SELECTION .....  | 17, 413                     | _POSIX_SHARED_MEMORY_OBJECTS ..... | 17, 22                       |
| .....                         | 2010, 3320, 3597            | .....                              | 414, 2011, 3599              |
| _POSIX_CPUTIME.....           | 17, 23, 413, 2010, 3597     | _POSIX_SHELL.....                  | 414, 2011, 3599              |
| _POSIX_C_SOURCE .....         | 448, 3398, 3402             | _POSIX_SIGQUEUE_MAX.....           | 255, 260                     |
| _POSIX_DELAYTIMER_MAX.....    | 254, 259                    | _POSIX_SOURCE.....                 | 448, 3398                    |
| _POSIX_FSYNC .....            | 17, 19, 22, 413, 2010, 3597 | _POSIX_SPAWN .....                 | 17, 23, 414, 2011, 3599      |
| _POSIX_HOST_NAME_MAX.....     | 254, 259                    | _POSIX_SPINLOCKS.....              | 3599                         |
| _POSIX_IPV6 .....             | 17, 413, 2010, 3597         | _POSIX_SPIN_LOCKS.....             | 17, 414, 2011, 3320          |
| _POSIX_JOB_CONTROL .....      | 17, 413                     | _POSIX_SPOADIC_SERVER .....        | 17, 23                       |
| .....                         | 2010, 3315, 3597, 3601      | .....                              | 414, 2011, 3599              |
| _POSIX_LINK_MAX.....          | 256, 259                    | _POSIX_SSIZE_MAX.....              | 260, 263                     |
| _POSIX_LOGIN_NAME_MAX.....    | 254, 259                    | _POSIX_SS_REPL_MAX.....            | 255, 260, 2011, 3447         |
| _POSIX_MAPPED_FILES .....     | 17, 413                     | _POSIX_STREAM_MAX .....            | 255, 260                     |
| .....                         | 2011, 3320, 3597            | _POSIX_SYMLINK_MAX .....           | 257, 260                     |
| _POSIX_MAX_CANON.....         | 256, 259                    | _POSIX_SYMLOOP_MAX.....            | 255, 260                     |
| _POSIX_MAX_INPUT .....        | 256, 259                    | _POSIX_SYNCHRONIZED_IO.....        | 17, 22                       |
| _POSIX_MEMLOCK .....          | 17, 22, 413, 2011, 3597     | .....                              | 415, 2011, 3599              |
| _POSIX_MEMLOCK_RANGE .....    | 17, 22                      | _POSIX_SYNC_IO.....                | 419, 858, 3597               |
| .....                         | 413, 2011, 3598             | _POSIX_THREADS .....               | 17, 416, 687                 |
| _POSIX_MEMORY_PROTECTION..... | 17, 413                     | .....                              | 2011, 3320, 3599             |
| .....                         | 2011, 3320, 3598            | _POSIX_THREAD_ATTR_STACKADDR.....  | 17                           |
| _POSIX_MESSAGE_PASSING .....  | 17, 22                      | .....                              | 19, 415, 2011, 3599          |
| .....                         | 413, 2011, 3598             | _POSIX_THREAD_ATTR_STACKSIZE.....  | 18-19                        |
| _POSIX_MONOTONIC_CLOCK.....   | 17, 23                      | .....                              | 415, 2011, 3599              |
| .....                         | 413, 2011, 3598             | _POSIX_THREAD_CPUTIME.....         | 18, 24, 415, 2011            |
| _POSIX_MQ_OPEN_MAX .....      | 254, 259                    | _POSIX_THREAD_DESTRUCTOR .....     |                              |
|                               |                             | ITERATIONS.....                    | 255, 260                     |

|                                         |                            |                              |                        |
|-----------------------------------------|----------------------------|------------------------------|------------------------|
| _POSIX_THREAD_KEYS_MAX .....            | 255, 260                   | in sysconf .....             | 2010                   |
| _POSIX_THREAD_PRIORITY_SCHEDULING ....  |                            | _SC_2_CHAR_TERM.....         | 2012                   |
| .....                                   | 18, 23-24, 415, 2011, 3600 | _SC_2_C_BIND.....            | 2012                   |
| _POSIX_THREAD_PRIO_INHERIT .....        | 18, 23-24                  | _SC_2_C_DEV.....             | 2012                   |
| .....                                   | 415, 2011, 3600            | _SC_2_FORT_DEV .....         | 2012                   |
| _POSIX_THREAD_PRIO_PROTECT .....        | 18, 23-24                  | _SC_2_FORT_RUN .....         | 2012                   |
| .....                                   | 415, 2011, 3600            | _SC_2_LOCALEDEF.....         | 2012                   |
| _POSIX_THREAD_PROCESS_SHARED .....      | 18-19                      | _SC_2_PBS_ACCOUNTING .....   | 2012                   |
| .....                                   | 415, 1609, 2011, 3600      | _SC_2_PBS_CHECKPOINT .....   | 2012                   |
| _POSIX_THREAD_ROBUST_PRIO_INHERIT ..... |                            | _SC_2_PBS_LOCATE.....        | 2012                   |
| .....                                   | 415, 2011                  | _SC_2_PBS_MESSAGE.....       | 2012                   |
| _POSIX_THREAD_ROBUST_PRIO_PROTECT ....  |                            | _SC_2_PBS_TRACK.....         | 2012                   |
| .....                                   | 415, 2011                  | _SC_2_SW_DEV .....           | 2012                   |
| _POSIX_THREAD_SAFE_FUNCTIONS .....      | 17                         | _SC_2_UPE.....               | 2012                   |
| .....                                   | 415, 687, 2011, 3320, 3600 | _SC_2_VERSION.....           | 1372, 2012             |
| _POSIX_THREAD_SPORADIC_SERVER .....     | 18                         | _SC_ADVISORY_INFO.....       | 2010                   |
| .....                                   | 24, 415, 2011, 3600        | _SC_AIO_LISTIO_MAX.....      | 2010                   |
| _POSIX_THREAD_THREADS_MAX .....         | 255, 260                   | _SC_AIO_MAX.....             | 2010                   |
| _POSIX_TIMEOUTS.....                    | 17, 416, 2011, 3320, 3600  | _SC_AIO_PRIO_DELTA_MAX ..... | 2010                   |
| _POSIX_TIMERS.....                      | 17, 416, 2011, 3320, 3600  | _SC_ARG_MAX .....            | 2010                   |
| _POSIX_TIMER_MAX .....                  | 255, 261                   | _SC_ASYNCCHRONOUS_IO.....    | 2010                   |
| _POSIX_TIMESTAMP_RESOLUTION .....       | 419, 858                   | _SC_ATEXIT_MAX .....         | 2010                   |
| _POSIX_TRACE.....                       | 18, 24-25, 416, 2011, 3600 | _SC_BARRIERS .....           | 2010                   |
| _POSIX_TRACE_EVENT_FILTER .....         | 18, 24-25                  | _SC_BC_BASE_MAX.....         | 2010                   |
| .....                                   | 416, 2011, 3600            | _SC_BC_DIM_MAX.....          | 2010                   |
| _POSIX_TRACE_EVENT_NAME_MAX.....        | 256                        | _SC_BC_SCALE_MAX .....       | 2010                   |
| .....                                   | 261, 1450, 1452, 2011      | _SC_BC_STRING_MAX .....      | 2010                   |
| _POSIX_TRACE_INHERIT .....              | 18, 24-25                  | _SC_CHILD_MAX .....          | 2010                   |
| .....                                   | 416, 2011, 3600            | _SC_CLK_TCK .....            | 2010, 2067             |
| _POSIX_TRACE_LOG.....                   | 18, 24-25, 416, 2011, 3601 | _SC_CLOCK_SELECTION .....    | 2010                   |
| _POSIX_TRACE_NAME_MAX.....              | 256, 261, 2011             | _SC_COLL_WEIGHTS_MAX.....    | 2010                   |
| _POSIX_TRACE_SYS_MAX .....              | 256, 261, 1447, 2011       | _SC_CPUTIME .....            | 2010                   |
| _POSIX_TRACE_USER_EVENT_MAX .....       | 256                        | _SC_DELAYTIMER_MAX .....     | 2010                   |
| .....                                   | 261, 1452, 2011            | _SC_EXPR_NEST_MAX.....       | 2010                   |
| _POSIX_TTY_NAME_MAX.....                | 256, 261                   | _SC_FSYNC.....               | 2010                   |
| _POSIX_TYPED_MEMORY_OBJECTS.....        | 18                         | _SC_GETGR_R_SIZE_MAX .....   | 979, 2010              |
| .....                                   | 23, 416, 2011, 3601        | _SC_GETPW_R_SIZE_MAX.....    | 2010                   |
| _POSIX_TZNAME_MAX.....                  | 256, 261, 3371             | _SC_IOV_MAX.....             | 2010                   |
| _POSIX_V6_ILP32_OFF32 .....             | 416, 2012                  | _SC_IPV6 .....               | 2010                   |
| _POSIX_V6_ILP32_OFFBIG.....             | 416, 2012                  | _SC_JOB_CONTROL.....         | 2010                   |
| _POSIX_V6_LP64_OFF64.....               | 416, 2012                  | _SC_LINE_MAX.....            | 2010                   |
| _POSIX_V6_LPBIG_OFFBIG .....            | 416, 2012                  | _SC_LOGIN_NAME_MAX .....     | 2010                   |
| _POSIX_V7_ILP32_OFF32.....              | 416, 2011                  | _SC_MEMLOCK.....             | 2011                   |
| _POSIX_V7_ILP32_OFFBIG.....             | 416, 2011                  | _SC_MEMLOCK_RANGE .....      | 2011                   |
| _POSIX_V7_LP64_OFF64 .....              | 417, 2011                  | _SC_MEMORY_PROTECTION.....   | 2011                   |
| _POSIX_V7_LPBIG_OFFBIG .....            | 417, 2011                  | _SC_MESSAGE_PASSING.....     | 2011                   |
| _POSIX_VDISABLE .....                   | 17, 424, 858, 3114, 3315   | _SC_MONOTONIC_CLOCK .....    | 2011                   |
| _POSIX_VERSION .....                    | 16, 412, 2011, 2098        | _SC_MQ_OPEN_MAX .....        | 2010                   |
| _PROCESS .....                          | 451                        | _SC_MQ_PRIO_MAX.....         | 2010                   |
| _PTHREAD_THREADS_MAX.....               | 1579                       | _SC_NGROUPS_MAX .....        | 2010                   |
| _SC constants                           |                            | _SC_OPEN_MAX.....            | 2010                   |
| defined in <unistd.h> .....             | <b>421</b>                 | _SC_PAGESIZE.....            | 1378, 2012, 3431, 3433 |
|                                         |                            | _SC_PAGE_SIZE.....           | 2012                   |



## Index

|                                       |      |                                  |                        |
|---------------------------------------|------|----------------------------------|------------------------|
| _SC_PRIORITIZED_IO.....               | 2011 | _SC_TZNAME_MAX.....              | 2012                   |
| _SC_PRIORITY_SCHEDULING .....         | 2011 | _SC_V6_ILP32_OFF32 .....         | 2012                   |
| _SC_RAW_SOCKETS.....                  | 2011 | _SC_V6_ILP32_OFFBIG .....        | 2012                   |
| _SC_READER_WRITER_LOCKS.....          | 2011 | _SC_V6_LP64_OFF64.....           | 2012                   |
| _SC_REALTIME_SIGNALS .....            | 2011 | _SC_V6_LPBIG_OFFBIG.....         | 2012                   |
| _SC_REGEX.....                        | 2011 | _SC_V7_ILP32_OFF32 .....         | 2011                   |
| _SC_RE_DUP_MAX.....                   | 2012 | _SC_V7_ILP32_OFFBIG.....         | 2011                   |
| _SC_RTSIG_MAX.....                    | 2012 | _SC_V7_LP64_OFF64.....           | 2011                   |
| _SC_SAVED_IDS .....                   | 2011 | _SC_V7_LPBIG_OFFBIG .....        | 2011                   |
| _SC_SEMAPHORES.....                   | 2011 | _SC_VERSION .....                | 2011                   |
| _SC_SEM_NSEMS_MAX .....               | 2012 | _SC_XOPEN_CRYPT .....            | 2012                   |
| _SC_SEM_VALUE_MAX .....               | 2012 | _SC_XOPEN_ENH_I18N .....         | 2012                   |
| _SC_SHARED_MEMORY_OBJECTS .....       | 2011 | _SC_XOPEN_REALTIME .....         | 2012                   |
| _SC_SHELL .....                       | 2011 | _SC_XOPEN_REALTIME_THREADS ..... | 2012                   |
| _SC_SIGQUEUE_MAX.....                 | 2012 | _SC_XOPEN_SHM.....               | 2012                   |
| _SC_SPAWN.....                        | 2011 | _SC_XOPEN_STREAMS .....          | 2012                   |
| _SC_SPIN_LOCKS .....                  | 2011 | _SC_XOPEN_UNIX .....             | 2012                   |
| _SC_SPORADIC_SERVER .....             | 2011 | _SC_XOPEN_VERSION .....          | 2012                   |
| _SC_SS_REPL_MAX .....                 | 2011 | _setjmp .....                    | 528                    |
| _SC_STREAM_MAX.....                   | 2012 | _setjmp().....                   | 3590                   |
| _SC_SYMLOOP_MAX.....                  | 2012 | _t.....                          | 451                    |
| _SC_SYNCHRONIZED_IO.....              | 2011 | _TIME.....                       | 451                    |
| _SC_THREADS.....                      | 2011 | _tolower().....                  | 530                    |
| _SC_THREAD_ATTR_STACKADDR.....        | 2011 | _toupper() .....                 | 531                    |
| _SC_THREAD_ATTR_STACKSIZE.....        | 2011 | _XOPEN_CRYPT .....               | 18, 22, 418, 2012      |
| _SC_THREAD_CPUTIME.....               | 2011 | _XOPEN_ENH_I18N .....            | 418, 2012              |
| _SC_THREAD_DESTRUCTOR_ITERATIONS..... | 2012 | _XOPEN_IOV_MAX.....              | 254, 262               |
| .....                                 | 2012 | _XOPEN_NAME_MAX.....             | 257, 262, 1294         |
| _SC_THREAD_KEYS_MAX .....             | 2012 | .....                            | 1304, 1780, 1788, 1858 |
| _SC_THREAD_PRIORITY_SCHEDULING .....  | 2011 | _XOPEN_PATH_MAX.....             | 257, 262, 1294         |
| _SC_THREAD_PRIO_INHERIT .....         | 2011 | .....                            | 1304, 1780, 1788, 1858 |
| _SC_THREAD_PRIO_PROTECT .....         | 2011 | _XOPEN_REALTIME .....            | 18, 22, 418, 787, 2012 |
| _SC_THREAD_PROCESS_SHARED.....        | 2011 | _XOPEN_REALTIME_THREADS .....    | 18, 23                 |
| _SC_THREAD_ROBUST_PRIO_INHERIT.....   | 2011 | .....                            | 418, 2012              |
| _SC_THREAD_ROBUST_PRIO_PROTECT .....  | 2011 | _XOPEN_SHM.....                  | 418, 2012              |
| _SC_THREAD_SAFE_FUNCTIONS .....       | 2011 | _XOPEN_SOURCE .....              | 448-449, 3399          |
| _SC_THREAD_SPORADIC_SERVER .....      | 2011 | _XOPEN_STREAMS .....             | 18, 25, 418, 2012      |
| _SC_THREAD_STACK_MIN .....            | 2012 | _XOPEN_UNIX .....                | 18-19, 418, 2012       |
| _SC_THREAD_THREADS_MAX .....          | 2012 | _XOPEN_UUCP .....                | 418                    |
| _SC_TIMEOUTS .....                    | 2011 | _XOPEN_VERSION .....             | 19, 412, 2012          |
| _SC_TIMERS .....                      | 2011 | __errno().....                   | 3406                   |
| _SC_TIMER_MAX.....                    | 2012 | a64l() .....                     | 532                    |
| _SC_TRACE.....                        | 2011 | ABDAY.....                       | 250                    |
| _SC_TRACE_EVENT_FILTER.....           | 2011 | ABDAY_1 .....                    | 1340                   |
| _SC_TRACE_EVENT_NAME_MAX.....         | 2011 | ABMON .....                      | 250                    |
| _SC_TRACE_INHERIT.....                | 2011 | abort() .....                    | 534, 3587              |
| _SC_TRACE_LOG.....                    | 2011 | abortive release .....           | 33                     |
| _SC_TRACE_NAME_MAX .....              | 2011 | abs().....                       | 536                    |
| _SC_TRACE_SYS_MAX .....               | 2011 | absolute pathname .....          | 33, 99                 |
| _SC_TRACE_USER_EVENT_MAX .....        | 2011 | accept() .....                   | 537                    |
| _SC_TTY_NAME_MAX .....                | 2012 | access.....                      | 3588                   |
| _SC_TYPED_MEMORY_OBJECTS.....         | 2011 | access mode.....                 | 33                     |
|                                       |      | access().....                    | 539, 3346              |

|                                                |                              |
|------------------------------------------------|------------------------------|
| Account_Name .....                             | 2326                         |
| acos() .....                                   | <b>542</b>                   |
| acosf .....                                    | 542                          |
| acosh() .....                                  | <b>544</b>                   |
| acoshf .....                                   | 544                          |
| acoshl .....                                   | 544                          |
| acosl .....                                    | 542                          |
| acosl() .....                                  | <b>546</b>                   |
| ACTION .....                                   | 1059                         |
| actions equivalent to functions .....          | 2231                         |
| active trace stream .....                      | 3517                         |
| adb, rationale for omission .....              | 3576                         |
| additional file access control mechanism ..... | 33                           |
| address families .....                         | 3491                         |
| address information .....                      | 888                          |
| address space .....                            | 33                           |
| address string .....                           | 888                          |
| addressing .....                               | 3491                         |
| addrinfo structure .....                       | 888                          |
| admin .....                                    | <b>2344</b>                  |
| ADV .....                                      | 7                            |
| advanced realtime .....                        | 23                           |
| ADVANCED REALTIME .....                        | 20, 642, 1302-1303, 1374     |
| .....1376, 1378, 1380, 1382, 1385, 1393, 1396  |                              |
| .....1398-1399, 1401, 1403, 1405, 1407, 1409   |                              |
| .....1411, 1413, 1415-1422, 1476, 1478         |                              |
| advanced realtime threads .....                | 24                           |
| ADVANCED REALTIME THREADS .....                | 1573                         |
| advisory information .....                     | 33, 3418                     |
| affirmative response .....                     | 34                           |
| AF .....                                       | 451                          |
| AF_INET .....                                  | 367                          |
| AF_INET6 .....                                 | 367                          |
| AF_UNIX .....                                  | 367                          |
| AF_UNSPEC .....                                | 367                          |
| AIO_ .....                                     | 450                          |
| aio_ .....                                     | 450                          |
| AIO_ALLDONE .....                              | 206, 547                     |
| aio_cancel() .....                             | <b>547</b> , 3426-3427       |
| AIO_CANCELED .....                             | 206, 547                     |
| aio_error() .....                              | <b>549</b>                   |
| aio_fsync() .....                              | <b>551</b> , 3411, 3425      |
| AIO_LISTIO_MAX .....                           | 253, 1186, 2010, 3601        |
| AIO_MAX .....                                  | 254, 1186, 2010, 3601        |
| AIO_NOTCANCELED .....                          | 206, 547                     |
| AIO_PRIO_DELTA_MAX .....                       | 254, 477, 2010, 3601         |
| aio_read() .....                               | <b>553</b> , 3427            |
| aio_return() .....                             | <b>556</b>                   |
| aio_suspend() .....                            | <b>558</b> , 3425, 3452      |
| aio_write() .....                              | <b>560</b> , 3427            |
| ai_ .....                                      | 450                          |
| AI_ADDRCONFIG .....                            | 284, 889                     |
| AI_ALL .....                                   | 284, 889                     |
| AI_CANONNAME .....                             | 284, 889                     |
| AI_INET .....                                  | 889                          |
| AI_INET6 .....                                 | 889                          |
| AI_NUMERICHOST .....                           | 284, 889                     |
| AI_NUMERICSERV .....                           | 284, 889                     |
| AI_PASSIVE .....                               | 284, 889                     |
| AI_V4MAPPED .....                              | 284, 889                     |
| alarm .....                                    | 3587                         |
| alarm() .....                                  | <b>563</b> , 3416, 3451      |
| alert .....                                    | 34                           |
| alert character .....                          | 34                           |
| alias .....                                    | 2244, 2264, 2349, 3545, 3591 |
| alias name .....                               | 34                           |
| alias substitution .....                       | 2248, 3548                   |
| alignment .....                                | 34                           |
| alphasort() .....                              | <b>565</b>                   |
| alternate file access control mechanism .....  | 34                           |
| alternate signal stack .....                   | 34                           |
| ALT_DIGITS .....                               | 250                          |
| AM_STR .....                                   | 250                          |
| anchoring .....                                | 173                          |
| ancillary data .....                           | 35                           |
| AND lists .....                                | 2267, 3564                   |
| AND-OR list .....                              | 2266                         |
| angle brackets .....                           | 35                           |
| anycast .....                                  | 504                          |
| ANYMARK .....                                  | 348, 1095                    |
| API .....                                      | 35                           |
| appending redirected output .....              | 2260                         |
| application .....                              | 35                           |
| application address .....                      | 35                           |
| application conformance .....                  | 28                           |
| application instrumentation .....              | 3505                         |
| application program interface .....            | 35                           |
| application-managed thread stack .....         | 495, 3490                    |
| appropriate privilege .....                    | 3324, 3336                   |
| appropriate privileges .....                   | 35, 541, 859                 |
| ar .....                                       | <b>2352</b> , 3593-3594      |
| arbitrary file size .....                      | 3543                         |
| archives                                       |                              |
| ar command .....                               | 2352                         |
| AREGTYPE .....                                 | 391                          |
| argc .....                                     | 752                          |
| argument .....                                 | 36                           |
| ARG_MAX .....                                  | 254, 456, 746, 749, 754      |
| .....2010, 3287, 3333, 3537, 3601              |                              |
| arithmetic expansion .....                     | 2257, 3554                   |
| arithmetic language                            |                              |
| bc .....                                       | 2409                         |
| arithmetic precision and operations .....      | 2231                         |
| arm (a timer) .....                            | 36                           |
| array identifiers .....                        | 2414                         |
| as, rationale for omission .....               | 3576                         |

## Index

- asa.....2359, 3592-3593, 3595  
ASCII.....3335  
asctime().....567  
asctime\_r.....567  
asin().....570  
asinf.....570  
asinh().....572  
asinhf.....572  
asinhf.....572  
asinl.....570  
asinl().....574  
assert().....575  
asterisk.....36  
async-cancel safety.....3488  
async-cancel-safe function.....36  
async-signal-safe.....1493, 3414  
async-signal-safe function.....36  
asynchronous error.....3492  
asynchronous events.....36  
asynchronous I/O.....3425, 3589  
asynchronous I/O completion.....37  
asynchronous I/O operation.....37  
asynchronous input and output.....36  
asynchronous lists.....2266, 3563  
asynchronously-generated signal.....37  
at.....2362, 3591  
at-job.....2362  
atan().....576  
atan2().....578  
atan2f.....578  
atan2l.....578  
atanf.....576  
atanf().....580  
atanh().....581  
atanhf.....581  
atanhl.....581  
atanl.....576  
atanl().....583  
atexit().....584, 3488  
ATEXIT\_MAX.....254, 584, 2010  
atof().....586  
atoi().....587, 3397-3398  
atol().....589  
atoll.....589  
attributes, clock-resolution.....513, 1425  
attributes, creation-time.....513, 1425  
attributes, generation-version.....513, 1425  
attributes, inheritance.....513, 1427  
attributes, log-full-policy.....511, 513, 1427, 1430  
attributes, log-max-size.....513, 1428, 1430  
attributes, max-data-size.....513, 1430-1431  
attributes, stream-full-policy.....510-511, 513, 1428  
attributes, stream-min-size.....513, 1431  
attributes, trace-name.....513, 1425  
attributes, truncation-status.....1450  
AT\_EACCESS.....225  
AT\_FDCWD.....225, 539, 631, 635, 916, 936  
.....1180, 1253, 1259, 1264, 1346, 1708  
.....1741, 2005, 3144  
AT\_REMOVEDIR.....226  
AT\_SYMLINK\_FOLLOW.....225, 1180  
AT\_SYMLINK\_NOFOLLOW.....225, 631  
.....635, 916, 936  
authentication.....37  
authorization.....37  
automatic storage class.....2417  
awk.....2371, 3592-3593  
actions.....2381  
arithmetic functions.....2383  
escape sequences.....2379  
expression patterns.....2381  
expressions.....2374  
functions.....2383  
grammar.....2386  
input/output and general functions.....2385  
lexical conventions.....2393  
output statements.....2382  
overall program structure.....2373  
pattern ranges.....2381  
patterns.....2380  
regular expressions.....2379  
special patterns.....2380  
string functions.....2384  
user-defined functions.....2386  
variables and special variables.....2377  
background.....1829, 3329-3332, 3381  
background job.....37  
background process.....37, 2042  
background process group.....37  
background work  
at.....2362  
batch.....2406  
bg.....2424  
crontab.....2490  
fg.....2658  
jobs.....2737  
nice.....2885  
nohup.....2898  
renice.....3043  
backquote.....38  
BACKREF.....177  
backslash.....38, 3546  
backspace character.....38  
bandinfo.....345  
banner, rationale for omission.....3576

|                                      |                               |
|--------------------------------------|-------------------------------|
| barrier.....                         | 38                            |
| barriers.....                        | 3467                          |
| basename.....                        | 38, 2403, 3591-3592           |
| basename().....                      | <b>590</b>                    |
| basic regular expression.....        | 38, 169, 3373                 |
| batch.....                           | <b>2406</b> , 3591            |
| general concepts.....                | 3573                          |
| batch access list.....               | 38                            |
| batch administration.....            | 2322                          |
| batch administrator.....             | 38                            |
| batch authorization.....             | 2322                          |
| batch client.....                    | 39                            |
| batch client-server interaction..... | 2319                          |
| batch destination.....               | 39                            |
| batch destination identifier.....    | 39                            |
| batch directive.....                 | 39                            |
| batch environment.....               | 3570                          |
| option definitions.....              | 3571                          |
| services.....                        | 2319                          |
| utilities.....                       | 2319                          |
| batch environment utilities          |                               |
| common behavior.....                 | 3576                          |
| batch job.....                       | 39                            |
| Batch Job Abort.....                 | 2321, 2332                    |
| batch job attribute.....             | 40                            |
| batch job creation.....              | 2320                          |
| Batch Job Execution.....             | 2321, 2324                    |
| Batch Job Exit.....                  | 2321, 2331                    |
| batch job identifier.....            | 40, 2340                      |
| Batch Job Message Request.....       | 2334                          |
| batch job name.....                  | 40                            |
| batch job owner.....                 | 40                            |
| batch job priority.....              | 40                            |
| Batch Job Routing.....               | 2320, 2331                    |
| batch job state.....                 | 40                            |
| batch job states.....                | 2323                          |
| Batch Job Status Request.....        | 2335                          |
| batch job tracking.....              | 2320                          |
| batch name service.....              | 40                            |
| batch name space.....                | 40                            |
| batch node.....                      | 41                            |
| batch notification.....              | 2322                          |
| batch operator.....                  | 41                            |
| batch queue.....                     | 41, 2320                      |
| batch queue attribute.....           | 41                            |
| batch queue position.....            | 41                            |
| batch queue priority.....            | 41                            |
| Batch Queue Status Request.....      | 2337                          |
| batch rerunability.....              | 41                            |
| batch restart.....                   | 42                            |
| batch server.....                    | 42                            |
| batch server name.....               | 42                            |
| Batch Server Restart.....            | 2332                          |
| batch service.....                   | 42                            |
| batch service request.....           | 42                            |
| batch services.....                  | 2322, 3575                    |
| batch submission.....                | 42                            |
| batch system.....                    | 42                            |
| batch systems                        |                               |
| historical implementations.....      | 3570                          |
| history.....                         | 3570                          |
| batch target user.....               | 43                            |
| batch user.....                      | 43                            |
| baud rate functions.....             | 624                           |
| bc.....                              | <b>2409</b> , 3591-3592, 3596 |
| grammar.....                         | 2410                          |
| lexical conventions.....             | 2412                          |
| operations.....                      | 2414                          |
| operators.....                       | 2414                          |
| bcc (mailer blind carbon copy).....  | 2828                          |
| bcmp.....                            | 3520                          |
| bcopy.....                           | 3520                          |
| BC_constants                         |                               |
| in sysconf.....                      | 2010                          |
| BC_BASE_MAX.....                     | 257, 2010, 2234, 3538         |
| BC_DIM_MAX.....                      | 257, 2010, 2234, 3538         |
| BC_SCALE_MAX.....                    | 257, 2010, 2234, 3538         |
| BC_STRING_MAX.....                   | 258, 2010, 2234, 2412         |
| BE.....                              | 7                             |
| bg.....                              | 2244, 2264, 2424, 3545, 3591  |
| binary primaries.....                | 3133                          |
| bind.....                            | 43                            |
| bind().....                          | <b>592</b>                    |
| bi.....                              | 450                           |
| blank character.....                 | 43                            |
| blank line.....                      | 43                            |
| blkcnt_t.....                        | 380                           |
| blksize_t.....                       | 380                           |
| BLKTYPE.....                         | 391                           |
| block special file.....              | 44                            |
| block-mode terminal.....             | 43                            |
| blocked process (or thread).....     | 43                            |
| blocking.....                        | 43                            |
| BOOT_TIME.....                       | 433, 733-734                  |
| bounded response.....                | 3589                          |
| braces.....                          | 44                            |
| bracket expression                   |                               |
| grammar.....                         | 3378                          |
| brackets.....                        | 44                            |
| BRE                                  |                               |
| expression anchoring.....            | 3375                          |
| grammar lexical conventions.....     | 3377                          |
| matching a collating element.....    | 3373                          |
| matching a single character.....     | 3373                          |
| matching multiple characters.....    | 3375                          |

## Index

|                                              |                                          |
|----------------------------------------------|------------------------------------------|
| ordinary character .....                     | 3373                                     |
| periods .....                                | 3373                                     |
| precedence .....                             | 3375                                     |
| special character .....                      | 3373                                     |
| BRE (ERE) matching a single character .....  | 168                                      |
| BRE (ERE) matching multiple characters ..... | 168                                      |
| break .....                                  | <b>2281</b>                              |
| BRKINT .....                                 | 394                                      |
| broadcast .....                              | 44                                       |
| broadcasting a condition .....               | 1542                                     |
| BSD .....                                    | 217, 526, 563, 637, 785, 860, 1012       |
| .....                                        | 1164, 1254, 1325, 1701, 1743, 1750, 1829 |
| .....                                        | 1875, 1900, 2034, 2057, 2098, 2136, 3326 |
| .....                                        | 3329, 3332, 3334, 3380-3382, 3406-3407   |
| .....                                        | 3409, 3413-3414, 3518                    |
| BSDLY .....                                  | 395                                      |
| bsd_signal .....                             | 3520                                     |
| bsearch() .....                              | <b>595</b>                               |
| BSn .....                                    | 395                                      |
| btowc() .....                                | <b>598</b>                               |
| buffer cache .....                           | 923                                      |
| BUFSIZ .....                                 | 334, 1810                                |
| built-in .....                               | 44                                       |
| built-in utilities .....                     | 2244, 3544                               |
| built-in utility .....                       | 44                                       |
| builtin .....                                | 2479                                     |
| BUS_ .....                                   | 450, 452                                 |
| BUS_ADRALN .....                             | 316                                      |
| BUS_ADRERR .....                             | 316                                      |
| BUS_OBJERR .....                             | 316                                      |
| byte .....                                   | 44                                       |
| byte input/output functions .....            | 45                                       |
| byte-oriented stream .....                   | 472                                      |
| byte-stream mode .....                       | 1698                                     |
| bzero .....                                  | 3520                                     |
| C Shell .....                                | 3329-3331                                |
| C-language extensions .....                  | 3585, 3590                               |
| c99 .....                                    | <b>2427</b> , 3593                       |
| external symbols .....                       | 2431                                     |
| standard libraries .....                     | 2430                                     |
| cabs() .....                                 | <b>599</b>                               |
| cabsf .....                                  | 599                                      |
| cabsl .....                                  | 599                                      |
| cacos() .....                                | <b>600</b>                               |
| cacosf .....                                 | 600                                      |
| cacosh() .....                               | <b>601</b>                               |
| cacoshf .....                                | 601                                      |
| cacoshl .....                                | 601                                      |
| caosl .....                                  | 600                                      |
| caosl() .....                                | <b>602</b>                               |
| cal .....                                    | <b>2436</b>                              |
| calendar, rationale for omission .....       | 3576                                     |
| calloc() .....                               | <b>603</b>                               |
| can .....                                    | 5                                        |
| cancel, rationale for omission .....         | 3576                                     |
| cancel-safe .....                            | 1656                                     |
| cancelability state .....                    | 490, 1580, 1656                          |
| cancelability type .....                     | 1580, 1656                               |
| canceling execution of a thread .....        | 1534                                     |
| cancellation cleanup handler .....           | 1539, 1551                               |
| .....                                        | 1569, 1583, 3486, 3488                   |
| cancellation cleanup stack .....             | 3486                                     |
| cancellation points .....                    | 491                                      |
| canonical mode input processing .....        | 187, 3382                                |
| canonical name .....                         | 889                                      |
| carg() .....                                 | <b>605</b>                               |
| cargf .....                                  | 605                                      |
| cargl .....                                  | 605                                      |
| carriage-control characters .....            | 2359                                     |
| carriage-return character .....              | 45                                       |
| case .....                                   | 3564                                     |
| case conditional construct .....             | 2269                                     |
| case folding .....                           | 3346-3347                                |
| casin() .....                                | <b>606</b>                               |
| casinf .....                                 | 606                                      |
| casinh() .....                               | <b>607</b>                               |
| casinhf .....                                | 607                                      |
| casinhl .....                                | 607                                      |
| casinl .....                                 | 606                                      |
| casinl() .....                               | <b>608</b>                               |
| cat .....                                    | <b>2438</b> , 3543                       |
| catan() .....                                | <b>609</b>                               |
| catanf .....                                 | 609                                      |
| catanh() .....                               | <b>610</b>                               |
| catanhf .....                                | 610                                      |
| catanhf .....                                | 610                                      |
| catanl .....                                 | 609                                      |
| catanl() .....                               | <b>611</b>                               |
| catclose() .....                             | <b>612</b> , 3592                        |
| catgets() .....                              | <b>613</b> , 3592                        |
| catopen() .....                              | <b>615</b> , 3592                        |
| CBAUD .....                                  | 452                                      |
| cbrt() .....                                 | <b>617</b>                               |
| cbrtf .....                                  | 617                                      |
| cbrtl .....                                  | 617                                      |
| cc (mailer carbon copy) .....                | 2828                                     |
| ccos() .....                                 | <b>618</b>                               |
| ccosf .....                                  | 618                                      |
| ccosh() .....                                | <b>619</b>                               |
| ccoshf .....                                 | 619                                      |
| ccoshl .....                                 | 619                                      |
| ccosl .....                                  | 618                                      |
| ccosl() .....                                | <b>620</b>                               |
| CD .....                                     | 7                                        |
| cd .....                                     | 2244, 2264, 2442, 3545, 3592             |

- ceil() .....621  
 ceilf .....621  
 ceill .....621  
 CEO .....3374  
 cexp() .....623  
 cexpf .....623  
 cexpl .....623  
 cfgetispeed() .....624  
 cfgetospeed() .....626  
 cflow .....2447  
 cfsetispeed() .....627  
 cfsetospeed() .....628  
 change current working directory .....630  
 change file modes .....633  
 change history .....3316, 3393, 3535  
 change owner and group of file .....637  
 changing the current working directory .....2231  
 char .....519  
 char type .....3519  
 character .....45, 3324  
 character array .....45  
 character class .....45  
 character counting .....3271  
 character encoding .....114, 3355  
   state-dependent .....118  
 character set .....45, 3354  
 character set description file .....3355  
 character set, portable filename .....3335  
 character special file .....46  
 character string .....46  
 character, rationale .....3324  
 CHARCLASS\_NAME\_MAX .....258, 3360  
 charmap  
   description .....115  
   with localedef .....2774  
   writing names with locale .....2769  
 charmap file .....2773, 3114  
 CHAR\_BIT .....262  
 CHAR\_MAX .....262, 1197, 1199, 3362  
 CHAR\_MIN .....262  
 chdir() .....629  
 Checkpoint .....2326  
 checksums  
   cksum .....2464  
 chgrp .....2450, 3543, 3592-3593  
 child process .....46, 3325  
 CHILD\_MAX .....254, 854, 2010, 3315  
   .....3518, 3538, 3563, 3601  
 chmod .....2453, 3543, 3588, 3592-3593  
   grammar .....2456  
 chmod() .....631, 3338  
 chown .....2460, 3543, 3588, 3592-3593  
 chown() .....635, 3337-3338  
 chroot() .....3335  
 chroot, rationale for omission .....3576  
 CHRTYPE .....391  
 cimag() .....639  
 cimagf .....639  
 cimagl .....639  
 circumflex .....46  
 cksum .....2464, 3543, 3592  
 CLD\_ .....450, 452  
 CLD\_CONTINUED .....316  
 CLD\_DUMPED .....316  
 CLD\_EXITED .....316  
 CLD\_KILLED .....316  
 CLD\_STOPPED .....316  
 CLD\_TRAPPED .....316  
 clearerr() .....640  
 CLOCAL .....396  
 clock .....46, 3447  
 clock jump .....46  
 clock tick .....46, 563, 2013, 2067, 3325  
 clock tick, rationale .....3325  
 clock ticks/second .....2010  
 clock() .....641  
 clock-resolution attribute .....513, 1425  
 clockid\_t .....380  
 CLOCKRES\_MIN .....3601  
 clocks .....3447  
 CLOCKS\_PER\_SEC .....380, 403, 641  
 CLOCK\_ .....451  
 clock .....451  
 clock\_getcpuclockid() .....642, 3454-3455  
 clock\_getres() .....643  
 clock\_gettime .....643  
 CLOCK\_MONOTONIC .....403, 484, 647  
 clock\_nanosleep() .....646, 3452  
 CLOCK\_PROCESS\_CPUTIME\_ID .....403  
   .....485, 3454-3455  
 CLOCK\_REALTIME .....258, 403, 484  
   .....643, 647, 1325, 1604, 2059, 3447-3452  
 clock\_settime .....643  
 clock\_settime() .....649  
 clock\_t .....380  
 CLOCK\_THREAD\_CPUTIME\_ID .....403  
   .....485, 3454-3455  
 clog() .....650  
 clogf .....650  
 clogl .....650  
 close .....3588  
 close a file .....653  
 close() .....651, 3432  
 closedir .....3588  
 closedir() .....654

## Index

|                              |                                        |                                              |                                |
|------------------------------|----------------------------------------|----------------------------------------------|--------------------------------|
| closelog()                   | 656, 3593                              | composite graphic symbol                     | 48                             |
| cmp                          | 2469, 3543, 3592                       | compound commands                            | 2268, 3564                     |
| cmsh                         | 451                                    | compound-list                                | 2266                           |
| CMSG_                        | 452                                    | compress                                     | 2480                           |
| CMSG_DATA                    | 366                                    | compression                                  |                                |
| CMSG_FIRSTHDR                | 366                                    | compress                                     | 2480                           |
| CMSG_NXTHDR                  | 366                                    | uncompress                                   | 3178                           |
| coded character set          | 46                                     | zcat                                         | 3306                           |
| codes                        | 3318                                   | concepts                                     | 3318                           |
| codeset                      | 47                                     | concurrent execution                         | 95, 3345                       |
| CODESET                      | 250                                    | concurrent execution of processes            | 2227                           |
| codeset conversion           | 2722                                   | condition variable                           | 48                             |
| tr                           | 3152                                   | condition variable initialization attributes | 1554                           |
| col, rationale for omission  | 3576                                   | conditional construct                        |                                |
| collating element            | 47                                     | case                                         | 3564                           |
| collating element order      | 3374                                   | if                                           | 3565                           |
| collation                    | 47                                     | configurable limits                          | 3596, 3601                     |
| collation sequence           | 47                                     | configuration interrogation                  | 3584, 3587                     |
| COLL_ELEM_MULTI              | 177                                    | configuration options                        | 3594                           |
| COLL_ELEM_SINGLE             | 177                                    | shell and utilities                          | 3594                           |
| COLL_WEIGHTS_MAX             | 258, 2010, 2234, 3538                  | system interfaces                            | 3596                           |
| colon                        | 2283                                   | configuration values                         | 2701                           |
| column position              | 47, 3325                               | conformance                                  | 15, 28, 3316, 3319, 3322, 3324 |
| COLUMNS                      | 163, 3370                              |                                              | 3348, 3517                     |
| comm                         | 2472, 3592                             | POSIX                                        | 15                             |
| command                      | 48, 2244, 2264, 2475, 3325, 3545, 3591 | POSIX system interfaces                      | 16                             |
| command execution            | 3561                                   | XSI                                          | 15                             |
| command interpreter          |                                        | XSI system interfaces                        | 19                             |
| portable                     | 2136                                   | conformance document                         | 15, 3316                       |
| command language             | 3585, 3590                             | conformance document, rationale              | 3316                           |
| command language interpreter | 48                                     | conforming application                       | 15, 1917                       |
| command mode                 | 2553                                   |                                              | 2357, 3322, 3407, 3541, 3543   |
| command search               | 3561                                   | conforming application, strictly             | 563                            |
| command search and execution | 2264                                   |                                              | 752, 3319, 3322, 3414          |
| command substitution         | 2256, 3553                             | conforming implementation options            | 20                             |
| communications commands      |                                        | confstr                                      | 3588                           |
| mailx                        | 2806                                   | confstr()                                    | 660                            |
| talk                         | 3124                                   | conj()                                       | 663                            |
| uucp                         | 3193                                   | confj                                        | 663                            |
| uudecode                     | 3197                                   | conjl                                        | 663                            |
| uuencode                     | 3200                                   | connect()                                    | 664                            |
| uustat                       | 3205                                   | connected socket                             | 48                             |
| uux                          | 3208                                   | connection                                   | 48                             |
| write                        | 3280                                   | connection indication queue                  | 3492                           |
| compare thread IDs           | 1568                                   | connection mode                              | 48                             |
| compilation environment      | 448, 3398                              | connectionless mode                          | 48                             |
| compilers                    |                                        | consequences of shell errors                 | 2262                           |
| c99                          | 2427                                   | continue                                     | 2285                           |
| fort77                       | 2680                                   | control character                            | 49                             |
| yacc                         | 3290                                   | control characters                           | 3111                           |
| complex                      | 210                                    | control data                                 | 473                            |
| complex data manipulation    | 3585, 3592                             | control mode                                 | 3384                           |
|                              |                                        | control operator                             | 49                             |

|                                  |                            |                                         |                 |
|----------------------------------|----------------------------|-----------------------------------------|-----------------|
| control-normal                   | 1698                       | creall                                  | 675             |
| controlling process              | 49                         | creat()                                 | 676, 3432, 3543 |
| controlling terminal             | 49, 186, 2227              | create a per-process timer              | 2060            |
|                                  | 3326, 3381, 3588           | create an interprocess channel          | 1366            |
| CONTTYTYPE                       | 391                        | create session and set process group ID | 1841            |
| conversion descriptor            | 49, 746, 751               | creation-time attribute                 | 513, 1425       |
|                                  | 1065-1066, 1068-1069       | CRn                                     | 394             |
| conversion specification         | 864, 900                   | CRNCYSTR                                | 250             |
|                                  | 940, 949, 1955, 1959, 1975 | cron daemon                             | 2493            |
| modified                         | 1961                       | crontab                                 | 2490, 3591      |
| conversion specifier             |                            | CRYPT                                   | 678, 719, 1822  |
| modified                         | 1976                       | crypt()                                 | 678             |
| Coordinated Universal Time (UTC) | 2515                       | csin()                                  | 680             |
| copy                             | 140                        | csinf                                   | 680             |
| copy files commands              |                            | csinh()                                 | 681             |
| cp                               | 2483                       | csinhf                                  | 681             |
| dd                               | 2517                       | csinhl                                  | 681             |
| ln                               | 2765                       | csinl                                   | 680             |
| mv                               | 2876                       | csinl()                                 | 682             |
| pax                              | 2925                       | CSIZE                                   | 396, 3384       |
| copysign()                       | 667                        | CSn                                     | 396             |
| copysignf                        | 667                        | csplit                                  | 2494, 3592      |
| copysignl                        | 667                        | csqrt()                                 | 683             |
| core                             | 2136, 3346                 | csqrtf                                  | 683             |
| core file                        | 49, 525                    | csqrtl                                  | 683             |
| cos()                            | 668                        | CSTOPB                                  | 396             |
| cosf                             | 668                        | ctags                                   | 2498, 3593      |
| cosh()                           | 670                        | ctan()                                  | 684             |
| coshf                            | 670                        | ctanf                                   | 684             |
| coshl                            | 670                        | ctanh()                                 | 685             |
| cosl                             | 668                        | ctanhf                                  | 685             |
| cosl()                           | 672                        | ctanhl                                  | 685             |
| covert channel                   | 1164, 3348                 | ctanl                                   | 684             |
| cp                               | 2483, 3543, 3592           | ctanl()                                 | 686             |
| cpio format                      | 2945                       | ctermid()                               | 687             |
| cpio, rationale for omission     | 3576                       | ctime()                                 | 689, 3590       |
| cpow()                           | 673                        | ctime_r                                 | 689             |
| cpowf                            | 673                        | cu, rationale for omission              | 3576            |
| cpowl                            | 673                        | currency_symbol                         | 140             |
| cpp, rationale for omission      | 3576                       | current job                             | 50              |
| cproj()                          | 674                        | current working directory               | 50, 94, 2227    |
| cprojf                           | 674                        | cursor position                         | 50              |
| cprojl                           | 674                        | cut                                     | 2503, 3592      |
| CPT                              | 7                          | CX                                      | 7               |
| CPU                              | 379                        | cxref                                   | 2507            |
| CPU time                         | 49, 3142                   | c_                                      | 451             |
| clock                            | 50                         | C_ constants in <cpio.h>                | 213             |
| timer                            | 50                         | C_IRGRP                                 | 213             |
| CRDLY                            | 394                        | C_IROTH                                 | 213             |
| CREAD                            | 396                        | C_IRUSR                                 | 213             |
| creal()                          | 675                        | C_ISBLK                                 | 213             |
| crealf                           | 675                        | C_ISCHR                                 | 213             |
|                                  |                            | C_ISCTG                                 | 213             |



## Index

|                         |                   |                                          |                          |
|-------------------------|-------------------|------------------------------------------|--------------------------|
| C_ISDIR .....           | 213               | size_t .....                             | 325                      |
| C_ISFIFO .....          | 213               | speed_t .....                            | 393                      |
| C_ISGID .....           | 213               | tflag_t .....                            | 393                      |
| C_ISLNK .....           | 213               | VISIT .....                              | 307                      |
| C_ISREG .....           | 213               | wchar_t .....                            | 325                      |
| C_ISSOCK .....          | 213               | wctrans_t .....                          | 439                      |
| C_ISUID .....           | 213               | wctype_t .....                           | 435                      |
| C_ISVTX .....           | 213               | wint_t .....                             | 435                      |
| C_IWGRP .....           | 213               | data types                               |                          |
| C_IWOTH .....           | 213               | defined in <fenv.h> .....                | <b>228</b>               |
| C_IWUSR .....           | 213               | defined in <sys/types.h> .....           | 380                      |
| C_IXGRP .....           | 213               | date .....                               | <b>2510</b> , 3592       |
| C_IXOTH .....           | 213               | conversion specifications .....          | 2510                     |
| C_IXUSR .....           | 213               | modified conversion specifications ..... | 2511                     |
| data access .....       | 3584, 3588        | DATEMSK .....                            | <b>163</b> , 965         |
| data key creation ..... | 1584              | daylight .....                           | <b>691</b> , 2092        |
| data keywords .....     | 2972              | DAY_ .....                               | 250                      |
| data messages .....     | 473               | DBL_constants                            |                          |
| data segment .....      | 50                | defined in <float.h> .....               | <b>233</b>               |
| data structure          |                   | DBL_DIG .....                            | 234                      |
| dirent .....            | 217               | DBL_EPSILON .....                        | 235                      |
| entry .....             | 307               | DBL_MANT_DIG .....                       | 233, 621, 833            |
| group .....             | 243               | DBL_MAX .....                            | 235                      |
| lconv .....             | 267               | DBL_MAX_10_EXP .....                     | 234                      |
| msqid_ds .....          | 355               | DBL_MAX_EXP .....                        | 234, 621, 833            |
| stat .....              | 370               | DBL_MIN .....                            | 235                      |
| tms .....               | 379               | DBL_MIN_10_EXP .....                     | 234                      |
| utimbuf .....           | 432               | DBL_MIN_EXP .....                        | 234                      |
| data type .....         | <b>518</b> , 3517 | DBM .....                                | 280, 692-693             |
| ACTION .....            | 307               | DBM_ .....                               | 450                      |
| cc_t .....              | 393               | dbm .....                                | 450                      |
| DIR .....               | 217               | dbm_clearerr() .....                     | <b>692</b>               |
| div_t .....             | 338               | dbm_close .....                          | 692                      |
| ENTRY .....             | 307               | dbm_delete .....                         | 692                      |
| FILE .....              | 335               | dbm_error .....                          | 692                      |
| fpos_t .....            | 335               | dbm_fetch .....                          | 692                      |
| glob_t .....            | 241               | dbm_firstkey .....                       | 692                      |
| ldiv_t .....            | 338               | DBM_INSERT .....                         | 280, 693                 |
| lldiv_t .....           | 338               | dbm_nextkey .....                        | 692                      |
| mbstate_t .....         | 435               | dbm_open .....                           | 692                      |
| msglen_t .....          | 355               | DBM_REPLACE .....                        | 280, 693                 |
| msgqnum_t .....         | 355               | dbm_store .....                          | 692                      |
| nl_catd .....           | 292               | dc, rationale for omission .....         | 3577                     |
| nl_item .....           | 292               | dd .....                                 | <b>2517</b> , 3543, 3592 |
| pid_t .....             | 312               | DEAD_PROCESS .....                       | 433, 733-734             |
| ptrdiff_t .....         | 325               | DECIMAL_DIG .....                        | 233                      |
| regex_t .....           | 303               | default queue .....                      | 2336                     |
| regmatch_t .....        | 303               | DEFECHO .....                            | 452                      |
| regoff_t .....          | 303               | deferred batch service .....             | 50                       |
| shmatt_t .....          | 363               | deferred batch services .....            | 2324                     |
| sigset_t .....          | 312               | deferred cancelability .....             | 1580                     |
| sig_atomic_t .....      | 312               | defined types .....                      | 518, 3517                |
|                         |                   | definitions .....                        | 3318                     |

- delay process execution.....1916
- DELAYTIMER\_MAX .....254, 2010, 2064, 3601
- Delete Batch Job Request.....2333
- delta .....**2526**
- dependency ordering.....707
- descriptive name .....888
- destination .....2341
- destroying a mutex.....1592
- destroying condition variables .....1546
- destructor functions .....1583
- detaching a thread.....1566
- determinism .....3584
- device .....50
  - output.....184
- device ID.....50
- device number .....3326
- device, logical.....3332
- DEV\_BSIZE .....373
- dev\_t.....380
- df .....**2530**, 3543, 3592-3593
- diff.....**2534**, 3592
  - binary output format .....2536
  - default output format .....2536
  - directory comparison format.....2535
  - c or -C output format .....2537
  - e output format.....2537
  - f output format .....2537
  - u or -U output format .....2538
- difftime() .....**696**
- DIR.....217, 518, 654, 1704, 1706, 1746, 1768, 2050
- dircmp, rationale for omission .....3577
- direct I/O.....3326
- directive .....864, 900, 940, 949, 1975
- directory.....51, 3326
- directory commands
  - cd.....2442
  - pwd.....2983
- directory device .....3378
- directory entry .....3326
- directory entry (or link).....51
- directory files .....3378
- directory lister.....2788
- directory operations.....797
- directory protection.....95, 3345
- directory stream.....51
- directory structure.....3378
- directory, root.....3335
- dirent structure .....217, 797
- dirfd().....**697**
- dirname.....**2543**, 3591-3592
- dirname().....**699**
- DIRTYPE.....391
- dis, rationale for omission.....3577
- disarm (a timer) .....51
- disk space commands
  - df .....2530
  - du .....2546
  - ulimit .....3167
- display.....51, 3326
- display line .....51
- div().....**701**
- dlclose() .....**702**
- dLError() .....**704**
- dlopen() .....**706**
- dlsym() .....**709**
- documentation.....15, 2851
- dollar sign.....51
- domain error .....104
- dot.....52, 797, 1743, 2287, 3327
- dot-dot.....52, 797, 1743, 3327, 3334, 3351
- double-quote .....52
- double-quotes .....2246, 3546
- downshifting.....52
- dprintf .....864
- dprintf() .....**711**
- drand48().....**712**
- driver.....52
- du.....**2546**, 3543, 3592-3593
- dup.....3588
- dup() .....**715**, 3432
- dup2.....715, 3588
- dup2() .....3432
- duplicating an input file descriptor.....2261
- duplicating an output file descriptor .....2261
- duplocale() .....**717**
- DUP\_COUNT.....177
- dynamic package initialization .....1630
- d\_.....450
- D\_FMT .....250
- D\_T\_FMT .....250
- E2BIG.....220, 456
- EACCES .....220, 456
- EADDRINUSE .....220, 456
- EADDRNOTAVAIL.....220, 457
- EAFNOSUPPORT .....220, 457
- EAGAIN .....220, 457, 462
- EAI\_AGAIN .....285, 956
- EAI\_BADFLAGS .....285, 956
- EAI\_FAIL .....285, 956
- EAI\_FAMILY .....285, 956
- EAI\_MEMORY.....285, 956
- EAI\_NONAME.....285, 956
- EAI\_OVERFLOW .....285, 956
- EAI\_SERVICE .....285, 956
- EAI\_SOCKETTYPE .....285, 956

## Index

|                                              |                      |                                     |                                       |
|----------------------------------------------|----------------------|-------------------------------------|---------------------------------------|
| EAI_SYSTEM .....                             | 285, 956             | write command.....                  | 2563                                  |
| EALREADY .....                               | 220, 457             | EDEADLK .....                       | 220, 458                              |
| EBADF .....                                  | 220, 457             | EDESTADDRREQ .....                  | 220, 458                              |
| EBADMSG.....                                 | 220, 457             | edit buffer .....                   | 2573, 3215                            |
| EBUSY .....                                  | 220, 457, 3405, 3475 | edit line .....                     | 3078                                  |
| ECANCELED.....                               | 220, 457, 3404       | editors                             |                                       |
| ECHILD .....                                 | 220, 457             | ed .....                            | 2553                                  |
| ECHO .....                                   | 396                  | ex .....                            | 2573                                  |
| echo.....                                    | <b>2550</b> , 3591   | sed.....                            | 3065                                  |
| ECHOCTL .....                                | 452                  | vi .....                            | 3215                                  |
| ECHOE.....                                   | 396, 3384            | EDOM .....                          | 220, 458, 3405                        |
| ECHOK .....                                  | 396, 3384            | EDQUOT .....                        | 220, 458                              |
| ECHOKE.....                                  | 452                  | ED_FILE_MAX .....                   | 2565                                  |
| ECHONL .....                                 | 396, 3384            | ED_LINE_MAX .....                   | 2565                                  |
| ECHOPRT.....                                 | 452                  | EEXIST .....                        | 221, 458                              |
| ECONNABORTED.....                            | 220, 457             | EFAULT .....                        | 221, 458, 3403-3404                   |
| ECONNREFUSED.....                            | 220, 457             | EFBIG .....                         | 221, 458                              |
| ECONNRESET.....                              | 220, 457             | effective group ID.....             | 52, 637, 753, 986, 2227               |
| ecvt.....                                    | 3520                 | effective user ID .....             | 52, 540, 753, 1164, 2227, 3346        |
| ed .....                                     | <b>2553</b> , 3592   | EFTYPE .....                        | 3404                                  |
| addresses .....                              | 2555                 | EHOSTUNREACH.....                   | 221, 458                              |
| append command.....                          | 2557                 | EIDRM .....                         | 221, 458                              |
| change command .....                         | 2557                 | eight-bit transparency.....         | 53                                    |
| commands .....                               | 2556                 | Eighth Edition UNIX.....            | 2217, 2479                            |
| copy command .....                           | 2562                 | EILSEQ.....                         | 221, 458, 473, 3405                   |
| delete command .....                         | 2558                 | EINPROGRESS.....                    | 221, 458, 477, 3426                   |
| edit command .....                           | 2558                 | EINTR.....                          | 221, 458, 493, 1444, 3404, 3406, 3416 |
| edit without checking command .....          | 2558                 | EINVAL.....                         | 221, 458, 1431, 1444, 1589            |
| filename command.....                        | 2558                 | .....                               | 1592, 1620, 3404                      |
| global command.....                          | 2558                 | EIO.....                            | 221, 459                              |
| global non-matched command.....              | 2563                 | EISCONN .....                       | 221, 459                              |
| help command .....                           | 2559                 | EISDIR.....                         | 221, 459                              |
| help-mode command.....                       | 2559                 | ELOOP.....                          | 221, 459, 3404                        |
| insert command.....                          | 2559                 | ELSIZE .....                        | 1226                                  |
| interactive global command .....             | 2559                 | emacs, rationale for omission ..... | 3577                                  |
| interactive global not-matched command ..... | 2563                 | EMFILE .....                        | 221, 459                              |
| join command .....                           | 2560                 | EMLINK.....                         | 221, 459                              |
| line number command .....                    | 2563                 | EMPTY .....                         | 433, 734                              |
| list command.....                            | 2560                 | empty directory .....               | 53                                    |
| mark command.....                            | 2560                 | empty line .....                    | 53                                    |
| move command .....                           | 2560                 | empty string (or null string) ..... | 53                                    |
| null command.....                            | 2564                 | empty wide-character string.....    | 53                                    |
| number command.....                          | 2560                 | EMSGSIZE.....                       | 221, 459                              |
| print command .....                          | 2561                 | EMULTIHOP.....                      | 221, 459                              |
| prompt command.....                          | 2561                 | ENAMETOOLONG .....                  | 221, 459, 3404                        |
| quit command.....                            | 2561                 | encoding                            |                                       |
| quit without checking command .....          | 2561                 | character.....                      | 114                                   |
| read command .....                           | 2561                 | encoding rule .....                 | 53                                    |
| regular expressions .....                    | 2554                 | encrypt() .....                     | <b>719</b>                            |
| shell escape command.....                    | 2563                 | encryption.....                     | 22                                    |
| substitute command.....                      | 2561                 | endgrent().....                     | <b>721</b> , 3344                     |
| undo command.....                            | 2562                 | endhostent().....                   | <b>723</b>                            |
|                                              |                      | endnetent().....                    | <b>725</b>                            |

|                                    |                            |
|------------------------------------|----------------------------|
| endprotoent()                      | 727                        |
| endpwent()                         | 729, 3344                  |
| endservent()                       | 731                        |
| endutxent()                        | 733                        |
| ENETDOWN                           | 221, 459                   |
| ENETRESET                          | 221, 459                   |
| ENETUNREACH                        | 221, 459                   |
| ENFILE                             | 221, 459                   |
| ENOBUFFS                           | 221, 459                   |
| ENODATA                            | 221, 460                   |
| ENODEV                             | 221, 460                   |
| ENOENT                             | 221, 460                   |
| ENOEXEC                            | 221, 460                   |
| ENOLCK                             | 221, 460                   |
| ENOLINK                            | 221, 460                   |
| ENOMEM                             | 221, 460, 3404             |
| ENOMSG                             | 221, 460                   |
| ENOPROTOOPT                        | 221, 460                   |
| ENOSPC                             | 221, 460                   |
| ENOSR                              | 221, 460                   |
| ENOSTR                             | 221, 460                   |
| ENOSYS                             | 222, 460, 3404, 3429       |
| ENOTCONN                           | 222, 460                   |
| ENOTDIR                            | 222, 460                   |
| ENOTEMPTY                          | 222, 461                   |
| ENOTRECOVERABLE                    | 222, 1549, 1600            |
| ENOTSOCK                           | 222, 461                   |
| ENOTSUP                            | 222, 461, 3404             |
| ENOTTY                             | 222, 461, 3379, 3404       |
| entire regular expression          | 53, 167                    |
| ENTRY                              | 1059                       |
| env                                | 2569, 3591, 3593           |
| environ                            | 736, 753                   |
| environment access                 | 3584, 3589                 |
| environment variable               | 3368                       |
| definition                         | 3368                       |
| environment variables              |                            |
| internationalization               | 160                        |
| envp                               | 753                        |
| ENXIO                              | 222, 461                   |
| EOF                                | 334                        |
| EOPNOTSUPP                         | 222, 461                   |
| E_OVERFLOW                         | 222, 461, 3405             |
| EOWNERDEAD                         | 222, 1549, 1600            |
| EPERM                              | 222, 461, 1600, 2488, 3484 |
| EPIPE                              | 222, 461, 3405             |
| epoch                              | 53                         |
| Epoch                              | 3327, 3352, 3448           |
| EPROTO                             | 222, 461                   |
| EPROTONOSUPPORT                    | 222, 461                   |
| EPROTOTYPE                         | 222, 461                   |
| equivalence class                  | 54                         |
| era                                | 54                         |
| ERA                                | 250                        |
| erand48                            | 712                        |
| erand48()                          | 737                        |
| ERANGE                             | 222, 461, 3405             |
| ERASE                              | 3382                       |
| ERA_D_FMT                          | 250                        |
| ERA_D_T_FMT                        | 250                        |
| ERA_T_FMT                          | 250                        |
| ERE                                | 3376                       |
| alternation                        | 3377                       |
| bracket expression                 | 3377                       |
| expression anchoring               | 3377                       |
| grammar                            | 3378                       |
| grammar lexical conventions        | 3377                       |
| matching a collating element       | 3376                       |
| matching a single character        | 3376                       |
| matching multiple characters       | 3377                       |
| ordinary character                 | 3376                       |
| periods                            | 3376                       |
| precedence                         | 3377                       |
| special character                  | 3376                       |
| erf()                              | 738                        |
| erfc()                             | 740                        |
| erfcf                              | 740                        |
| erfcl                              | 740                        |
| erff                               | 738                        |
| erff()                             | 742                        |
| erfl                               | 738, 742                   |
| EROFS                              | 222, 461, 3405             |
| errno                              | 743, 3403                  |
| per-thread                         | 3405                       |
| error conditions                   | 3353                       |
| mathematical functions             | 104                        |
| error descriptions                 | 956                        |
| error handling                     | 3568                       |
| error numbers                      | 456, 3403, 3406            |
| additional                         | 462                        |
| errors                             | 3559                       |
| Error_Path                         | 2327                       |
| escape character                   | 3546                       |
| escape character (backslash)       | 2246                       |
| escape sequences                   |                            |
| awk                                | 2379                       |
| gencat                             | 2690                       |
| lex                                | 2756                       |
| ESPIPE                             | 222, 461                   |
| ESRCH                              | 222, 462                   |
| EST5EDT                            | 2092                       |
| establish the locale               | 2231                       |
| establishing cancellation handlers | 1539                       |
| ESTALE                             | 222, 462                   |
| ETIME                              | 222, 462                   |

## Index

|                                          |                                          |
|------------------------------------------|------------------------------------------|
| ETIMEDOUT.....                           | 222, 462                                 |
| ETXTBSY.....                             | 222, 462                                 |
| eval .....                               | <b>2289</b>                              |
| event management.....                    | 54                                       |
| EWOULDBLOCK.....                         | 222, 462                                 |
| ex .....                                 | <b>2573</b> , 3591-3592                  |
| <backslash>.....                         | 2585                                     |
| <control>-D command .....                | 2607                                     |
| <newline>.....                           | 2585                                     |
| abbreviate command.....                  | 2588                                     |
| addressing .....                         | 2579                                     |
| adjust window command .....              | 2605                                     |
| append command.....                      | 2588                                     |
| args command.....                        | 2589                                     |
| autoindent option.....                   | 2609                                     |
| autoprint option.....                    | 2610                                     |
| autowrite option.....                    | 2610                                     |
| beautify option.....                     | 2610                                     |
| change command .....                     | 2589                                     |
| chdir command.....                       | 2589                                     |
| command descriptions .....               | 2586                                     |
| copy command .....                       | 2589                                     |
| delete command .....                     | 2590                                     |
| directory option .....                   | 2610                                     |
| edcompatible option .....                | 2610                                     |
| edit command .....                       | 2590                                     |
| edit options.....                        | 2609                                     |
| errorbells option.....                   | 2611                                     |
| escape command .....                     | 2606                                     |
| execute command.....                     | 2608                                     |
| excrc command .....                      | 2611                                     |
| file command .....                       | 2591                                     |
| global command .....                     | 2591                                     |
| ignorecase option.....                   | 2611                                     |
| initialization .....                     | 2576                                     |
| input editing.....                       | 2584                                     |
| insert command .....                     | 2592                                     |
| join command .....                       | 2592                                     |
| list command.....                        | 2593, 2611                               |
| magic command.....                       | 2611                                     |
| map command .....                        | 2593                                     |
| mark command.....                        | 2594                                     |
| mesg command.....                        | 2611                                     |
| move command .....                       | 2595                                     |
| next command .....                       | 2596                                     |
| number command.....                      | 2596                                     |
| number option .....                      | 2612                                     |
| open command .....                       | 2596                                     |
| paragraphs option.....                   | 2612                                     |
| preserve.....                            | 2573                                     |
| preserve command.....                    | 2597                                     |
| print command .....                      | 2597                                     |
| prompt command.....                      | 2612                                     |
| put command .....                        | 2598                                     |
| quit command .....                       | 2598                                     |
| read command .....                       | 2598                                     |
| readonly command .....                   | 2612                                     |
| recover command.....                     | 2599                                     |
| redraw command .....                     | 2612                                     |
| regular expressions .....                | 2608                                     |
| remap command.....                       | 2613                                     |
| replacement strings .....                | 2608                                     |
| report command .....                     | 2613                                     |
| rewind command .....                     | 2599                                     |
| scroll command.....                      | 2584, 2613                               |
| sections command.....                    | 2613                                     |
| set command .....                        | 2599                                     |
| shell command.....                       | 2600                                     |
| shell option.....                        | 2613                                     |
| shift left command .....                 | 2607                                     |
| shift right command.....                 | 2607                                     |
| shiftwidth option.....                   | 2614                                     |
| showmatch option.....                    | 2614                                     |
| showmode command .....                   | 2614                                     |
| slowopen command.....                    | 2614                                     |
| source command.....                      | 2600                                     |
| substitute command.....                  | 2600                                     |
| suspend command .....                    | 2601                                     |
| tabstop option .....                     | 2614                                     |
| tag command .....                        | 2601                                     |
| taglength option .....                   | 2614                                     |
| tags command .....                       | 2614                                     |
| term command.....                        | 2615                                     |
| terse command.....                       | 2615                                     |
| unabbrev command .....                   | 2602                                     |
| undo command.....                        | 2602                                     |
| unmap command .....                      | 2602                                     |
| version command.....                     | 2603                                     |
| visual command .....                     | 2603                                     |
| warn command.....                        | 2615                                     |
| window command .....                     | 2615                                     |
| wrapmargin option .....                  | 2615                                     |
| wrapscan option .....                    | 2616                                     |
| write command.....                       | 2603                                     |
| write line number command .....          | 2607                                     |
| writeany option .....                    | 2616                                     |
| xit command .....                        | 2604                                     |
| yank command .....                       | 2605                                     |
| examine and change blocked signals ..... | 1664                                     |
| examine and change signal action .....   | 1874                                     |
| EXDEV .....                              | 222, 462                                 |
| exec .....                               | <b>745</b> , 2291, 2899, 3511            |
| of shell scripts .....                   | 752                                      |
| exec family.....                         | 540, 653, 784, 830, 856, 1493            |
| .....                                    | 1830, 2135, 2244, 2479, 2883, 3287, 3329 |

|                                   |                                           |
|-----------------------------------|-------------------------------------------|
| .....                             | 3430, 3488, 3539, 3544, 3566, 3587        |
| execl .....                       | 745                                       |
| execle .....                      | 745                                       |
| execlp .....                      | 745                                       |
| executable file .....             | 54                                        |
| execute .....                     | 54                                        |
| execute a file .....              | 752                                       |
| execution time .....              | 49, 54                                    |
| measurement .....                 | 98                                        |
| monitoring .....                  | 54                                        |
| execution time monitoring .....   | 485                                       |
| Execution Time Monitoring .....   | 3453                                      |
| execution time, measurement ..... | 3348                                      |
| Execution_Time .....              | 2328                                      |
| execv .....                       | 745                                       |
| execve .....                      | 745                                       |
| execvp .....                      | 745                                       |
| EXINIT .....                      | 2573                                      |
| exit .....                        | <b>2293</b>                               |
| exit status .....                 | 3559                                      |
| exit status and errors .....      | 2262                                      |
| exit status for commands .....    | 2262                                      |
| exit() .....                      | <b>757</b> , 3414, 3587                   |
| EXIT_FAILURE .....                | 338, 523, 757, 3590                       |
| EXIT_SUCCESS .....                | 338, 523, 526, 757, 3590                  |
| exp() .....                       | <b>758</b>                                |
| exp2() .....                      | <b>760</b>                                |
| exp2f .....                       | 760                                       |
| exp2l .....                       | 760                                       |
| expand .....                      | 55, 2642, 3592                            |
| expf .....                        | 758                                       |
| expl .....                        | 758                                       |
| expm1() .....                     | <b>762</b>                                |
| expm1f .....                      | 762                                       |
| expm1l .....                      | 762                                       |
| export .....                      | <b>2295</b>                               |
| expr .....                        | <b>2645</b> , 3591-3592                   |
| matching expression .....         | 2647                                      |
| string operand .....              | 2647                                      |
| expression argument .....         | 2382                                      |
| expression list .....             | 2382                                      |
| EXPR_NEST_MAX .....               | 258, 2010, 2234, 3538                     |
| EXTA .....                        | 452                                       |
| EXTB .....                        | 452                                       |
| extended regular expression ..... | 55, 174                                   |
| .....                             | 2371, 2379, 2486, 2671, 2711, 2755        |
| .....                             | 2878, 2935, 3048, 3285, 3376              |
| extended security controls .....  | 55, 95, 3346                              |
| extension                         |                                           |
| CX .....                          | 7                                         |
| OH .....                          | 9                                         |
| XSI .....                         | 12                                        |
| extensions to setlocale .....     | 1824                                      |
| F-LOCK .....                      | <b>421</b>                                |
| fabs() .....                      | <b>764</b>                                |
| fabsf .....                       | 764                                       |
| fabsl .....                       | 764                                       |
| faccessat .....                   | 539                                       |
| faccessat() .....                 | <b>766</b>                                |
| false .....                       | 2244, 2264, 2650, 3545, 3591              |
| fattach() .....                   | <b>767</b>                                |
| fc .....                          | 2244, 2264, 2652, 3545, 3591              |
| fchdir() .....                    | <b>770</b>                                |
| fchmod .....                      | 3588                                      |
| fchmod() .....                    | <b>771</b>                                |
| fchmodat .....                    | 631                                       |
| fchmodat() .....                  | <b>773</b>                                |
| fchown() .....                    | <b>774</b>                                |
| fchownat .....                    | 635                                       |
| fchownat() .....                  | <b>776</b>                                |
| fclose .....                      | 3588                                      |
| fclose() .....                    | <b>777</b>                                |
| fcntl .....                       | 3588                                      |
| fcntl() .....                     | <b>779</b> , 3379, 3404, 3432, 3435, 3518 |
| fcntl() locks .....               | 3490                                      |
| fcvt .....                        | 3520                                      |
| FD .....                          | 7                                         |
| fdatasync() .....                 | <b>787</b>                                |
| fdetach() .....                   | <b>788</b>                                |
| fdim() .....                      | <b>790</b>                                |
| fdimf .....                       | 790                                       |
| fdiml .....                       | 790                                       |
| fdopen() .....                    | <b>792</b> , 3432                         |
| fdopendir() .....                 | <b>795</b>                                |
| fds .....                         | 450                                       |
| FD_ .....                         | 450                                       |
| fd .....                          | 450                                       |
| FD_CLOEXEC .....                  | 224, 470, 615, 746, 779, 797              |
| .....                             | 1069, 1344, 1365, 1386, 1393, 1853, 3435  |
| FD_CLR .....                      | 1486                                      |
| FD_CLR() .....                    | <b>522</b>                                |
| FD_ISSET .....                    | 522, 1486                                 |
| fd_set .....                      | <b>359</b> , 377                          |
| FD_SET .....                      | 522, 1486                                 |
| FD_SETSIZE .....                  | 359                                       |
| FD_ZERO .....                     | 522, 1486                                 |
| feature test macro .....          | 55, 448, 960, 3398-3399, 3518             |
| _POSIX_C_SOURCE .....             | 448                                       |
| _XOPEN_SOURCE .....               | 448                                       |
| feclearexcept() .....             | <b>799</b>                                |
| fegetenv() .....                  | <b>800</b>                                |
| fegetexceptflag() .....           | <b>801</b>                                |
| fegetround() .....                | <b>802</b>                                |
| feholdexcept() .....              | <b>804</b>                                |
| fenv_t .....                      | 228                                       |

## Index

- feof().....**805**
- feraiseexcept() .....**806**
- ferror() .....**807**
- fesetenv .....800
- fesetenv().....**808**
- fesetexceptflag.....801
- fesetexceptflag() .....**809**
- fesetround.....802
- fesetround() .....**810**
- fetestexcept().....**811**
- feupdateenv() .....**813**
- fexcept\_t.....228
- fexecve.....745, 815
- FE\_ .....452
- FE\_ constants
  - defined in <fenv.h> .....**228**
- FE\_ALL\_EXCEPT .....228
- FE\_DFL\_ENV .....229
- FE\_DIVBYZERO .....228
- FE\_DOWNWARD .....228
- FE\_INEXACT .....228
- FE\_INVALID .....228
- FE\_OVERFLOW .....228
- FE\_TONEAREST .....228
- FE\_TOWARDZERO .....228
- FE\_UNDERFLOW .....228
- FE\_UPWARD .....228
- FFDLY.....395
- fflush() .....**816**
- FFn .....395
- ffs() .....**819**
- fg.....2244, 2264, 2658, 3545, 3591
- fgetc .....3588
- fgetc().....**820**
- fgetpos() .....**822**, 3518
- fgets() .....**824**
- fgetwc().....**826**
- fgetws().....**828**
- field.....55
- field splitting.....2258, 3557
- FIFO.....55, 1259-1260, 1262, 1349
  - .....2216, 3327, 3334, 3441
- FIFO special file .....55, 3327
- FIFO special files .....2861
- FIFOTYPE.....391
- file .....56
- FILE .....335, 435, 518, 640, 777, 792, 805
  - .....807, 816, 820, 822, 824, 826, 828, 830-831
  - .....849, 864, 877, 879, 881, 883, 885, 894, 900
  - .....907, 910, 925, 935, 939-940, 947, 949
  - .....958-959, 1050, 1361, 1371, 1673-1674, 1686
  - .....1745, 1810, 1851, 1934, 2069, 2100, 2102
  - .....2118, 2120, 2122, 2124, 2126, 2128
- file .....**2660**, 3327
  - locking.....784
- file access permissions .....96, 2228, 3346
- file accessibility .....540
- file characteristics
  - data structure .....372
  - header.....372
- file classes .....3327
- file comparisons
  - cmp .....2469
  - comm.....2472
  - diff.....2534
  - uniq.....3187
- file contents .....2230
- file control.....784
- file conversion
  - cut .....2503
  - dd .....2517
  - expand.....2642
  - fold .....2677
  - head .....2719
  - join .....2741
  - od .....2901
  - paste .....2909
  - patch.....2913
  - sort .....3093
  - strings.....3102
  - tail .....3120
  - tr .....3152
  - tsort.....3160
  - unexpand .....3181
  - uniq.....3187
  - uudecode .....3197
  - uuencode .....3200
- file creation .....2228, 3536
- file description .....56
- file descriptor .....56, 2227, 2259
- file descriptors .....3416
- file format notation.....3354
- file group class .....56
- file hierarchy .....96, 3346
- file hierarchy manipulation .....3585, 3592
- file mode .....56
- file mode bits.....56
- file mode creation mask.....2227
- FILE object .....469
- file offset.....57
- file other class.....57
- file owner class.....57
- file permission bits .....57, 541
- file permission commands
  - chgrp .....2450

|                                     |                           |                             |                    |
|-------------------------------------|---------------------------|-----------------------------|--------------------|
| chmod.....                          | 2453                      | tr.....                     | 3152               |
| chown.....                          | 2460                      | uncompress.....             | 3178               |
| umask.....                          | 3169                      | unexpand.....               | 3181               |
| file permissions.....               | 540, 860, 918, 3346, 3381 | zcat.....                   | 3306               |
| file position indicator.....        | 469                       | FIND.....                   | 1059               |
| file read.....                      | 2228, 3536                | find.....                   | <b>2668</b> , 3592 |
| file removal.....                   | 2230, 3536                | find string token.....      | 1990               |
| file searching                      |                           | FIPS.....                   | 16                 |
| grep.....                           | 2711                      | FIPS requirements.....      | 3315               |
| file serial number.....             | 57                        | first open (of a file)..... | 58                 |
| file size, arbitrary.....           | 3543                      | flockfile().....            | <b>831</b> , 3406  |
| file system.....                    | 57, 3327                  | floor().....                | <b>833</b>         |
| file system, mounted.....           | 3333                      | floorf.....                 | 833                |
| file system, root.....              | 3335                      | floorl.....                 | 833                |
| file time values.....               | 2230                      | flow control.....           | 58                 |
| file times update.....              | 97, 3348                  | FLT_ constants              |                    |
| file tree commands                  |                           | defined in <float.h>.....   | <b>233</b>         |
| diff.....                           | 2534                      | FLT_DIG.....                | 234                |
| find.....                           | 2668                      | FLT_EPSILON.....            | 235                |
| ls.....                             | 2788                      | FLT_EVAL_METHOD.....        | 232                |
| mkdir.....                          | 2858                      | FLT_MANT_DIG.....           | 233                |
| rmdir.....                          | 3054                      | FLT_MAX.....                | 235                |
| file type.....                      | 57                        | FLT_MAX_10_EXP.....         | 234                |
| file write.....                     | 2228, 3536                | FLT_MAX_EXP.....            | 234                |
| file, passwd.....                   | 3334                      | FLT_MIN.....                | 235                |
| filename.....                       | 56, 3327, 3346            | FLT_MIN_10_EXP.....         | 234                |
| filename portability.....           | 97, 3348                  | FLT_MIN_EXP.....            | 234                |
| filenames.....                      | 97                        | FLT_RADIX.....              | 233, 1216          |
| FILENAME_MAX.....                   | 334                       | FLT_ROUNDS.....             | 232, 835           |
| fileno().....                       | <b>830</b> , 3334         | FLUSH.....                  | 450                |
| FILESIZEBITS.....                   | 256, 858                  | FLUSHO.....                 | 452                |
| filter.....                         | 58                        | FLUSHR.....                 | 347, 1089          |
| filtering of trace event types..... | 3514                      | FLUSHRW.....                | 347, 1089          |
| filtering trace event types.....    | 3514                      | FLUSHW.....                 | 347, 1089          |
| filters                             |                           | fma().....                  | <b>835</b>         |
| asa.....                            | 2359                      | fmaf.....                   | 835                |
| awk.....                            | 2371                      | fmal.....                   | 835                |
| compress.....                       | 2480                      | fmax().....                 | <b>837</b>         |
| dd.....                             | 2517                      | fmaxf.....                  | 837                |
| expand.....                         | 2642                      | fmaxl.....                  | 837                |
| fold.....                           | 2677                      | fmemopen().....             | <b>838</b>         |
| head.....                           | 2719                      | fmin().....                 | <b>841</b>         |
| iconv.....                          | 2722                      | fminf.....                  | 841                |
| more.....                           | 2864                      | fminl.....                  | 841                |
| nl.....                             | 2889                      | FMNAMESZ.....               | 346-347, 1088      |
| paste.....                          | 2909                      | fmod().....                 | <b>842</b>         |
| pax.....                            | 2925                      | fmodf.....                  | 842                |
| pr.....                             | 2961                      | fmodl.....                  | 842                |
| read.....                           | 3040                      | fmtmsg().....               | <b>844</b>         |
| sed.....                            | 3065                      | fnmatch().....              | <b>847</b> , 3593  |
| tail.....                           | 3120                      | FNM_.....                   | 450                |
| tee.....                            | 3128                      | FNM_ constants              |                    |
|                                     |                           | in <fnmatch.h>.....         | <b>238</b>         |



## Index

|                                   |                                     |                                  |                              |
|-----------------------------------|-------------------------------------|----------------------------------|------------------------------|
| FNM_NOESCAPE.....                 | 238, 847                            | frexp() .....                    | <b>898</b>                   |
| FNM_NOMATCH.....                  | 238, 847                            | frexpf .....                     | 898                          |
| FNM_PATHNAME .....                | 238, 847                            | frexpl .....                     | 898                          |
| FNM_PERIOD .....                  | 238, 847                            | fsblkcnt_t .....                 | 380                          |
| fold.....                         | <b>2677</b> , 3592                  | FSC.....                         | <b>8</b>                     |
| fopen.....                        | 3588                                | fscanf() .....                   | <b>900</b>                   |
| fopen() .....                     | <b>849</b> , 3328, 3543             | fseek .....                      | 3588                         |
| FOPEN_MAX.....                    | 255, 334, 792, 839, 850, 2069       | fseek() .....                    | <b>907</b>                   |
| for loop.....                     | 2268, 3564                          | fseeko.....                      | 907                          |
| foreground .....                  | 1829, 3329-3332, 3380-3381          | fseeko() .....                   | 3518                         |
| foreground job .....              | 58                                  | fsetpos .....                    | 3588                         |
| foreground process.....           | 58                                  | fsetpos() .....                  | <b>910</b> , 3518            |
| foreground process group.....     | 58                                  | fsfilcnt_t .....                 | 380                          |
| foreground process group ID ..... | 58                                  | fstat.....                       | 3587-3588                    |
| fork handler.....                 | 1494                                | fstat().....                     | <b>912</b>                   |
| fork() .....                      | <b>853</b> , 3329, 3380, 3421, 3430 | fstatat().....                   | <b>915</b>                   |
| .....                             | 3432, 3511, 3518, 3522, 3587        | fstatvfs() .....                 | <b>920</b>                   |
| forkall .....                     | 856                                 | fsync() .....                    | <b>923</b> , 3425            |
| form-feed character .....         | 59                                  | ftell().....                     | <b>925</b>                   |
| format of entries .....           | <b>205</b> , 445                    | ftello .....                     | 925                          |
| fort77 .....                      | <b>2680</b> , 3593                  | ftello() .....                   | 3518                         |
| external symbols.....             | 2683                                | ftime.....                       | 3521                         |
| standard libraries.....           | 2682                                | ftok().....                      | <b>927</b>                   |
| fpathconf.....                    | 3588                                | ftruncate .....                  | 3588                         |
| fpathconf() .....                 | <b>858</b> , 3587                   | ftruncate() .....                | <b>929</b> , 3431-3432, 3434 |
| fpclassify().....                 | <b>863</b>                          | ftrylockfile .....               | 831                          |
| FPE .....                         | 450, 452                            | ftrylockfile() .....             | <b>931</b>                   |
| FPE_FLTDIV.....                   | 316                                 | FTW .....                        | 239, 450, 1334-1335          |
| FPE_FLTINV .....                  | 316                                 | ftw().....                       | <b>932</b> , 3543            |
| FPE_FLTOVF .....                  | 316                                 | FTW_constants                    |                              |
| FPE_FLTRES.....                   | 316                                 | in <ftw.h>.....                  | <b>239</b>                   |
| FPE_FLTSUB .....                  | 316                                 | FTW_CHDIR.....                   | 239, 1334                    |
| FPE_FLTUND .....                  | 316                                 | FTW_D .....                      | 239, 932, 1334               |
| FPE_INTDIV .....                  | 316                                 | FTW_DEPTH .....                  | 239, 1334                    |
| FPE_INTOVF .....                  | 316                                 | FTW_DNR.....                     | 239, 932, 1334-1335          |
| fprintf().....                    | <b>864</b>                          | FTW_DP.....                      | 239, 1334                    |
| fputc.....                        | 3588                                | FTW_F.....                       | 239, 932, 1334               |
| fputc() .....                     | <b>877</b>                          | FTW_MOUNT .....                  | 239, 1334                    |
| fputs() .....                     | <b>879</b>                          | FTW_NS.....                      | 239, 932, 1335               |
| fputwc().....                     | <b>881</b>                          | FTW_PHYS.....                    | 239, 1334                    |
| fputws().....                     | <b>883</b>                          | FTW_SL.....                      | 239, 932, 1334               |
| FP_ILOGB0.....                    | 1075                                | FTW_SLN .....                    | 239, 1334                    |
| FP_ILOGBNAN .....                 | 1075                                | fully-qualified domain name..... | 1000                         |
| FQDN.....                         | 1000                                | function definition command..... | 2270, 3565                   |
| FR.....                           | 7                                   | function identifiers .....       | 2414                         |
| frac_digits .....                 | 140                                 | functions .....                  | <b>447</b>                   |
| fread.....                        | 3588                                | implementation.....              | 447                          |
| fread() .....                     | <b>885</b>                          | implementation of .....          | 3397                         |
| free().....                       | <b>887</b> , 3415, 3487             | use.....                         | 447                          |
| freeaddrinfo() .....              | <b>888</b>                          | use of .....                     | 3397                         |
| freelocale().....                 | <b>892</b>                          | funlockfile .....                | 831                          |
| freopen().....                    | <b>894</b>                          | funlockfile() .....              | <b>935</b>                   |
|                                   |                                     | fuser .....                      | <b>2686</b>                  |

|                                     |                    |                 |                              |
|-------------------------------------|--------------------|-----------------|------------------------------|
| futimens()                          | 936                | getconf         | 2701, 3537, 3587, 3593       |
| fwide()                             | 939                | getcontext      | 3521                         |
| fwprintf()                          | 940                | getcwd()        | 963                          |
| fwrite                              | 3588               | getc_unlocked() | 959                          |
| fwrite()                            | 947                | getdate()       | 965                          |
| fwscanf()                           | 949                | getdate_err     | 965                          |
| f_                                  | 451                | getdelim()      | 970                          |
| F_                                  | 452                | getegid         | 3587                         |
| F_DUPFD                             | 224, 779, 781-782  | getegid()       | 972                          |
| F_GETFD                             | 224, 779, 781, 784 | getenv          | 753                          |
| F_GETFL                             | 224, 779, 781, 784 | getenv()        | 973                          |
| F_GETLK                             | 224, 780-782       | geteuid         | 3587                         |
| F_GETOWN                            | 224, 779, 781      | geteuid()       | 976                          |
| F_LOCK                              | 1205               | getgid          | 3587                         |
| F_OK                                | 419                | getgid()        | 977                          |
| F_RDLCK                             | 224, 782           | getgrent        | 721                          |
| F_SETFD                             | 224, 779, 781, 784 | getgrent()      | 978, 3344                    |
| F_SETFL                             | 224, 779, 781, 784 | getgrgid        | 3587                         |
| F_SETLK                             | 224, 780-782       | getgrgid()      | 979, 3344, 3479              |
| F_SETLKW                            | 224, 491, 780-782  | getgrgid_r      | 979                          |
| F_SETOWN                            | 224, 779, 781      | getgrnam        | 3587                         |
| F_TEST                              | 421, 1205          | getgrnam()      | 982, 3344, 3348, 3479        |
| F_TLOCK                             | 421, 1205          | getgrnam_r      | 982                          |
| F_ULOCK                             | 421, 1205          | getgroups()     | 985, 3336                    |
| F_UNLCK                             | 224, 780-781       | gethostbyaddr   | 3521                         |
| F_WRLCK                             | 224, 782           | gethostbyname   | 3521                         |
| g-file                              | 2526               | gethostent      | 723                          |
| gai_strerror()                      | 956                | gethostent()    | 987                          |
| gcvt                                | 3520               | gethostid()     | 988                          |
| gencat                              | 2689, 3592         | gethostname()   | 989                          |
| escape sequences                    | 2690               | getitimer()     | 990                          |
| general terminal interface          | 3379               | getline         | 970                          |
| generated file                      | 2526               | getline()       | 992                          |
| generation-version attribute        | 513, 1425          | getlogin        | 3587                         |
| get                                 | 2693               | getlogin()      | 993                          |
| get configurable pathname variables | 860                | getlogin_r      | 993                          |
| get configurable system variables   | 2013               | getmsg()        | 996                          |
| get file status                     | 918                | getnameinfo()   | 1000                         |
| get process times                   | 2067               | GETNCNT         | 361, 1791-1792               |
| get supplementary group IDs         | 986                | getnetbyaddr    | 725                          |
| get system time                     | 2057               | getnetbyaddr()  | 1003                         |
| get thread ID                       | 1654               | getnetbyname    | 725, 1003                    |
| get user name                       | 994                | getnetent       | 725, 1003                    |
| getaddrinfo                         | 888                | getopt()        | 1004, 3387, 3592-3593        |
| getaddrinfo()                       | 957                | getopts         | 2244, 2264, 2706, 3545, 3593 |
| GETALL                              | 361, 1791          | getpeername()   | 1009                         |
| getc                                | 3588               | getpgid()       | 1011                         |
| getc()                              | 958, 3478          | getpgrp()       | 1012, 3331                   |
| getch                               | 3588               | GETPID          | 361, 1791-1792               |
| getchar()                           | 961                | getpid          | 3587                         |
| getchar_unlocked                    | 959                | getpid()        | 1013, 3416                   |
| getchar_unlocked()                  | 962                | getpmsg         | 996                          |
|                                     |                    | getpmsg()       | 1014                         |

## Index

|                            |                                          |
|----------------------------|------------------------------------------|
| getppid                    | 3587                                     |
| getppid()                  | <b>1015</b>                              |
| getpriority()              | <b>1016</b> , 3442                       |
| getprotent                 | 1019                                     |
| getprotobyname             | 727                                      |
| getprotobyname()           | <b>1019</b>                              |
| getprotobynumber           | 727, 1019                                |
| getprotoent                | 727                                      |
| getpwent                   | 729                                      |
| getpwent()                 | <b>1020</b> , 3344                       |
| getpwnam                   | 3587                                     |
| getpwnam()                 | <b>1021</b> , 3344, 3348, 3479           |
| getpwnam_r                 | 1021                                     |
| getpwuid                   | 3587                                     |
| getpwuid()                 | <b>1025</b> , 3344, 3479                 |
| getpwuid_r                 | 1025                                     |
| getrlimit()                | <b>1029</b> , 3544                       |
| getrusage()                | <b>1032</b> , 3455                       |
| gets()                     | <b>1034</b>                              |
| getservbyname              | 731                                      |
| getservbyname()            | <b>1036</b>                              |
| getservbyport              | 731, 1036                                |
| getservent                 | 731, 1036                                |
| getsid()                   | <b>1037</b>                              |
| getsockname()              | <b>1038</b>                              |
| getsockopt()               | <b>1040</b>                              |
| getsubopt()                | <b>1043</b>                              |
| gettimeofday()             | <b>1047</b>                              |
| getty                      | 3381                                     |
| getuid                     | 3587                                     |
| getuid()                   | <b>1048</b> , 3416, 3519                 |
| getutxent                  | 733                                      |
| getutxent()                | <b>1049</b>                              |
| getutxid                   | 733, 1049                                |
| getutxline                 | 733, 1049                                |
| GETVAL                     | 361, 1791-1792                           |
| getwc()                    | <b>1050</b>                              |
| getwchar()                 | <b>1051</b>                              |
| getwd                      | 3521                                     |
| GETZCNT                    | 361, 1791-1792                           |
| gid_t                      | 380, 3344                                |
| glob()                     | <b>1052</b> , 3593                       |
| global storage class       | 2417                                     |
| globfree                   | 1052                                     |
| globfree()                 | 3593                                     |
| GLOB_                      | 450                                      |
| GLOB_constants             |                                          |
| defined in <glob.h>        | <b>241</b>                               |
| error returns of glob      | 1054                                     |
| used in glob               | 1052                                     |
| GLOB_ABORTED               | 241, 1054                                |
| GLOB_APPEND                | 241, 1052-1053                           |
| GLOB_DOOFFS                | 241, 1052-1053                           |
| GLOB_ERR                   | 241, 1052, 1054                          |
| GLOB_MARK                  | 241, 1052                                |
| GLOB_NOCHECK               | 241, 1053-1054                           |
| GLOB_NOESCAPE              | 241, 1053                                |
| GLOB_NOMATCH               | 241, 1054                                |
| GLOB_NOSORT                | 241, 1053                                |
| GLOB_NOSPACE               | 241, 1054                                |
| gl_                        | 450                                      |
| GMT0                       | 2092                                     |
| gmtime()                   | <b>1056</b> , 3352                       |
| gmtime_r                   | 1056                                     |
| GNU make                   | 2846                                     |
| grammar                    |                                          |
| locale                     | 151                                      |
| regular expression         | 177                                      |
| grammar conventions        | 3540                                     |
| grantpt()                  | <b>1058</b>                              |
| graphic character          | 59                                       |
| grep                       | <b>2711</b> , 3592                       |
| group database             | 59, 3328                                 |
| group database access      | 3345                                     |
| group file                 | 3328                                     |
| group ID                   | 59                                       |
| group name                 | 59                                       |
| grouping commands          | 2268, 3564                               |
| HALT                       | 845                                      |
| hard limit                 | 59                                       |
| hard link                  | 59                                       |
| hash                       | <b>2716</b>                              |
| hcreate()                  | <b>1059</b>                              |
| hdestroy                   | 1059                                     |
| head                       | <b>2719</b> , 3592                       |
| headers                    | <b>205</b> , 3388                        |
| here-document              | 2260, 3559                               |
| high resolution sleep      | 1325                                     |
| historical implementations | 3328                                     |
| history command            |                                          |
| fc                         | 2652                                     |
| Hold Batch Job Request     | 2334                                     |
| Hold_Types                 | 2328                                     |
| HOME                       | <b>163</b> , 2589, 3315                  |
| home directory             | 60                                       |
| host byte order            | 60, 98, 3348                             |
| host name                  | 888                                      |
| hosted implementation      | 3328                                     |
| HOST_NAME_MAX              | 254                                      |
| hsearch                    | 1059                                     |
| htonl()                    | <b>1062</b>                              |
| htons                      | 1062                                     |
| HUGE_VAL                   | 271, 581, 621, 670, 758, 760             |
| .....                      | 762, 790, 833, 1063, 1173, 1178, 1208    |
| .....                      | 1212, 1214, 1327, 1332, 1481, 1747, 1752 |

|                               |                                    |                                        |               |
|-------------------------------|------------------------------------|----------------------------------------|---------------|
| .....                         | 1754, 1913, 1985, 2023, 2054, 2173 | imaxdiv().....                         | <b>1078</b>   |
| HUGE_VALF.....                | 271, 833, 1481, 1747, 1985, 2054   | implementation.....                    | 3328          |
| HUGE_VALL.....                | 271, 833, 1481, 1747, 1985, 2054   | implementation, historical.....        | 3328          |
| hunk.....                     | 2915                               | implementation, hosted.....            | 3328          |
| HUPCL.....                    | 396                                | implementation, native.....            | 3333          |
| hypot().....                  | <b>1063</b>                        | implementation, specific.....          | 3328          |
| hypotf.....                   | 1063                               | implementation-defined.....            | 5, 3316-3317  |
| hypotl.....                   | 1063                               | implementation-defined, rationale..... | 3316          |
| h.....                        | 450                                | IMPLINK_.....                          | 452           |
| h_errno.....                  | 3521                               | in6.....                               | 450           |
| I.....                        | 210                                | IN6.....                               | 452           |
| ICANON.....                   | 396, 3382, 3384                    | IN6_IS_ADDR_LINKLOCAL.....             | 289           |
| iconv.....                    | <b>2722</b>                        | IN6_IS_ADDR_LOOPBACK.....              | 289           |
| iconv().....                  | <b>1065</b> , 3592                 | IN6_IS_ADDR_MC_GLOBAL.....             | 289           |
| iconv_close().....            | <b>1068</b> , 3592                 | IN6_IS_ADDR_MC_LINKLOCAL.....          | 289           |
| iconv_open().....             | <b>1069</b> , 3592                 | IN6_IS_ADDR_MC_NODELOCAL.....          | 289           |
| ICRNL.....                    | 394                                | IN6_IS_ADDR_MC_ORGLOCAL.....           | 289           |
| ic.....                       | 450                                | IN6_IS_ADDR_MC_SITELOCAL.....          | 289           |
| id.....                       | <b>2725</b> , 3593                 | IN6_IS_ADDR_MULTICAST.....             | 289           |
| idtype_t.....                 | <b>387</b>                         | IN6_IS_ADDR_SITELOCAL.....             | 289           |
| id_t.....                     | 380                                | IN6_IS_ADDR_UNSPECIFIED.....           | 289           |
| IEEE Std 754-1985.....        | 445                                | IN6_IS_ADDR_V4COMPAT.....              | 289           |
| IEEE Std 854-1987.....        | 445                                | IN6_IS_ADDR_V4MAPPED.....              | 289           |
| IEXTEN.....                   | 396                                | INADDR.....                            | 450           |
| if.....                       | 3565                               | INADDR_ANY.....                        | 288           |
| if conditional construct..... | 2269                               | INADDR_BROADCAST.....                  | 288           |
| ifc.....                      | 451                                | include line.....                      | 2834          |
| ifra.....                     | 451                                | incomplete line.....                   | 60            |
| ifru.....                     | 451                                | incomplete pathname.....               | 3329          |
| IF.....                       | 450                                | index.....                             | 3521          |
| if.....                       | 450-451                            | INET6_ADDRSTRLEN.....                  | 288           |
| if_freenameindex().....       | <b>1071</b>                        | inet.....                              | 450           |
| if_indextoname().....         | <b>1072</b>                        | inet_addr().....                       | <b>1079</b>   |
| if_nameindex().....           | <b>1073</b>                        | INET_ADDRSTRLEN.....                   | 288           |
| if_nametoindex().....         | <b>1074</b>                        | inet_ntoa.....                         | 1079          |
| IGNBRK.....                   | 394                                | inet_ntop().....                       | <b>1081</b>   |
| IGNCR.....                    | 394                                | inet_pton.....                         | 1081          |
| IGNPAR.....                   | 394                                | Inf.....                               | 60, 570       |
| ILL.....                      | 450, 452                           | INF.....                               | 867, 943      |
| ILL_BADSTK.....               | 316                                | inference rule.....                    | 2830          |
| ILL_COPROC.....               | 316                                | INFINITY.....                          | 271, 867, 943 |
| ILL_ILLADR.....               | 316                                | INFO.....                              | 845           |
| ILL_ILOPC.....                | 316                                | infu.....                              | 451           |
| ILL_ILOPN.....                | 316                                | inheritance attribute.....             | 513, 1427     |
| ILL_ILLTRP.....               | 316                                | init.....                              | 526, 1164     |
| ILL_PRVOPC.....               | 316                                | initialize a named semaphore.....      | 1780          |
| ILL_PRVREG.....               | 316                                | initializing a mutex.....              | 1592          |
| ilogb().....                  | <b>1075</b>                        | initializing condition variables.....  | 1546          |
| ilogbf.....                   | 1075                               | initstate().....                       | <b>1083</b>   |
| ilogbl.....                   | 1075                               | INIT_PROCESS.....                      | 433, 733-734  |
| imaginary.....                | 210                                | INLCR.....                             | 394           |
| imaxabs().....                | <b>1077</b>                        | ino_t.....                             | 380           |
|                               |                                    | INPCK.....                             | 394           |

## Index

|                                     |                            |                                |                                  |
|-------------------------------------|----------------------------|--------------------------------|----------------------------------|
| input and output rationale.....     | 1700                       | IPC .....                      | 350, 474, 1308, 1310, 1313, 1315 |
| input file descriptor               |                            | .....                          | 1796, 1800, 1864, 1867, 3417     |
| duplication .....                   | 3559                       | ipcrm .....                    | <b>2729</b>                      |
| input mode .....                    | 2553, 3383                 | ipcs .....                     | <b>2731</b>                      |
| input processing .....              | 3382                       | IPC_ .....                     | 450                              |
| canonical mode .....                | 3382                       | ipc_ .....                     | 450                              |
| non-canonical mode .....            | 3382                       | IPC_ constants                 |                                  |
| insque() .....                      | <b>1085</b>                | defined in <sys/ipc.h> .....   | <b>350</b>                       |
| instrumented application .....      | 60                         | used in semctl .....           | 1791                             |
| INT .....                           | 452                        | used in shmctl .....           | 1862                             |
| inter-user communication .....      | 3586, 3592                 | IPC_CREAT .....                | 350, 1309, 1794, 1866            |
| interactive facilities .....        | 3585, 3591                 | IPC_EXCL .....                 | 350, 1309, 1794                  |
| interactive shell .....             | 60                         | IPC_NOWAIT .....               | 350, 1311-1312, 1314-1315, 1797  |
| interface characteristics .....     | 3380                       | IPC_PRIVATE .....              | 350, 1309, 1794, 1866            |
| interfaces .....                    | 3491                       | IPC_RMID .....                 | 350, 1307, 1792, 1862            |
| international environment .....     | 1824                       | IPC_SET .....                  | 350, 1307, 1792, 1862            |
| internationalization .....          | 60                         | IPC_STAT .....                 | 350, 1307, 1791, 1862            |
| internationalization variable ..... | 3369                       | IPPORT .....                   | 452                              |
| Internet Protocols .....            | 503                        | IPPROTO_ .....                 | 450                              |
| interprocess communication .....    | 60, 3417                   | IPPROTO_ICMP .....             | 288                              |
| INTMAX_MAX .....                    | 331                        | IPPROTO_IP .....               | 288                              |
| INTMAX_MIN .....                    | 331                        | IPPROTO_IPV6 .....             | 288                              |
| INTN_MAX .....                      | 330                        | IPPROTO_RAW .....              | 288                              |
| INTN_MIN .....                      | 330                        | IPPROTO_TCP .....              | 288                              |
| INTPTR_MAX .....                    | 330                        | IPPROTO_UDP .....              | 288                              |
| INTPTR_MIN .....                    | 330                        | IPv4 .....                     | 504                              |
| int_curr_symbol .....               | 140                        | IPv4-compatible address .....  | 505                              |
| INT_FASTN_MAX .....                 | 330                        | IPv4-mapped address .....      | 505                              |
| INT_FASTN_MIN .....                 | 330                        | IPv6 .....                     | 504                              |
| int_frac_digits .....               | 140                        | compatibility with IPv4 .....  | 505                              |
| INT_LEASTN_MAX .....                | 330                        | interface identification ..... | 505                              |
| INT_LEASTN_MIN .....                | 330                        | options .....                  | 506                              |
| INT_MAX .....                       | 262, 1075                  | IPv6 address                   |                                  |
| INT_MIN .....                       | 262, 536                   | anycast .....                  | 504                              |
| int_n_cs_precedes .....             | 141                        | loopback .....                 | 505                              |
| int_n_sep_by_space .....            | 141                        | multicast .....                | 504                              |
| int_n_sign_posn .....               | 141                        | unicast .....                  | 504                              |
| int_p_cs_precedes .....             | 141                        | unspecified .....              | 505                              |
| int_p_sep_by_space .....            | 141                        | IPv6_ .....                    | 450                              |
| int_p_sign_posn .....               | 141                        | IPV6_JOIN_GROUP .....          | 289, 506                         |
| invalid .....                       | 168                        | IPV6_LEAVE_GROUP .....         | 289, 506                         |
| use in RE .....                     | 3373                       | IPV6_MULTICAST_HOPS .....      | 289, 506                         |
| invariant values .....              | 264                        | IPV6_MULTICAST_IF .....        | 289, 506                         |
| invoke .....                        | 61                         | IPV6_MULTICAST_LOOP .....      | 289, 506                         |
| in_ .....                           | 450                        | IPV6_UNICAST_HOPS .....        | 289, 506                         |
| IN_ .....                           | 452                        | IPV6_V6ONLY .....              | 289, 506                         |
| ioctl() .....                       | <b>1088</b> , 3379, 3404   | ip_ .....                      | 450                              |
| iovec .....                         | <b>384</b>                 | IP_ .....                      | 452                              |
| iov_ .....                          | 451                        | isalnum() .....                | <b>1099</b>                      |
| IOV_ .....                          | 452                        | isalnum_l .....                | 1099                             |
| IOV_MAX .....                       | 254, 384, 1711, 2010, 2220 | isalpha() .....                | <b>1101</b>                      |
| IP6 .....                           | <b>8</b>                   | isalpha_l .....                | 1101                             |
|                                     |                            | isascii() .....                | <b>1103</b>                      |

|                                              |                         |                |                 |
|----------------------------------------------|-------------------------|----------------|-----------------|
| isastream()                                  | 1104                    | iswlower_l     | 1146            |
| isatty()                                     | 1105                    | iswprint()     | 1148            |
| isblank()                                    | 1106                    | iswprint_l     | 1148            |
| isblank_l                                    | 1106                    | iswpunct()     | 1150            |
| iscntrl()                                    | 1107                    | iswpunct_l     | 1150            |
| iscntrl_l                                    | 1107                    | iswspace()     | 1152            |
| isdigit()                                    | 1109                    | iswspace_l     | 1152            |
| isdigit_l                                    | 1109                    | iswupper()     | 1154            |
| isfinite()                                   | 1111                    | iswupper_l     | 1154            |
| isgraph()                                    | 1112                    | iswxdigit()    | 1156            |
| isgraph_l                                    | 1112                    | iswxdigit_l    | 1156            |
| isgreater()                                  | 1114                    | isxdigit()     | 1158            |
| isgreaterequal()                             | 1115                    | isxdigit_l     | 1158            |
| ISIG                                         | 396                     | itimerval      | 377             |
| isinf()                                      | 1116                    | ITIMER         | 451             |
| isless()                                     | 1117                    | ITIMER_PROF    | 377, 990        |
| islessequal()                                | 1118                    | ITIMER_REAL    | 377, 990        |
| islessgreater()                              | 1119                    | ITIMER_VIRTUAL | 377, 990        |
| islower()                                    | 1120                    | it_            | 451             |
| islower_l                                    | 1120                    | IXANY          | 394             |
| isnan()                                      | 1122                    | IXOFF          | 394             |
| isnormal()                                   | 1123                    | IXON           | 394             |
| ISO/IEC 646: 1991 standard                   | 3335                    | I_             | 450             |
| ISO C standard                               | 210, 445, 563, 752, 784 | I_ ATMARK      | 346, 1094-1095  |
| .....960, 1228, 1694, 1743, 1824, 1875, 1900 |                         | I_ CANPUT      | 346, 1095       |
| .....2057, 3322, 3325, 3352, 3379, 3397-3399 |                         | I_ CKBAND      | 346, 1095       |
| .....3401-3402, 3405, 3407, 3414, 3518       |                         | I_ FDINSERT    | 346, 1091       |
| isprint()                                    | 1124                    | I_ FIND        | 346, 1090       |
| isprint_l                                    | 1124                    | I_ FLUSH       | 346, 1089       |
| ispunct()                                    | 1126                    | I_ FLUSHBAND   | 346, 1089       |
| ispunct_l                                    | 1126                    | I_ GETBAND     | 346, 1095       |
| isspace()                                    | 1128                    | I_ GETCLTIME   | 346, 1095       |
| isspace_l                                    | 1128                    | I_ GETSIG      | 346, 1090       |
| ISTRIP                                       | 394, 3383               | I_ GRDOPT      | 346, 1091, 1698 |
| isunordered()                                | 1130                    | I_ GWROPT      | 346, 1093       |
| isupper()                                    | 1131                    | I_ ISVTX       | 2455            |
| isupper_l                                    | 1131                    | I_ LINK        | 346, 1095       |
| iswalnum()                                   | 1133                    | I_ LIST        | 346, 1094       |
| iswalnum_l                                   | 1133                    | I_ LOOK        | 346, 1088       |
| iswalpunct_l                                 | 1135                    | I_ NREAD       | 346, 1091       |
| iswblank()                                   | 1137                    | I_ PEEK        | 346, 1090       |
| iswblank_l                                   | 1137                    | I_ PLINK       | 346, 1096       |
| iswcntrl()                                   | 1138                    | I_ POP         | 346, 1088       |
| iswcntrl_l                                   | 1138                    | I_ PUNLINK     | 346, 1097       |
| iswctype()                                   | 1140                    | I_ PUSH        | 346, 1088       |
| iswctype_l                                   | 1140                    | I_ RECVFD      | 346, 457, 1094  |
| iswdigit()                                   | 1142                    | I_ SENDFD      | 346, 1093-1094  |
| iswdigit_l                                   | 1142                    | I_ SETCLTIME   | 346, 651, 1095  |
| iswgraph()                                   | 1144                    | I_ SETSIG      | 346, 1089-1090  |
| iswgraph_l                                   | 1144                    | I_ SRDOPT      | 346, 1091, 1698 |
| iswlower()                                   | 1146                    | I_ STR         | 346, 1092       |
|                                              |                         | I_ SWROPT      | 346, 1093, 2214 |
|                                              |                         | I_ UNLINK      | 346, 1096       |

## Index

- j0().....1160
- j1.....1160
- jn.....1160
- job .....61
- job control .....61, 526, 1012, 1164, 1829  
.....1841, 2013, 2136, 3329-3332, 3334  
.....3380-3381, 3407, 3414, 3587
- job control job ID .....61
- job control, implementing applications .....3331
- job control, implementing shells .....3329
- job control, implementing systems .....3332
- jobs.....2244, 2264, 2737, 3545, 3591
- Job\_Owner.....2328
- join .....2741, 3592
- Join\_Path.....2328
- jrnd48.....712
- jrnd48() .....1162
- JST-9.....2092
- Keep\_Files.....2328
- kernel.....3332
- kernel entity .....3480
- keyword-value pairs .....2341
- key\_t.....380
- kill .....2244, 2264, 2746, 3545, 3591
- kill() .....1163, 3407-3408, 3411-3412, 3414, 3518
- killpg() .....1166
- l64a.....532
- l64a() .....1168
- labs() .....1169
- LANG.....160, 615
- last close .....1858, 3432
- last close (of a file) .....61
- LASTMARK .....348, 1095
- lchown().....1170, 3337
- lcong48 .....712
- lcong48().....1172
- LC\_ALL.....161, 267, 746, 1199, 1340, 1823, 1825
- LC\_COLLATE.....161, 258, 267, 1052-1053  
.....1823, 1825, 1944, 2000, 2151, 2193, 3360  
description.....132
- LC\_CTYPE.....161, 250, 267, 439, 598  
.....1140, 1233, 1235, 1237, 1239, 1241, 1243  
.....1245, 1823, 1825, 2074, 2076, 2078, 2080  
.....2082, 2144, 2168, 2185, 2194-2195, 2197  
.....2199, 3359  
description.....124
- LC\_MESSAGES .....161, 250, 267, 292  
.....615, 1823-1825, 1952, 3364  
description.....150
- LC\_MONETARY.....161, 250, 267, 1199  
.....1823, 1825, 1957, 3362  
description.....139
- LC\_NUMERIC .....161, 250, 267, 865  
.....900, 940, 949, 1199, 1823, 1825, 1957  
.....1985, 2173, 3363, 3387  
description.....143
- LC\_TIME .....161, 250, 267, 966, 1340  
.....1823, 1825, 3363  
description.....144
- ld, rationale for omission.....3577
- LDBL\_ constants  
defined in <float.h>.....233
- LDBL\_DIG.....234
- LDBL\_EPSILON .....235
- LDBL\_MANT\_DIG .....233
- LDBL\_MAX.....235
- LDBL\_MAX\_10\_EXP .....234
- LDBL\_MAX\_EXP .....234
- LDBL\_MIN.....235
- LDBL\_MIN\_10\_EXP .....234
- LDBL\_MIN\_EXP .....234
- ldexp().....1173
- ldexpf.....1173
- ldexpl.....1173
- ldiv().....1175
- legacy.....5, 3317
- legacy, rationale .....3317
- lex .....2751, 3593-3594  
actions .....2757  
definitions.....2753  
escape sequences .....2756  
regular expressions .....2755  
rules.....2754  
table sizes.....2754  
user subroutines .....2755
- lex, translation table .....2761
- lfind .....1226
- lfind() .....1177
- lgamma() .....1178
- lgammaf .....1178
- lgammal .....1178
- libraries  
ar command .....2352
- library routine .....3332
- LIMIT .....2233
- limit  
numerical .....262
- limits.....3537
- line .....61
- line counting.....3271
- line, rationale for omission.....3577
- LINES .....163, 3370
- LINE\_MAX.....258, 2010, 2234, 2372, 2565  
.....2574, 2827, 3073, 3190, 3333, 3343, 3538
- linger .....61

|                               |                                       |
|-------------------------------|---------------------------------------|
| link                          | 62, 2763, 3588                        |
| link count                    | 62                                    |
| link to a file                | 1182                                  |
| link()                        | <b>1180</b> , 3326, 3338              |
| linkat                        | 1180                                  |
| linkat()                      | <b>1184</b>                           |
| LINK_MAX                      | 256, 459, 858, 1181, 1741, 3538, 3601 |
| lint, rationale for omission  | 3577                                  |
| LIO_                          | 450                                   |
| lio_                          | 450                                   |
| lio_listio()                  | <b>1185</b> , 3411, 3426              |
| LIO_NOP                       | 206, 1185                             |
| LIO_NOWAIT                    | 206, 1185                             |
| LIO_READ                      | 206, 1185                             |
| LIO_WAIT                      | 206, 1185                             |
| LIO_WRITE                     | 206, 1185                             |
| list directed I/O             | 1187                                  |
| listen()                      | <b>1189</b>                           |
| lists                         | 2266, 3563                            |
| AND-OR                        | 2266                                  |
| compound-list                 | 2266                                  |
| llabs                         | 1169                                  |
| llabs()                       | <b>1191</b>                           |
| lldiv                         | 1175                                  |
| lldiv()                       | <b>1192</b>                           |
| LLONG_MAX                     | 262, 1993, 2181                       |
| LLONG_MIN                     | 263, 1993, 2181                       |
| llrint()                      | <b>1193</b>                           |
| llrintf                       | 1193                                  |
| llrintl                       | 1193                                  |
| llround()                     | <b>1195</b>                           |
| llroundf                      | 1195                                  |
| llroundl                      | 1195                                  |
| ln                            | <b>2765</b> , 3543, 3592              |
| LNKTYPE                       | 391                                   |
| load ordering                 | 707                                   |
| LOBLK                         | 452                                   |
| local customs                 | 62                                    |
| local IPC                     | 62                                    |
| local mode                    | 3384                                  |
| locale                        | 62, 121, 2769, 3357, 3592             |
| definition example            | 3365                                  |
| definition grammar            | 3364                                  |
| grammar                       | 151, 3365                             |
| lexical conventions           | 3365                                  |
| POSIX                         | 122                                   |
| locale configuration          | 3586, 3592                            |
| locale definition             | 122, 3358                             |
| localeconv()                  | <b>1197</b>                           |
| localedef                     | <b>2774</b> , 3592-3593               |
| localization                  | 62                                    |
| localtime()                   | <b>1201</b> , 3352, 3477              |
| localtime_r                   | 1201                                  |
| Locate Batch Job Request      | 2335                                  |
| lockf()                       | <b>1205</b>                           |
| locking                       | 784                                   |
| advisory                      | 784                                   |
| mandatory                     | 784                                   |
| locking and unlocking a mutex | 1601                                  |
| locking file                  | 2457                                  |
| log()                         | <b>1208</b>                           |
| log-full-policy attribute     | 511, 513, 1427, 1430, 1447            |
| log-max-size attribute        | 513, 1428, 1430                       |
| log10()                       | <b>1210</b>                           |
| log10f                        | 1210                                  |
| log10l                        | 1210                                  |
| log1p()                       | <b>1212</b>                           |
| log1pf                        | 1212                                  |
| log1pl                        | 1212                                  |
| log2()                        | <b>1214</b>                           |
| log2f                         | 1214                                  |
| log2l                         | 1214                                  |
| logb()                        | <b>1216</b>                           |
| logbf                         | 1216                                  |
| logbl                         | 1216                                  |
| logf                          | 1208                                  |
| logf()                        | <b>1218</b>                           |
| logger                        | <b>2778</b> , 3593                    |
| logical device                | 3332                                  |
| login                         | 62                                    |
| login name                    | 62                                    |
| login shell                   | 752                                   |
| login, rationale for omission | 3577                                  |
| LOGIN_NAME_MAX                | 254, 993, 2010, 3601                  |
| LOGIN_PROCESS                 | 433, 733-734                          |
| logl                          | 1208, 1218                            |
| LOGNAME                       | <b>164</b>                            |
| logname                       | <b>2781</b>                           |
| LOGNAME                       | 3315, 3370                            |
| logname                       | 3593                                  |
| LOG_                          | 451                                   |
| LOG_ constants in syslog      | <b>656</b>                            |
| LOG_ALERT                     | 390, 656                              |
| LOG_AUTH                      | 389                                   |
| LOG_CONS                      | 389, 657                              |
| LOG_CRIT                      | 390, 656                              |
| LOG_CRON                      | 389                                   |
| LOG_DAEMON                    | 389                                   |
| LOG_DEBUG                     | 390, 656                              |
| LOG_EMERG                     | 390, 656                              |
| LOG_ERR                       | 390, 656                              |
| LOG_INFO                      | 390, 656                              |
| LOG_KERN                      | 389                                   |
| LOG_LOCAL                     | 389, 657                              |
| LOG_LPR                       | 389                                   |



## Index

|                                       |                                   |                                       |                 |
|---------------------------------------|-----------------------------------|---------------------------------------|-----------------|
| LOG_MAIL .....                        | 389                               | declare aliases .....                 | 2816            |
| LOG_MASK .....                        | 389                               | declare alternatives .....            | 2817            |
| LOG_NDELAY .....                      | 389, 657                          | delete aliases.....                   | 2823            |
| LOG_NEWS .....                        | 389                               | delete messages .....                 | 2817            |
| LOG_NOTICE.....                       | 390, 656                          | delete messages and display.....      | 2818            |
| LOG_NOWAIT .....                      | 389, 657                          | direct messages to mbox.....          | 2820            |
| LOG_ODELAY .....                      | 389, 657                          | discard header fields.....            | 2817            |
| LOG_PID .....                         | 389, 657                          | display beginning of messages.....    | 2823            |
| LOG_USER.....                         | 389, 656-657                      | display current message number.....   | 2824            |
| LOG_UUCP.....                         | 389                               | display header summary.....           | 2819            |
| LOG_WARNING.....                      | 390, 656                          | display list of folders.....          | 2819            |
| longjmp() .....                       | 1219, 3404, 3415, 3485-3486, 3590 | display message.....                  | 2821            |
| LONG_BIT .....                        | 262-263                           | display message size .....            | 2822            |
| LONG_MAX .....                        | 263, 1993, 2181, 3385             | echo a string .....                   | 2818            |
| LONG_MIN .....                        | 263, 1993, 2181, 3385             | edit message .....                    | 2818, 2823      |
| lorder, rationale for omission.....   | 3577                              | execute commands conditionally.....   | 2820            |
| lower multiplexing.....               | 83                                | exit .....                            | 2818            |
| lp.....                               | 2783, 3593                        | follow up specified messages .....    | 2819            |
| lpstat, rationale for omission .....  | 3577                              | help .....                            | 2819            |
| LR(1) grammars.....                   | 3303                              | hold messages .....                   | 2819            |
| lrand48 .....                         | 712                               | internal variables .....              | 2813            |
| lrand48().....                        | 1221                              | invoke a shell .....                  | 2822            |
| lrint() .....                         | 1222                              | invoke shell command.....             | 2824            |
| lrintf .....                          | 1222                              | list available commands .....         | 2820            |
| lrintl .....                          | 1222                              | mail a message .....                  | 2820            |
| lround() .....                        | 1224                              | null command .....                    | 2824            |
| lroundf .....                         | 1224                              | pipe message.....                     | 2820            |
| lroundl.....                          | 1224                              | process next specified message .....  | 2820            |
| ls .....                              | 2788, 3543, 3592                  | quit.....                             | 2821            |
| lsearch() .....                       | 1226                              | read mailx commands from a file ..... | 2822            |
| lseek .....                           | 3588                              | receive mode .....                    | 2806            |
| lseek() .....                         | 1228, 3425-3426, 3432, 3476, 3518 | reply to a message .....              | 2821            |
| lstat .....                           | 915, 3587-3588                    | reply to a message list.....          | 2821            |
| lstat() .....                         | 1230, 3338, 3543                  | retain header fields.....             | 2822            |
| lutime().....                         | 3337                              | save messages .....                   | 2822            |
| L_ .....                              | 450-451                           | scroll header display .....           | 2824            |
| L_ANCHOR.....                         | 177                               | send mode .....                       | 2806            |
| L_ctermid.....                        | 334, 687                          | set variables.....                    | 2822            |
| L_sysid .....                         | 784                               | start-up.....                         | 2813            |
| L_tmpnam .....                        | 334                               | touch messages .....                  | 2823            |
| m4 .....                              | 2797                              | undelete messages.....                | 2823            |
| macro.....                            | 3318                              | unset variables .....                 | 2823            |
| macro processor.....                  | 2797                              | write messages to a file.....         | 2823            |
| MAGIC.....                            | 213                               | Mail_Points.....                      | 2329            |
| magic file.....                       | 2665                              | Mail_Users.....                       | 2329            |
| mail, rationale for omission .....    | 3577                              | make.....                             | 2830, 3593-3594 |
| mailx.....                            | 2806, 3591-3592                   | default rules .....                   | 2840            |
| change current directory .....        | 2817                              | inference rules.....                  | 2838            |
| change folder.....                    | 2818                              | internal macros .....                 | 2839            |
| command escapes.....                  | 2824                              | libraries .....                       | 2839            |
| commands .....                        | 2816                              | macros .....                          | 2836            |
| copy messages.....                    | 2817                              | makefile execution .....              | 2834            |
| declare aliases .....                 | 2816                              | makefile syntax .....                 | 2833            |
| declare alternatives .....            | 2817                              |                                       |                 |
| delete aliases.....                   | 2823                              |                                       |                 |
| delete messages .....                 | 2817                              |                                       |                 |
| delete messages and display.....      | 2818                              |                                       |                 |
| direct messages to mbox.....          | 2820                              |                                       |                 |
| discard header fields.....            | 2817                              |                                       |                 |
| display beginning of messages.....    | 2823                              |                                       |                 |
| display current message number.....   | 2824                              |                                       |                 |
| display header summary.....           | 2819                              |                                       |                 |
| display list of folders.....          | 2819                              |                                       |                 |
| display message.....                  | 2821                              |                                       |                 |
| display message size .....            | 2822                              |                                       |                 |
| echo a string .....                   | 2818                              |                                       |                 |
| edit message .....                    | 2818, 2823                        |                                       |                 |
| execute commands conditionally.....   | 2820                              |                                       |                 |
| exit .....                            | 2818                              |                                       |                 |
| follow up specified messages .....    | 2819                              |                                       |                 |
| help .....                            | 2819                              |                                       |                 |
| hold messages .....                   | 2819                              |                                       |                 |
| internal variables .....              | 2813                              |                                       |                 |
| invoke a shell .....                  | 2822                              |                                       |                 |
| invoke shell command.....             | 2824                              |                                       |                 |
| list available commands .....         | 2820                              |                                       |                 |
| mail a message .....                  | 2820                              |                                       |                 |
| null command .....                    | 2824                              |                                       |                 |
| pipe message.....                     | 2820                              |                                       |                 |
| process next specified message .....  | 2820                              |                                       |                 |
| quit.....                             | 2821                              |                                       |                 |
| read mailx commands from a file ..... | 2822                              |                                       |                 |
| receive mode .....                    | 2806                              |                                       |                 |
| reply to a message .....              | 2821                              |                                       |                 |
| reply to a message list.....          | 2821                              |                                       |                 |
| retain header fields.....             | 2822                              |                                       |                 |
| save messages .....                   | 2822                              |                                       |                 |
| scroll header display .....           | 2824                              |                                       |                 |
| send mode .....                       | 2806                              |                                       |                 |
| set variables.....                    | 2822                              |                                       |                 |
| start-up.....                         | 2813                              |                                       |                 |
| touch messages .....                  | 2823                              |                                       |                 |
| undelete messages.....                | 2823                              |                                       |                 |
| unset variables .....                 | 2823                              |                                       |                 |
| write messages to a file.....         | 2823                              |                                       |                 |
| Mail_Points.....                      | 2329                              |                                       |                 |
| Mail_Users.....                       | 2329                              |                                       |                 |
| make.....                             | 2830, 3593-3594                   |                                       |                 |
| default rules .....                   | 2840                              |                                       |                 |
| inference rules.....                  | 2838                              |                                       |                 |
| internal macros .....                 | 2839                              |                                       |                 |
| libraries .....                       | 2839                              |                                       |                 |
| macros .....                          | 2836                              |                                       |                 |
| makefile execution .....              | 2834                              |                                       |                 |
| makefile syntax .....                 | 2833                              |                                       |                 |

|                               |                                  |                                     |                          |
|-------------------------------|----------------------------------|-------------------------------------|--------------------------|
| target rules.....             | 2835                             | memccpy() .....                     | <b>1247</b>              |
| make, GNU version.....        | 2846                             | memchr().....                       | <b>1248</b>              |
| makecontext .....             | 3521                             | memcmp().....                       | <b>1249</b>              |
| malloc().....                 | <b>1231</b> , 3415, 3435, 3437   | memcpy() .....                      | <b>1250</b>              |
| .....                         | 3465, 3478, 3487-3488            | MEMLOCK_FUTURE .....                | 1280                     |
| man.....                      | <b>2851</b> , 3591               | memmove().....                      | <b>1251</b>              |
| manipulate signal sets .....  | 1881                             | memory locking.....                 | 3428                     |
| map.....                      | 62, 3332                         | memory management.....              | 478, 3427, 3588          |
| mapped .....                  | 3332                             | memory management unit.....         | 3428                     |
| mappings .....                | 1280                             | memory mapped files.....            | 63                       |
| MAP_ .....                    | 450                              | memory object.....                  | 63, 3333                 |
| MAP_FAILED .....              | 1280                             | memory synchronization .....        | 98, 3349                 |
| MAP_FIXED.....                | 352, 1276, 1279                  | memory-resident .....               | 63, 3332                 |
| MAP_PRIVATE.....              | 352, 853, 1276, 1280, 1285, 1317 | memset().....                       | <b>1252</b>              |
| MAP_SHARED .....              | 352, 856, 1276-1277              | msg .....                           | <b>2855</b> , 3592-3593  |
| margin code notation.....     | 3319                             | message.....                        | 63                       |
| margin codes.....             | 3318                             | message catalog .....               | 64                       |
| notation .....                | 13                               | message catalog descriptor .....    | 64, 523, 746, 751        |
| marked message .....          | 63                               | message catalog generation .....    | 2689                     |
| matched.....                  | 63, 167                          | message parts.....                  | 474                      |
| mathematical functions .....  | 2233                             | message passing.....                | 3419, 3588-3589          |
| domain error .....            | 104                              | message priority .....              | 474                      |
| error conditions .....        | 104, 3353                        | high-priority.....                  | 474                      |
| NaN arguments .....           | 105, 3353                        | normal .....                        | 474                      |
| pole error .....              | 104                              | priority .....                      | 474                      |
| range error .....             | 104                              | message queue.....                  | 64, 3419                 |
| max-data-size attribute ..... | 513, 1430-1431                   | message-discard mode .....          | 1698                     |
| MAXARGS .....                 | 323                              | message-nondiscard mode.....        | 1698                     |
| MAXFLOAT .....                | 271                              | MET-1MEST .....                     | 2092                     |
| maximum values .....          | 258                              | META_CHAR .....                     | 177                      |
| MAX_CANON.....                | 256, 858, 3382, 3538, 3601       | MIL-STD-1753 .....                  | 2684                     |
| MAX_INPUT.....                | 256, 858, 3538, 3602             | minimum values.....                 | 258                      |
| may .....                     | 5, 3316                          | Minimum_Cpu_Interval .....          | 2326                     |
| may, rationale.....           | 3316                             | MINSIGSTKSZ.....                    | 314, 1878                |
| mblen() .....                 | <b>1233</b>                      | mkdir .....                         | <b>2858</b> , 3588, 3592 |
| mbrlen().....                 | <b>1235</b>                      | mkdir().....                        | <b>1253</b>              |
| mbrtowc().....                | <b>1237</b>                      | mkdirat.....                        | 1253                     |
| mbsinit() .....               | <b>1239</b>                      | mkdirat() .....                     | <b>1256</b>              |
| mbsnrtowcs .....              | 1241                             | mkdtemp() .....                     | <b>1257</b>              |
| mbsnrtowcs().....             | <b>1240</b>                      | mkfifo .....                        | <b>2861</b> , 3592       |
| mbsrtowcs() .....             | <b>1241</b>                      | mkfifo().....                       | <b>1259</b> , 3328       |
| mbstowcs().....               | <b>1243</b>                      | mkfifoat.....                       | 1259                     |
| mbtowc() .....                | <b>1245</b>                      | mkfifoat() .....                    | <b>1262</b>              |
| MB_CUR_MAX .....              | 338, 1233, 1235, 1237            | mknod().....                        | <b>1263</b> , 3328       |
| .....                         | 1245, 2144, 2195                 | mknod, rationale for omission ..... | 3577                     |
| MB_LEN_MAX .....              | 262-263                          | mknodat.....                        | 1263                     |
| MC1 .....                     | <b>8</b>                         | mknodat() .....                     | <b>1267</b>              |
| MCL_.....                     | 450                              | mkstemp .....                       | 1257                     |
| MCL_CURRENT .....             | 352, 1273                        | mkstemp().....                      | <b>1268</b>              |
| MCL_FUTURE.....               | 352, 1273, 3429                  | mktemp .....                        | 3521                     |
| MCL_INHERIT.....              | 3430                             | mktime().....                       | <b>1269</b> , 3352       |
| mcontext_t.....               | <b>314</b>                       | ML.....                             | <b>8</b>                 |
|                               |                                  | mlock().....                        | <b>1271</b>              |

## Index

|                                    |                            |
|------------------------------------|----------------------------|
| mlockall()                         | 1273, 3429                 |
| MLR                                | 8                          |
| mmap()                             | 1275, 3431-3433, 3435      |
| MMU                                | 3428                       |
| MM_                                | 450                        |
| MM_macros                          | 236                        |
| MM_APPL                            | 236, 844                   |
| MM_CONSOLE                         | 236, 844                   |
| MM_ERROR                           | 236, 845-846               |
| MM_FIRM                            | 236                        |
| mm_FIRM                            | 844                        |
| MM_HALT                            | 236, 845                   |
| MM_HARD                            | 236, 844                   |
| MM_INFO                            | 236, 845                   |
| MM_NOCON                           | 237, 845                   |
| MM_NOMSG                           | 236, 845                   |
| MM_NOSEV                           | 236, 845                   |
| MM_NOTOK                           | 236, 845                   |
| MM_NRECOV                          | 236, 844                   |
| MM_NULLACT                         | 236                        |
| MM_NULLLBL                         | 236                        |
| MM_NULLMC                          | 236, 844                   |
| MM_NULLSEV                         | 236                        |
| MM_NULLTAG                         | 236                        |
| MM_NULLTXT                         | 236                        |
| MM_OK                              | 236, 845                   |
| MM_OPYSYS                          | 236, 844                   |
| MM_PRINT                           | 236, 844, 846              |
| MM_RECOVER                         | 236, 844                   |
| MM_SOFT                            | 236, 844                   |
| MM_UTIL                            | 236, 844                   |
| MM_WARNING                         | 236, 845                   |
| mode                               | 64                         |
| modem disconnect                   | 3383                       |
| mode_t                             | 380                        |
| modf()                             | 1283                       |
| modff                              | 1283                       |
| modfl                              | 1283                       |
| Modify Batch Job Request           | 2335                       |
| MON                                | 8                          |
| monotonic clock                    | 64, 3451                   |
| MON_                               | 250                        |
| mon_decimal_point                  | 140                        |
| mon_grouping                       | 140                        |
| mon_thousands_sep                  | 140                        |
| more                               | 2864, 3591-3592            |
| discard and refresh                | 2870                       |
| display position                   | 2872                       |
| examine new file                   | 2871                       |
| examine next file                  | 2871                       |
| examine previous file              | 2872                       |
| go to beginning of file            | 2870                       |
| go to end-of-file                  | 2870                       |
| go to tag                          | 2872                       |
| help                               | 2869                       |
| invoke editor                      | 2872                       |
| mark position                      | 2870                       |
| quit                               | 2872                       |
| refresh the screen                 | 2870                       |
| repeat search                      | 2871                       |
| repeat search in reverse           | 2871                       |
| return to mark                     | 2870                       |
| return to previous position        | 2870                       |
| scroll backward one half screenful | 2870                       |
| scroll backward one line           | 2869                       |
| scroll backward one screenful      | 2869                       |
| scroll forward one half screenful  | 2869                       |
| scroll forward one line            | 2869                       |
| scroll forward one screenful       | 2869                       |
| search backward for pattern        | 2871                       |
| search forward for pattern         | 2871                       |
| skip forward one line              | 2869                       |
| MORECTL                            | 348, 997                   |
| MOREDATA                           | 348, 997                   |
| motion command                     | 3079                       |
| mount point                        | 64, 3333                   |
| mount()                            | 3333                       |
| mounted file system                | 3333                       |
| Move Batch Job Request             | 2336                       |
| mprotect()                         | 1285, 3432                 |
| MQ                                 | 450                        |
| mq                                 | 450                        |
| mq_close()                         | 1287                       |
| mq_getattr()                       | 1288                       |
| mq_notify()                        | 1290                       |
| mq_open()                          | 1292, 3420                 |
| MQ_OPEN_MAX                        | 254, 2010, 3602            |
| MQ_PRIO_MAX                        | 254, 1298-1299, 2010, 3602 |
| mq_receive()                       | 1295, 3421                 |
| mq_send()                          | 1298, 3421                 |
| mq_setattr()                       | 1300                       |
| mq_timedreceive                    | 1295                       |
| mq_timedreceive()                  | 1302, 3452                 |
| mq_timedsend                       | 1298                       |
| mq_timedsend()                     | 1303, 3452                 |
| mq_unlink()                        | 1304                       |
| mrnd48                             | 712                        |
| mrnd48()                           | 1306                       |
| MSG                                | 8                          |
| msg                                | 450                        |
| msg*()                             | 3417                       |
| msgctl()                           | 1307, 3417                 |
| msgget()                           | 1309, 3417                 |
| msgrcv()                           | 1311, 3417                 |
| msgsnd()                           | 1314, 3417                 |

|                                        |                                   |                                   |                                          |
|----------------------------------------|-----------------------------------|-----------------------------------|------------------------------------------|
| MSGVERB.....                           | 164, 845-846                      | name space .....                  | 449, 3399                                |
| MSG_.....                              | 450-451                           | name space pollution.....         | 3398-3399                                |
| msg.....                               | 451                               | named STREAM.....                 | 65                                       |
| MSG_ANY.....                           | 348, 996                          | NAME_MAX.....                     | 99, 217, 257, 459, 539, 565              |
| MSG_BAND.....                          | 348, 996, 1679                    | .....                             | 615, 629, 632, 636, 750, 767, 788, 795   |
| MSG_CTRUNC.....                        | 367                               | .....                             | 850, 858, 895, 916, 920, 927, 937, 1170  |
| MSG_DONTRROUTE.....                    | 367                               | .....                             | 1181, 1254, 1259, 1264, 1335, 1347, 1479 |
| MSG_EOR.....                           | 367, 1802, 1804, 1807, 1921, 1923 | .....                             | 1704, 1708, 1715, 1741, 1749, 2005, 2085 |
| MSG_HIPRI.....                         | 348, 996, 1679                    | .....                             | 2105, 2114, 2357, 2952, 3538, 3602       |
| MSG_NOERROR.....                       | 355, 1311-1312                    | NaN.....                          | 232                                      |
| MSG_NOSIGNAL.....                      | 367, 1802, 1804, 1807             | NAN.....                          | 271                                      |
| MSG_OOB.....                           | 367, 1802, 1804, 1807             | NaN.....                          | 570                                      |
| MSG_PEEK.....                          | 367                               | NAN.....                          | 867                                      |
| msg_perm.....                          | 475                               | NaN.....                          | 867                                      |
| MSG_TRUNC.....                         | 367                               | NAN.....                          | 943                                      |
| MSG_WAITALL.....                       | 367                               | NaN.....                          | 943                                      |
| msqid.....                             | 475                               | NaN (Not a Number).....           | 65                                       |
| MST7MDT.....                           | 2092                              | NaN arguments.....                | 3353                                     |
| msync().....                           | 1317, 3432                        | mathematical functions.....       | 105                                      |
| MS.....                                | 450                               | nan().....                        | 1324                                     |
| MS_ASYNC.....                          | 352, 1277, 1317                   | nanf.....                         | 1324                                     |
| MS_INVALIDATE.....                     | 352, 1317-1318                    | nanl.....                         | 1324                                     |
| MS_SYNC.....                           | 352, 1277, 1317                   | nanosleep().....                  | 1325, 3449, 3451-3452, 3589              |
| multi-byte character.....              | 3327, 3382-3383                   | native implementation.....        | 3333                                     |
| multi-character collating element..... | 64                                | native language.....              | 65                                       |
| multicast.....                         | 504                               | NCCS.....                         | 393                                      |
| multiple tasks.....                    | 3585, 3591                        | NDEBUG.....                       | 209, 455, 575                            |
| munlock.....                           | 1271                              | nearbyint().....                  | 1327                                     |
| munlock().....                         | 1320                              | nearbyintf.....                   | 1327                                     |
| munlockall.....                        | 1273                              | nearbyintl.....                   | 1327                                     |
| munlockall().....                      | 1321                              | negative response.....            | 65                                       |
| munmap().....                          | 1322, 3431-3433, 3435, 3438       | negative_sign.....                | 140                                      |
| mutex.....                             | 64, 3469                          | network.....                      | 65                                       |
| mutex attributes.....                  | 1609                              | network address.....              | 65                                       |
| extended.....                          | 3472                              | network byte order.....           | 66, 98, 3348                             |
| mutex initialization.....              | 3484                              | network interfaces.....           | 496                                      |
| mutex initialization attributes.....   | 1608                              | newgrp.....                       | 2244, 2264, 2881, 3593                   |
| mutex performance.....                 | 1609                              | newline character.....            | 66                                       |
| MUXID_ALL.....                         | 348, 1096-1097                    | newlocale().....                  | 1329                                     |
| MUXID_R.....                           | 452                               | news, rationale for omission..... | 3578                                     |
| mv.....                                | 2876, 3543, 3592                  | NEW_TIME.....                     | 433, 733-734                             |
| MX.....                                | 9                                 | nextafter().....                  | 1332                                     |
| M.....                                 | 271, 452                          | nextafterf.....                   | 1332                                     |
| M_E.....                               | 270                               | nextafterl.....                   | 1332                                     |
| M_LN.....                              | 270                               | nexttoward.....                   | 1332                                     |
| M_LOG10E.....                          | 270                               | nexttowardf.....                  | 1332                                     |
| M_LOG2E.....                           | 270                               | nexttowardl.....                  | 1332                                     |
| M_PI.....                              | 270                               | nftw().....                       | 1334                                     |
| M_SQRT1_2.....                         | 271                               | NGROUPS_MAX.....                  | 258, 986, 2010, 2884                     |
| M_SQRT2.....                           | 271                               | .....                             | 3315, 3336, 3538, 3596, 3602             |
| name.....                              | 65                                | nice.....                         | 2885, 3591                               |
| name information.....                  | 1000                              | nice value.....                   | 66, 3333                                 |
|                                        |                                   | nice().....                       | 1338, 3443                               |

## Index

|                                                    |                                           |
|----------------------------------------------------|-------------------------------------------|
| Ninth Edition UNIX.....                            | 2421, 2552, 2970                          |
| NI_DGRAM .....                                     | 284                                       |
| NI_NAMEREQD .....                                  | 284                                       |
| NI_NOFQDN.....                                     | 284                                       |
| NI_NUMERICHOST.....                                | 284                                       |
| NI_NUMERICSCOPE.....                               | 284                                       |
| NI_NUMERICSERV.....                                | 284                                       |
| nl.....                                            | <b>2889</b>                               |
| NLDLY .....                                        | 394                                       |
| nlink_t.....                                       | 380                                       |
| NLn.....                                           | 394                                       |
| NLSPATH .....                                      | <b>161</b> , 615                          |
| NL.....                                            | 450                                       |
| NL_ARGMAX.....                                     | 264, 864, 900, 940, 949                   |
| NL_CAT_LOCALE.....                                 | 292, 615                                  |
| nl_langinfo().....                                 | <b>1340</b> , 3590                        |
| nl_langinfo_l.....                                 | 1340                                      |
| NL_LANGMAX.....                                    | 264                                       |
| NL_MSGMAX.....                                     | 264                                       |
| NL_SETD.....                                       | 292                                       |
| NL_SETMAX.....                                     | 264                                       |
| NL_TEXTMAX.....                                    | 264                                       |
| nm.....                                            | <b>2893</b> , 3593                        |
| noclobber option.....                              | 2923, 3557                                |
| NOEXPR .....                                       | 250                                       |
| NOFLSH .....                                       | 396                                       |
| nohup.....                                         | <b>2898</b> , 3591                        |
| nohup utility .....                                | 753                                       |
| non-blocking .....                                 | 66                                        |
| non-canonical mode input processing .....          | 188, 3382                                 |
| non-local jumps .....                              | 1900                                      |
| non-printable.....                                 | 2565, 3071, 3126, 3343                    |
| non-spacing characters.....                        | 66                                        |
| non-volatile storage.....                          | 923                                       |
| normative references.....                          | 3316                                      |
| NOSTR.....                                         | 250                                       |
| NQS.....                                           | 3571                                      |
| rand48.....                                        | 712                                       |
| rand48().....                                      | <b>1342</b>                               |
| ntohl.....                                         | 1062                                      |
| ntohl().....                                       | <b>1343</b>                               |
| ntohs.....                                         | 1062, 1343                                |
| NUL.....                                           | 66                                        |
| NULL.....                                          | 325, 403, 661, 687, 694, 704, 709         |
| .....                                              | 1280, 1706                                |
| null byte.....                                     | 66                                        |
| null pointer.....                                  | 67                                        |
| null string.....                                   | 67                                        |
| null wide-character code.....                      | 67                                        |
| number sign.....                                   | 67                                        |
| numerical limits.....                              | 262                                       |
| NUM_EMPL.....                                      | 1060                                      |
| NZERO.....                                         | 264, 1016, 1338                           |
| n_.....                                            | 450                                       |
| n_cs_precedes.....                                 | 141                                       |
| n_sep_by_space.....                                | 141                                       |
| n_sign_posn.....                                   | 141                                       |
| OB.....                                            | <b>9</b>                                  |
| object file.....                                   | 67                                        |
| object files.....                                  | 2893                                      |
| obsolescent.....                                   | 3316                                      |
| obsolescent, rationale.....                        | 3316                                      |
| OCRNL.....                                         | 394                                       |
| octet.....                                         | 67                                        |
| od.....                                            | <b>2901</b> , 3592-3593                   |
| OF.....                                            | <b>9</b>                                  |
| OFDEL.....                                         | 394                                       |
| offset maximum.....                                | 67                                        |
| off_t.....                                         | 380                                       |
| OFILL.....                                         | 394                                       |
| OH.....                                            | <b>9</b>                                  |
| OLD_TIME.....                                      | 433, 733-734                              |
| ONLCR.....                                         | 394                                       |
| ONLRET.....                                        | 394                                       |
| ONOCR.....                                         | 394                                       |
| opaque address.....                                | 67                                        |
| open.....                                          | 3588                                      |
| open a file.....                                   | 1349                                      |
| open a named semaphore.....                        | 1780                                      |
| open a shared memory object.....                   | 1855                                      |
| open file.....                                     | 67                                        |
| open file description.....                         | 67, 3333                                  |
| open file descriptors.....                         | 3559                                      |
| open file descriptors for reading and writing..... | 2262                                      |
| open mode.....                                     | 2573                                      |
| open().....                                        | <b>1344</b> , 3328, 3381, 3432-3435, 3543 |
| openat.....                                        | 1344                                      |
| openat().....                                      | <b>1355</b>                               |
| opendir.....                                       | 795, 3588                                 |
| opendir().....                                     | <b>1356</b>                               |
| openlog.....                                       | 656                                       |
| openlog().....                                     | <b>1357</b> , 3593                        |
| OPEN_MAX.....                                      | 254, 309, 566, 729, 782                   |
| .....                                              | 795, 932, 1292, 1393, 1396, 2010, 2070    |
| .....                                              | 3315, 3538, 3602-3603                     |
| open_memstream().....                              | <b>1352</b>                               |
| open_wmemstream.....                               | 1352                                      |
| open_wmemstream().....                             | <b>1354</b>                               |
| operand.....                                       | 68                                        |
| operator.....                                      | 68                                        |
| OPOST.....                                         | 394                                       |
| optarg.....                                        | 1004, 1358                                |
| opterr.....                                        | 1004, 1358                                |
| optind.....                                        | 1004, 1358                                |

- option .....68
  - ADV .....7
  - BE .....7
  - CD .....7
  - CPT .....7
  - FD .....7
  - FR .....7
  - FSC .....8
  - IP6 .....8
  - MC1 .....8
  - ML .....8
  - MLR .....8
  - MON .....8
  - MSG .....8
  - MX .....9
  - PIO .....9
  - PS .....9
  - RPI .....9
  - RPP .....9
  - RS .....10
  - SD .....10
  - SHM .....10
  - SIO .....10
  - SPN .....10
  - SS .....10
  - TCT .....10
  - TEF .....11
  - TPI .....11
  - TPP .....11
  - TPS .....11
  - TRC .....11
  - TRI .....11
  - TRL .....11
  - TSA .....11
  - TSH .....12
  - TSP .....12
  - TSS .....12
  - TYM .....12
  - UP .....12
  - UU .....12
  - XSR .....13
- option definitions .....3571
- option-argument .....68
- optional behavior .....3603
- options .....3492
  - shell and utilities .....26
  - system interfaces .....26
- optopt .....1004, 1007, 1358
- optstring .....1007
- OR lists .....2267, 3564
- ordinary identifiers .....2414
- ORD\_CHAR .....177
- orientation .....68
- orphaned process group .....68, 526, 3334, 3414
- output device .....3378
- output devices .....184
- output file descriptor
  - duplication .....3559
- output mode .....3384
- output processing .....3383
- Output\_Path .....2329
- overrun conditions .....3517
- overrun in dumping trace streams .....3517
- overrun in trace streams .....3517
- O\_ .....452
- O\_ constants
  - defined in <fcntl.h> .....224-225
  - used in open() .....1344
  - used in posix\_openpt() .....1383
- O\_ACCMODE .....225, 779
- O\_APPEND .....225, 476, 560, 692, 793, 1344, 2212
- O\_CREAT .....224, 676, 1292-1293, 1304, 1344, 1771  
.....1779, 1853-1854, 1856
- O\_DIRECTORY .....225, 1345
- O\_DSYNC .....225, 551, 1345, 1698, 2213
- O\_EXCL .....224, 1293, 1345, 1779, 1853-1854
- O\_EXEC .....225, 1344
- O\_NDELAY .....2217
- O\_NOCTTY .....224, 1345, 1383
- O\_NOFOLLOW .....225, 1345
- O\_NONBLOCK .....225, 458, 651, 777, 816, 820, 826  
.....877, 881, 908, 910, 997, 1092, 1094, 1293, 1295  
.....1298, 1300, 1345, 1365, 1368, 1680, 1697  
.....2213, 2216
- O\_RDONLY .....225, 699, 1292, 1344, 1853, 1856
- O\_RDWR .....225, 770, 1205, 1292, 1344, 1383  
.....1853, 1856
- O\_RSYNC .....225, 1345, 1698
- O\_SYNC .....225, 551, 1345, 1698, 2213
- O\_TRUNC .....224, 676, 1345, 1854, 1856
- O\_WRONLY .....225, 676, 770, 1205, 1292, 1344
- pack, rationale for omission .....3578
- page .....69, 3334, 3431, 3435
- page size .....69
- PAGESIZE .....254, 478, 1271, 1500, 2012  
.....3431, 3476, 3602
- PAGE\_SIZE .....255, 2012
- paginators
  - more .....2864
- parallel I/O .....3476
- parameter .....69
- parameter expansion .....2254, 3553
- parameters .....3383, 3549
- parameters and variables .....2249
- parameters, positional .....3549

## Index

- parameters, special .....3549
- PARENB.....396
- parent directory .....69, 3334
- parent process .....69
- parent process ID.....69
- PARMRK.....394
- PARODD.....396
- passwd file.....3334
- passwd, rationale for omission.....3578
- paste .....2909, 3592
- patch.....2913, 3592-3593
  - filename determination .....2916
  - patch application .....2916
  - patch file format .....2915
- PATH .....164, 661, 3370
- PATH environment variable .....754
- path prefix .....70
- pathchk .....2920, 3592
- pathconf .....858, 3588
- pathconf().....1359, 3328, 3537, 3587
- pathname .....70
- pathname component .....70
- pathname expansion.....2259, 3557
- pathname manipulation
  - basename .....2403
  - dirname .....2543
  - pathchk .....2920
- pathname resolution .....99, 2231, 3350
- pathname variable values .....256
- pathname, incomplete .....3329
- PATH\_MAX.....257, 264, 459, 539, 565, 615
  - .....629, 632, 636, 750, 767, 788, 795, 850, 858
  - .....895, 916, 920, 927, 937, 964, 1170, 1181, 1254
  - .....1259, 1264, 1335, 1347, 1479, 1708, 1715, 1741
  - .....1749, 2005, 2014, 2085, 2105, 2114, 2234, 2957
  - .....3050, 3404, 3538, 3602
- pattern .....70
- pattern matching .....2669, 2934, 3194, 3210
  - definition .....2278
  - in case statements.....2269
  - in shell variables .....2255
  - multiple character .....3569
  - notation .....3568
  - single character .....3568
- pattern matching notation.....2278, 2674, 2951
- pattern scanning and processing language
  - at.....2371
- patterns
  - filename expansion .....3569
  - patterns matching a single character .....2278
  - patterns matching multiple characters.....2279
  - patterns used for filename expansion .....2279
- pause .....3587
- pause() .....1360, 3413, 3416
- pax .....2925, 3592-3593
  - archive character set encoding/decoding ..2956
  - cpio file data .....2947
  - cpio filename .....2947
  - cpio header .....2945
  - cpio interchange format.....2945
  - cpio special entries .....2947
  - extended header .....2938
  - extended header file times .....2941
  - extended header keyword precedence.....2941
  - list mode format specifications.....2933
  - ustar format.....2942
  - ustar interchange format .....2942
- pcat, rationale for omission.....3578
- pclose() .....1361, 3593
- pd\_ .....450
- PENDIN .....452
- Pending error .....3492
- per-thread errno.....3405
- performance enhancements .....3584
- period .....70
- permissions .....70
- perror().....1363
- persistence .....70
- persistent connection (I\_PLINK).....1096
- PF\_ .....451
- pg, rationale for omission .....3578
- physical write.....923
- ph\_ .....450
- PID\_MAX .....3518
- pid\_t .....380
- PIO .....9
- pipe.....71, 855, 1349, 2216, 3329-3330, 3334, 3588
- pipe().....1365, 3413
- pipelines.....2265, 3562
- PIPE\_BUF .....257, 858, 2213, 2216, 3538, 3602
- PIPE\_MAX.....2217
- plain characters.....1955
- PM\_STR .....250
- pointer to a function.....466
- pointer types .....519, 3519
- pole error .....104
- POLL .....450
- poll().....1367
- POLLERR.....293, 1367
- pollfd .....293
- POLLHUP.....293, 1367
- POLLIN.....293, 1367
- polling .....71
- POLLNVAL .....293, 1368
- POLLOUT.....293, 1367

|                                      |                               |
|--------------------------------------|-------------------------------|
| POLLPRI.....                         | 293, 1367                     |
| POLLRDBAND.....                      | 293, 1367                     |
| POLLRDNORM.....                      | 293, 1367                     |
| POLLWRBAND.....                      | 293, 1367                     |
| POLLWRNORM.....                      | 293, 1367                     |
| POLL_.....                           | 450, 452                      |
| POLL_ERR.....                        | 316                           |
| POLL_HUP.....                        | 316                           |
| POLL_IN.....                         | 316                           |
| POLL_MSG.....                        | 316                           |
| POLL_OUT.....                        | 316                           |
| POLL_PRI.....                        | 316                           |
| popen().....                         | <b>1371</b> , 3590, 3593-3594 |
| portability.....                     | 3318                          |
| portability codes.....               | 3318                          |
| portable character set.....          | 71, 111, 2836, 3354           |
| portable filename character set..... | 71, 3335                      |
| positional parameter.....            | 71                            |
| positional parameters.....           | 2249, 3549                    |
| positive_sign.....                   | 140                           |
| POSIX                                |                               |
| conformance.....                     | 15                            |
| POSIX locale.....                    | 122, 3358                     |
| POSIX shell and utilities.....       | 18                            |
| POSIX system interfaces              |                               |
| conformance.....                     | 16                            |
| POSIX.1 symbols.....                 | 448, 3398                     |
| POSIX.13.....                        | 3435                          |
| POSIX2_BC_BASE_MAX.....              | 2233-2234, 3596               |
| POSIX2_BC_DIM_MAX.....               | 2233-2234, 3596               |
| POSIX2_BC_SCALE_MAX.....             | 2233-2234, 3596               |
| POSIX2_BC_STRING_MAX.....            | 2233-2234, 3596               |
| POSIX2_CHAR_TERM.....                | 18, 26, 3595                  |
| POSIX2_COLL_WEIGHTS_MAX.....         | 2233-2234, 3596               |
| POSIX2_C_BIND.....                   | 3539, 3594                    |
| POSIX2_C_DEV.....                    | 18, 26, 3539, 3594            |
| POSIX2_EXPR_NEST_MAX.....            | 2233-2234, 3596               |
| POSIX2_FORT_DEV.....                 | 18, 27, 3539, 3595            |
| POSIX2_FORT_RUN.....                 | 18, 27, 3539, 3595            |
| POSIX2_LINE_MAX.....                 | 2233-2234, 3596               |
| POSIX2_LOCALEDEF.....                | 18, 27, 3539, 3592, 3595      |
| POSIX2_PBS.....                      | 18, 27, 3595                  |
| POSIX2_PBS_ACCOUNTING.....           | 19, 27, 3595                  |
| POSIX2_PBS_CHECKPOINT.....           | 27, 3595                      |
| POSIX2_PBS_LOCATE.....               | 19, 27, 3595                  |
| POSIX2_PBS_MESSAGE.....              | 19, 27, 3595                  |
| POSIX2_PBS_TRACK.....                | 19, 27, 3595                  |
| POSIX2_RE_DUP_MAX.....               | 2233-2234, 3596               |
| POSIX2_SW_DEV.....                   | 19, 27, 3539, 3594            |
| POSIX2_SYMLINKS.....                 | 858, 2235, 3538               |
| POSIX2_UPE.....                      | 19, 28, 3539, 3594-3595       |
| POSIX2_VERSION.....                  | 3596                          |
| POSIX_.....                          | 449                           |
| posix_.....                          | 449                           |
| POSIX_ALLOC_SIZE_MIN.....            | 257, 858, 3418                |
| POSIX_ASYNCHRONOUS_IO.....           | 3607                          |
| POSIX_BARRIERS.....                  | 3607                          |
| POSIX_CLOCK_SELECTION.....           | 3608                          |
| POSIX_C_LANG_JUMP.....               | 3607                          |
| POSIX_C_LANG_MATH.....               | 3607                          |
| POSIX_C_LANG_SUPPORT.....            | 3608                          |
| POSIX_C_LANG_SUPPORT_R.....          | 3608                          |
| POSIX_C_LANG_WIDE_CHAR.....          | 3608                          |
| POSIX_C_LANG_WIDE_CHAR_EXT.....      | 3608                          |
| POSIX_C_LIB_EXT.....                 | 3608                          |
| POSIX_DEVICE_IO.....                 | 3608                          |
| POSIX_DEVICE_IO_EXT.....             | 3608                          |
| POSIX_DEVICE_SPECIFIC.....           | 3608                          |
| POSIX_DEVICE_SPECIFIC_R.....         | 3609                          |
| POSIX_DYNAMIC_LINKING.....           | 3609                          |
| posix_fadvise().....                 | <b>1374</b> , 3418            |
| POSIX_FADV_DONTNEED.....             | 226, 1374, 3418               |
| POSIX_FADV_NOREUSE.....              | 226, 1374, 3418               |
| POSIX_FADV_NORMAL.....               | 226, 1374                     |
| POSIX_FADV_RANDOM.....               | 226, 1374, 3418               |
| POSIX_FADV_SEQUENTIAL.....           | 226, 1374, 3418               |
| POSIX_FADV_WILLNEED.....             | 226, 1374, 3418               |
| posix_fallocate().....               | <b>1376</b>                   |
| POSIX_FD_MGMT.....                   | 3609                          |
| POSIX_FIFO.....                      | 3609                          |
| POSIX_FIFO_FD.....                   | 3609                          |
| POSIX_FILE_ATTRIBUTES.....           | 3609                          |
| POSIX_FILE_ATTRIBUTES_FD.....        | 3609                          |
| POSIX_FILE_LOCKING.....              | 3609                          |
| POSIX_FILE_SYSTEM.....               | 3609                          |
| POSIX_FILE_SYSTEM_EXT.....           | 3609                          |
| POSIX_FILE_SYSTEM_FD.....            | 3609                          |
| POSIX_FILE_SYSTEM_GLOB.....          | 3609                          |
| POSIX_FILE_SYSTEM_R.....             | 3609                          |
| POSIX_I18N.....                      | 3609                          |
| POSIX_JOB_CONTROL.....               | 3609                          |
| posix_madvise().....                 | <b>1378</b> , 3418            |
| POSIX_MADV_DONTNEED.....             | 352, 1378, 3418               |
| POSIX_MADV_NORMAL.....               | 352, 1378                     |
| POSIX_MADV_RANDOM.....               | 352, 1378, 3418               |
| POSIX_MADV_SEQUENTIAL.....           | 352, 1378, 3418               |
| POSIX_MADV_WILLNEED.....             | 353, 1378, 3418               |
| POSIX_MAPPED_FILES.....              | 3609                          |
| posix_memalign().....                | <b>1382</b>                   |
| POSIX_MEMORY_PROTECTION.....         | 3609                          |
| posix_mem_offset().....              | <b>1380</b> , 3435-3436       |
| POSIX_MULTI_CONCURRENT_LOCALES.....  | 3609                          |
| POSIX_MULTI_PROCESS.....             | 3610                          |
| POSIX_MULTI_PROCESS_FD.....          | 3610                          |
| POSIX_NETWORKING.....                | 3610                          |



## Index

|                                        |                  |
|----------------------------------------|------------------|
| posix_openpt()                         | 1383             |
| POSIX_PIPE                             | 3610             |
| POSIX_REALTIME_SIGNALS                 | 3610             |
| POSIX_REC_INCR_XFER_SIZE               | 257, 858, 3419   |
| POSIX_REC_MAX_XFER_SIZE                | 257, 858, 3419   |
| POSIX_REC_MIN_XFER_SIZE                | 257, 858, 3419   |
| POSIX_REC_XFER_ALIGN                   | 257, 858, 3418   |
| POSIX_REGEX                            | 3610             |
| POSIX_ROBUST_MUTEXES                   | 3610             |
| POSIX_RW_LOCKS                         | 3610             |
| POSIX_SEMAPHORES                       | 3610             |
| POSIX_SHELL_FUNC                       | 3610             |
| POSIX_SIGNALS                          | 3610             |
| POSIX_SIGNALS_EXT                      | 3610             |
| POSIX_SIGNAL_JUMP                      | 3610             |
| POSIX_SINGLE_PROCESS                   | 3611             |
| posix_spawn()                          | 1385, 3522, 3587 |
| posix_spawnattr_destroy()              | 1401             |
| posix_spawnattr_getflags()             | 1403             |
| posix_spawnattr_getpgroup()            | 1405             |
| posix_spawnattr_getschedparam()        | 1407             |
| posix_spawnattr_getschedpolicy()       | 1409             |
| posix_spawnattr_getsigdefault()        | 1411             |
| posix_spawnattr_getsigmask()           | 1413             |
| posix_spawnattr_init                   | 1401             |
| posix_spawnattr_init()                 | 1415             |
| posix_spawnattr_setflags               | 1403             |
| posix_spawnattr_setflags()             | 1416             |
| posix_spawnattr_setpgroup              | 1405             |
| posix_spawnattr_setpgroup()            | 1417             |
| posix_spawnattr_setschedparam          | 1407             |
| posix_spawnattr_setschedparam()        | 1418             |
| posix_spawnattr_setschedpolicy         | 1409             |
| posix_spawnattr_setschedpolicy()       | 1419             |
| posix_spawnattr_setsigdefault          | 1411             |
| posix_spawnattr_setsigdefault()        | 1420             |
| posix_spawnattr_setsigmask             | 1413             |
| posix_spawnattr_setsigmask()           | 1421             |
| posix_spawnnp                          | 1385             |
| posix_spawnnp()                        | 1422, 3522, 3587 |
| posix_spawn_file_actions_addclose()    | 1393             |
| posix_spawn_file_actions_adddup2()     | 1396             |
| posix_spawn_file_actions_addopen       | 1393             |
| posix_spawn_file_actions_addopen()     | 1398             |
| posix_spawn_file_actions_destroy()     | 1399             |
| posix_spawn_file_actions_init          | 1399             |
| POSIX_SPAWN_RESETIDS                   | 1386, 1403       |
| POSIX_SPAWN_SETPGROUP                  | 1386, 1403, 1405 |
| POSIX_SPAWN_SETSCHEDPARAM              | 1403, 1407       |
| POSIX_SPAWN_SETSCHEDULER               | 1386, 1403       |
| .....                                  | 1407, 1409       |
| POSIX_SPAWN_SETSIGDEF                  | 1387, 1403, 1411 |
| POSIX_SPAWN_SETSIGMASK                 | 1403, 1413       |
| POSIX_SPIN_LOCKS                       | 3611             |
| POSIX_SYMBOLIC_LINKS                   | 3611             |
| POSIX_SYMBOLIC_LINKS_FD                | 3611             |
| POSIX_SYSTEM_DATABASE                  | 3611             |
| POSIX_SYSTEM_DATABASE_R                | 3611             |
| POSIX_THREADS_BASE                     | 3611             |
| POSIX_THREADS_EXT                      | 3611             |
| POSIX_TIMERS                           | 3611             |
| POSIX_TRACE_ADD_EVENTSET               | 1462             |
| POSIX_TRACE_ALL_EVENTS                 | 1455             |
| POSIX_TRACE_APPEND                     | 1428, 1448       |
| posix_trace_attr_destroy()             | 1423             |
| posix_trace_attr_getclockres()         | 1425             |
| posix_trace_attr_getcreatetime         | 1425             |
| posix_trace_attr_getgenversion         | 1425             |
| posix_trace_attr_getinherited()        | 1427             |
| posix_trace_attr_getlogfullpolicy      | 1427             |
| posix_trace_attr_getlogsize()          | 1430             |
| posix_trace_attr_getmaxdatasize        | 1430             |
| posix_trace_attr_getmaxsystemevents    | 1430             |
| posix_trace_attr_getmaxuserevents      | 1430             |
| posix_trace_attr_getname               | 1425             |
| posix_trace_attr_getname()             | 1433             |
| posix_trace_attr_getstreamfullpolicy   | 1427             |
| posix_trace_attr_getstreamfullpolicy() | 1434             |
| posix_trace_attr_getstreamsize         | 1430             |
| posix_trace_attr_getstreamsize()       | 1435             |
| posix_trace_attr_init                  | 1423             |
| posix_trace_attr_init()                | 1436             |
| posix_trace_attr_setinherited          | 1427             |
| posix_trace_attr_setinherited()        | 1437             |
| posix_trace_attr_setlogfullpolicy      | 1427, 1437       |
| posix_trace_attr_setlogsize            | 1430             |
| posix_trace_attr_setlogsize()          | 1438             |
| posix_trace_attr_setmaxdatasize        | 1430, 1438       |
| posix_trace_attr_setname               | 1425             |
| posix_trace_attr_setname()             | 1439             |
| posix_trace_attr_setstreamfullpolicy   | 1427             |
| posix_trace_attr_setstreamfullpolicy() | 1440             |
| posix_trace_attr_setstreamsize         | 1430             |
| posix_trace_attr_setstreamsize()       | 1441             |
| posix_trace_clear()                    | 1442             |
| posix_trace_close()                    | 1444             |
| POSIX_TRACE_CLOSE_FOR_CHILD            | 1427             |
| posix_trace_create()                   | 1446             |
| posix_trace_create_withlog             | 1446             |
| POSIX_TRACE_ERROR trace event          | 514              |
| posix_trace_event()                    | 1450             |
| posix_trace_eventid_equal()            | 1452             |
| posix_trace_eventid_get_name           | 1452             |
| posix_trace_eventid_open               | 1450             |
| posix_trace_eventid_open()             | 1454, 3512       |

|                                            |                                  |
|--------------------------------------------|----------------------------------|
| posix_trace_eventset_add()                 | 1455                             |
| posix_trace_eventset_del                   | 1455                             |
| posix_trace_eventset_empty                 | 1455                             |
| posix_trace_eventset_fill                  | 1455                             |
| posix_trace_eventset_ismember              | 1455                             |
| posix_trace_eventtypelist_getnext_id()     | 1457                             |
| posix_trace_eventtypelist_rewind           | 1457                             |
| posix_trace_event_info structure           | 511                              |
| POSIX_TRACE_FILTER trace event             | 514, 1462                        |
| POSIX_TRACE_FLUSH                          | 1428                             |
| posix_trace_flush                          | 1446                             |
| posix_trace_flush()                        | 1459                             |
| POSIX_TRACE_FLUSHING                       | 510                              |
| POSIX_TRACE_FULL                           | 509-511                          |
| posix_trace_getnext_event()                | 1465                             |
| posix_trace_get_attr()                     | 1460                             |
| posix_trace_get_filter()                   | 1462                             |
| posix_trace_get_status                     | 1460                             |
| posix_trace_get_status()                   | 1464                             |
| POSIX_TRACE_INHERITED                      | 1427                             |
| POSIX_TRACE_LOOP                           | 510, 1427-1428, 1447, 3517       |
| POSIX_TRACE_NOT_FLUSHING                   | 511                              |
| POSIX_TRACE_NOT_FULL                       | 509-511                          |
| POSIX_TRACE_NOT_FULL                       | 1442                             |
| POSIX_TRACE_NOT_TRUNCATED                  | 512, 1466                        |
| POSIX_TRACE_NO_OVERRUN                     | 510-511, 1460                    |
| posix_trace_open                           | 1444                             |
| posix_trace_open()                         | 1468                             |
| POSIX_TRACE_OVERFLOW trace event           | 514                              |
| POSIX_TRACE_OVERRUN                        | 510-511                          |
| POSIX_TRACE_RESUME trace event             | 514                              |
| posix_trace_rewind                         | 1444, 1468                       |
| POSIX_TRACE_RUNNING                        | 509-510, 1471                    |
| POSIX_TRACE_SET_EVENTSET                   | 1462                             |
| posix_trace_set_filter                     | 1462                             |
| posix_trace_set_filter()                   | 1469                             |
| posix_trace_shutdown                       | 1446                             |
| posix_trace_shutdown()                     | 1470                             |
| POSIX_TRACE_START trace event              | 514, 1471                        |
| posix_trace_start()                        | 1471                             |
| posix_trace_status_info structure          | 509                              |
| posix_trace_stop                           | 1471                             |
| POSIX_TRACE_STOP trace event               | 514, 1471                        |
| POSIX_TRACE_SUB_EVENTSET                   | 1462                             |
| POSIX_TRACE_SUSPENDED                      | 509-510, 1471                    |
| POSIX_TRACE_SYSTEM_EVENTS                  | 1455                             |
| posix_trace_timedgetnext_event             | 1465                             |
| posix_trace_timedgetnext_event()           | 1473                             |
| posix_trace_trid_eventid_open              | 1452                             |
| posix_trace_trid_eventid_open()            | 1474                             |
| POSIX_TRACE_TRUNCATED_READ                 | 512, 1466                        |
| POSIX_TRACE_TRUNCATED_RECORD               | 512                              |
|                                            | 1466                             |
| posix_trace_trygetnext_event               | 1465                             |
| posix_trace_trygetnext_event()             | 1475                             |
| POSIX_TRACE_UNTIL_FULL                     | 510                              |
|                                            | 1427-1428, 1447                  |
| POSIX_TRACE_USER_EVENT_MAX                 | 1450                             |
| POSIX_TRACE_WOPID_EVENTS                   | 1455                             |
| POSIX_TYPED_MEM_ALLOCATE                   | 353                              |
|                                            | 1275-1276                        |
|                                            | 1380, 1476, 1478                 |
| POSIX_TYPED_MEM_ALLOCATE_CONTIG            | 353, 1275-1276, 1380, 1476, 1478 |
| posix_typed_mem_get_info()                 | 1476, 3435                       |
| POSIX_TYPED_MEM_MAP_ALLOCATABLE            | 353, 1322, 1478                  |
| posix_typed_mem_open()                     | 1478, 3435                       |
| POSIX_USER_GROUPS                          | 3611                             |
| POSIX_USER_GROUPS_R                        | 3611                             |
| POSIX_VERSION                              | 3602                             |
| POSIX_WIDE_CHAR_DEVICE_IO                  | 3611                             |
| post-mortem filtering of trace event types | 3514                             |
| pow()                                      | 1481                             |
| powf                                       | 1481                             |
| powl                                       | 1481                             |
| pr                                         | 2961, 3592-3593                  |
| pread                                      | 1697                             |
| pread()                                    | 1484, 3477                       |
| preallocation                              | 71                               |
| predefined stream                          |                                  |
| standard error                             | 472                              |
| standard input                             | 472                              |
| standard output                            | 472                              |
| preempted process (or thread)              | 72                               |
| preempted thread                           | 1550                             |
| previous job                               | 72                               |
| PRI                                        | 452                              |
| print-related commands                     |                                  |
| fold                                       | 2677                             |
| lp                                         | 2783                             |
| pr                                         | 2961                             |
| printable character                        | 72                               |
| printable file                             | 72                               |
| printf                                     | 864, 2966, 3591-3592             |
| printf()                                   | 1485                             |
| printing                                   | 3586                             |
| priority                                   | 72, 474                          |
| Priority                                   | 2330                             |
| priority band                              | 72                               |
| priority inversion                         | 72                               |
| priority scheduling                        | 72                               |
| priority-based scheduling                  | 73                               |
| PRIO_                                      | 450                              |

## Index

|                                                  |                            |
|--------------------------------------------------|----------------------------|
| PRIO_constants                                   |                            |
| defined in <sys/resource.h>.....                 | 357                        |
| PRIO_INHERIT .....                               | 1604                       |
| PRIO_PGRP.....                                   | 357, 1016                  |
| PRIO_PROCESS.....                                | 357, 1016                  |
| PRIO_USER.....                                   | 357, 1016                  |
| privilege.....                                   | 73, 3346                   |
| privileges.....                                  | 2856, 2887                 |
| process.....                                     | 73                         |
| concurrent execution.....                        | 855                        |
| setting real and effective user IDs.....         | 1837                       |
| single-threaded.....                             | 855                        |
| process attributes.....                          | 2227                       |
| process creation.....                            | 855                        |
| process group.....                               | 73, 3380                   |
| orphaned.....                                    | 526                        |
| termios.....                                     | 185                        |
| process group ID.....                            | 73, 1012, 1829, 1841       |
| .....                                            | 2227, 3329-3330, 3380      |
| process group leader.....                        | 73                         |
| process group lifetime.....                      | 73, 3380                   |
| process group, orphaned.....                     | 3334, 3414                 |
| process groups, concepts in job control.....     | 3329                       |
| process ID.....                                  | 74, 2227                   |
| process ID reuse.....                            | 100, 3351                  |
| process ID, 1.....                               | 526                        |
| process ID, rationale.....                       | 3518                       |
| process lifetime.....                            | 74, 1165, 3335             |
| process management.....                          | 3584, 3587                 |
| process memory locking.....                      | 74                         |
| process scheduling.....                          | 479, 3441, 3588            |
| process shared memory.....                       | 1609                       |
| process status report.....                       | 2976                       |
| process synchronization.....                     | 1609                       |
| process termination.....                         | 74, 525, 3335              |
| process virtual time.....                        | 74                         |
| process-to-process communication.....            | 74                         |
| prof, rationale for omission.....                | 3578                       |
| profiling.....                                   | 3594                       |
| program.....                                     | 75                         |
| programming manipulation.....                    | 3507                       |
| prompting.....                                   | 3551                       |
| protocol.....                                    | 75                         |
| protocols.....                                   | 3491                       |
| PROT.....                                        | 450                        |
| PROT_EXEC.....                                   | 352, 1276, 1285            |
| PROT_NONE.....                                   | 352, 478, 1275-1276, 1285  |
| PROT_READ.....                                   | 352, 1276, 1285            |
| PROT_READ constants                              |                            |
| in <sys/mman.h>.....                             | 352                        |
| PROT_WRITE.....                                  | 352, 1276-1277, 1280, 1285 |
| prs.....                                         | 2971                       |
| PS.....                                          | 9                          |
| ps.....                                          | 2976, 3591, 3593           |
| pselect().....                                   | 1486                       |
| pseudo-random sequence generation functions..... | 1694                       |
| pseudo-terminal.....                             | 75                         |
| psiginfo().....                                  | 1491                       |
| psignal.....                                     | 1491                       |
| psignal().....                                   | 1492                       |
| PST8PDT.....                                     | 2092                       |
| ps.....                                          | 450                        |
| pthread.....                                     | 3515                       |
| PTHREAD.....                                     | 450                        |
| pthread.....                                     | 450                        |
| pthread_atfork().....                            | 1493                       |
| pthread_attr_destroy().....                      | 1495                       |
| pthread_attr_getdetachstate().....               | 1498                       |
| pthread_attr_getguardsize().....                 | 1500, 3476                 |
| pthread_attr_getinheritsched().....              | 1503                       |
| pthread_attr_getschedparam().....                | 1505                       |
| pthread_attr_getschedpolicy().....               | 1507                       |
| pthread_attr_getscope().....                     | 1509                       |
| pthread_attr_getstack().....                     | 1511                       |
| pthread_attr_getstackaddr.....                   | 3521                       |
| pthread_attr_getstacksize().....                 | 1513                       |
| pthread_attr_init.....                           | 1495                       |
| pthread_attr_init().....                         | 1515                       |
| pthread_attr_setdetachstate.....                 | 1498                       |
| pthread_attr_setdetachstate().....               | 1516                       |
| pthread_attr_setguardsize.....                   | 1500                       |
| pthread_attr_setguardsize().....                 | 1517, 3476                 |
| pthread_attr_setinheritsched.....                | 1503                       |
| pthread_attr_setinheritsched().....              | 1518                       |
| pthread_attr_setschedparam.....                  | 1505                       |
| pthread_attr_setschedparam().....                | 1519                       |
| pthread_attr_setschedpolicy.....                 | 1507                       |
| pthread_attr_setschedpolicy().....               | 1520                       |
| pthread_attr_setscope.....                       | 1509                       |
| pthread_attr_setscope().....                     | 1521                       |
| pthread_attr_setstack.....                       | 1511                       |
| pthread_attr_setstack().....                     | 1522                       |
| pthread_attr_setstackaddr.....                   | 3521                       |
| pthread_attr_setstacksize.....                   | 1513                       |
| pthread_attr_setstacksize().....                 | 1523                       |
| pthread_barrierattr_destroy().....               | 1528                       |
| pthread_barrierattr_getpshared().....            | 1530                       |
| pthread_barrierattr_init.....                    | 1528                       |
| pthread_barrierattr_init().....                  | 1532                       |
| pthread_barrierattr_setpshared.....              | 1530                       |
| pthread_barrierattr_setpshared().....            | 1533                       |
| pthread_barrier_destroy().....                   | 1524                       |
| pthread_barrier_init.....                        | 1524                       |
| PTHREAD_BARRIER_SERIAL_THREAD.....               | 295                        |

- .....1526, 3467
- pthread\_barrier\_wait().....**1526**, 3468, 3487
- pthread\_cancel().....**1534**
- PTHREAD\_CANCELED.....295, 494, 1570
- PTHREAD\_CANCEL\_ASYNCHRONOUS .....295  
.....490, 1655
- PTHREAD\_CANCEL\_DEFERRED .....295, 490  
.....494, 748, 1548, 1655
- PTHREAD\_CANCEL\_DISABLE .....295  
.....490, 494, 1655
- PTHREAD\_CANCEL\_ENABLE .....295  
.....490, 494, 1655
- PTHREAD\_CANCEL\_ENABLED .....748
- pthread\_cleanup\_pop().....**1536**
- pthread\_cleanup\_push .....1536
- pthread\_condattr\_destroy() .....**1554**
- pthread\_condattr\_getclock() .....**1556**
- pthread\_condattr\_getpshared() .....**1558**
- pthread\_condattr\_init .....1554
- pthread\_condattr\_init().....**1560**
- pthread\_condattr\_setclock .....1556
- pthread\_condattr\_setclock().....**1561**
- pthread\_condattr\_setpshared .....1558
- pthread\_condattr\_setpshared() .....**1562**
- pthread\_cond\_broadcast().....**1541**
- pthread\_cond\_destroy().....**1544**
- pthread\_cond\_init .....1544
- pthread\_cond\_init().....3464
- PTHREAD\_COND\_INITIALIZER .....295, 1544
- pthread\_cond\_signal .....1541
- pthread\_cond\_signal() .....**1547**
- pthread\_cond\_timedwait().....**1548**, 3406  
.....3451, 3473, 3597
- pthread\_cond\_wait .....1548
- pthread\_cond\_wait() .....3406, 3423, 3473
- pthread\_create().....**1563**, 3464-3465
- PTHREAD\_CREATE\_DETACHED .....295, 465  
.....1498, 3485
- PTHREAD\_CREATE\_JOINABLE .....295, 465  
.....748, 1498, 1580
- PTHREAD\_DESTRUCTOR\_ITERATIONS .....255  
.....1577, 1582, 2012, 3602
- pthread\_detach().....**1566**, 3485
- pthread\_equal().....**1568**
- pthread\_exit() .....**1569**
- PTHREAD\_EXPLICIT\_SCHED.....295, 1503
- pthread\_getconcurrency() .....**1571**, 3476
- pthread\_getcpuclockid() .....**1573**, 3454-3455
- pthread\_getschedparam() .....**1574**
- pthread\_getspecific() .....**1577**
- PTHREAD\_INHERIT\_SCHED.....295, 1503
- pthread\_join() .....**1579**, 3406, 3485
- PTHREAD\_KEYS\_MAX.....255, 1582, 2012, 3602
- pthread\_key\_create().....**1582**, 3467
- pthread\_key\_delete().....**1585**
- pthread\_kill().....**1587**
- pthread\_mutexattr\_destroy().....**1608**
- pthread\_mutexattr\_getprioceiling().....**1613**
- pthread\_mutexattr\_getprotocol() .....**1615**
- pthread\_mutexattr\_getpshared() .....**1618**
- pthread\_mutexattr\_getrobust() .....**1620**
- pthread\_mutexattr\_gettype().....**1622**, 3474
- pthread\_mutexattr\_init.....1608
- pthread\_mutexattr\_init() .....**1624**
- pthread\_mutexattr\_setprioceiling.....1613
- pthread\_mutexattr\_setprioceiling() .....**1625**
- pthread\_mutexattr\_setprotocol.....1615
- pthread\_mutexattr\_setprotocol().....**1626**
- pthread\_mutexattr\_setpshared .....1618
- pthread\_mutexattr\_setpshared() .....**1627**
- pthread\_mutexattr\_setrobust .....1620
- pthread\_mutexattr\_setrobust() .....**1628**
- pthread\_mutexattr\_settype.....1622
- pthread\_mutexattr\_settype() .....**1629**, 3474
- pthread\_mutex\_consistent().....**1589**
- PTHREAD\_MUTEX\_DEFAULT .....295, 1599  
.....1622, 3473
- pthread\_mutex\_destroy() .....**1591**
- PTHREAD\_MUTEX\_ERRORCHECK .....295, 1599  
.....1622, 3473
- pthread\_mutex\_getprioceiling().....**1596**
- pthread\_mutex\_init .....1591
- pthread\_mutex\_init() .....**1598**, 3464
- PTHREAD\_MUTEX\_INITIALIZER .....295, 1591
- pthread\_mutex\_lock() .....**1599**, 3406, 3473, 3487
- PTHREAD\_MUTEX\_NORMAL .....295, 1599  
.....1622, 3473
- PTHREAD\_MUTEX\_RECURSIVE .....98, 295, 1596  
.....1599, 1622-1623, 3473
- PTHREAD\_MUTEX\_ROBUST .....1620
- pthread\_mutex\_setprioceiling.....1596
- pthread\_mutex\_setprioceiling() .....**1603**
- PTHREAD\_MUTEX\_STALLED .....1620
- pthread\_mutex\_timedlock().....**1604**, 3452
- pthread\_mutex\_trylock .....1599
- pthread\_mutex\_trylock().....**1607**, 3473
- pthread\_mutex\_unlock.....1599, 1607
- pthread\_mutex\_unlock() .....3473
- pthread\_once() .....**1630**
- PTHREAD\_ONCE\_INIT .....295, 1630
- PTHREAD\_PRIO\_INHERIT .....295, 1615
- PTHREAD\_PRIO\_NONE .....295, 1596, 1615
- PTHREAD\_PRIO\_PROTECT .....295, 1600, 1615
- PTHREAD\_PROCESS\_PRIVATE ..295, 1530, 1558  
.....1609, 1618, 1650, 1666, 3475

## Index

- PTHREAD\_PROCESS\_SHARED...295, 1530, 1558  
     .....1609, 1618, 1650, 1666, 3475  
 pthread\_rwlockattr\_destroy().....**1648**, 3475  
 pthread\_rwlockattr\_getpshared().....**1650**, 3475  
 pthread\_rwlockattr\_init.....1648  
 pthread\_rwlockattr\_init().....**1652**, 3474  
 pthread\_rwlockattr\_setpshared.....1650  
 pthread\_rwlockattr\_setpshared().....**1653**, 3475  
 pthread\_rwlock\_destroy().....**1632**  
 pthread\_rwlock\_init.....1632  
 pthread\_rwlock\_init().....3475  
 PTHREAD\_RWLOCK\_INITIALIZER.....295, 3475  
 pthread\_rwlock\_rdlock().....**1635**, 3475  
 pthread\_rwlock\_t.....3474  
 pthread\_rwlock\_timedrdlock().....**1638**  
 pthread\_rwlock\_timedwrlock().....**1640**  
 pthread\_rwlock\_tryrdlock.....1635  
 pthread\_rwlock\_tryrdlock().....**1642**, 3475  
 pthread\_rwlock\_trywrlock().....**1643**, 3475  
 pthread\_rwlock\_unlock().....**1645**, 3475, 3489  
 pthread\_rwlock\_wrlock.....1643  
 pthread\_rwlock\_wrlock().....**1647**, 3475  
 PTHREAD\_SCOPE\_PROCESS.....295, 488, 1509  
 PTHREAD\_SCOPE\_SYSTEM.....295, 488, 1509  
 pthread\_self().....**1654**, 3466  
 pthread\_setcancelstate().....**1655**  
 pthread\_setcanceltype.....1655  
 pthread\_setconcurrency.....1571  
 pthread\_setconcurrency().....**1657**, 3476  
 pthread\_setprio().....3484  
 pthread\_setschedparam.....1574  
 pthread\_setschedparam().....**1658**, 3484  
 pthread\_setschedprio().....**1659**  
 pthread\_setspecific.....1577  
 pthread\_setspecific().....**1661**, 3466  
 pthread\_sigmask().....**1662**  
 pthread\_spin\_destroy().....**1666**  
 pthread\_spin\_init.....1666  
 pthread\_spin\_lock().....**1668**, 3468, 3487  
 pthread\_spin\_trylock.....1668  
 pthread\_spin\_trylock().....3468  
 pthread\_spin\_unlock().....**1670**  
 PTHREAD\_STACK\_MIN.....255, 1511  
     .....1513, 2012, 3602  
 pthread\_testcancel.....1655  
 pthread\_testcancel().....**1671**  
 PTHREAD\_THREADS\_MAX.....255, 1563  
     .....2012, 3602  
 PTRDIFF\_MAX.....331  
 PTRDIFF\_MIN.....331  
 ptsname().....**1672**  
 public locale.....2769  
 putc.....3588  
 putc().....**1673**, 3478  
 putchar.....3588  
 putchar().....**1675**  
 putchar\_unlocked.....959  
 putchar\_unlocked().....**1676**  
 putc\_unlocked.....959  
 putc\_unlocked().....**1674**  
 putenv().....**1677**  
 putmsg().....**1679**  
 putpmsg.....1679  
 puts().....**1683**  
 pututxline.....733  
 pututxline().....**1685**  
 putwc().....**1686**  
 putwchar().....**1687**  
 PWD.....**164**  
 pwd.....2244, 2264, 2983, 3592  
 pwrite.....2212  
 pwrite().....**1688**, 3477  
 pw.....450  
 p.....450  
 P.....451  
 P\_ALL.....387, 2139  
 p\_cs\_precedes.....140  
 P\_GID.....387  
 P\_PGID.....2139  
 P\_PID.....387, 2139  
 p\_sep\_by\_space.....140  
 p\_sign\_posn.....141  
 P\_tmpdir.....335  
 qalter.....**2986**  
 qdel.....**2995**  
 qhold.....**2998**  
 qmove.....**3001**  
 qmsg.....**3004**  
 qrerun.....**3007**  
 qrls.....**3009**  
 qselect.....**3012**  
 qsig.....**3020**  
 qsort().....**1689**  
 qstat.....**3023**  
 qsub.....**3028**  
 queue a signal to a process.....1898  
 Queue Batch Job Request.....2336  
 queuing of waiting threads.....3489  
 quiet NaN.....232  
 quote removal.....2259, 3557  
 QUOTED\_CHAR.....177  
 quoting.....2246, 3546  
 radix character.....75  
 RADIXCHAR.....250  
 raise().....**1691**

- rand() .....1693, 3487
- random.....1083
- random() .....1696
- RAND\_MAX .....338, 1693
- rand\_r.....1693
- range error .....104
  - result overflows .....104
  - result underflows .....104
- RCS, rationale for omission.....3578
- RE
  - grammar .....3378
- RE bracket expression.....3373
- read .....2244, 2264, 3040, 3545, 3588, 3591
- read from a file.....1700
- read lock.....3474
- read() .....1697, 3329, 3381-3382, 3404
  - .....3413-3414, 3416, 3425-3427, 3431-3432, 3476
  - .....3487, 3519
- read-only file system.....75
- read-write attributes .....3474
- read-write lock.....75
- read-write locks .....3474
- readdir.....3588
- readdir() .....1704
- readdir\_r .....1704
- reading an active trace stream.....3517
- reading data .....3382
- readlink.....3588
- readlink().....1708, 3338
- readlinkat.....1708
- readonly .....2298
- readv() .....1711
- real group ID.....75, 2227
- real time .....76
- real user ID .....76, 540, 1164, 2227
- realloc().....1713
- realpath.....3588
- realpath().....1715
- realtime .....22
- REALTIME .....206, 278, 787, 1271, 1273, 1287-1288
  - .....1290, 1292, 1295, 1298, 1300, 1304, 1758-1762
  - .....1764, 1853, 1858
- realtime .....3418
- realtime signal delivery .....3410
- realtime signal extension.....76
- realtime signal generation.....3410
- realtime signals .....3423
- realtime threads .....23
- REALTIME THREADS .23, 1503, 1507, 1509, 1518
  - .....1520-1521, 1574, 1596, 1603, 1613, 1615
  - .....1625-1626, 1658-1659
- record .....76
- recv().....1718
- recvfrom() .....1720
- recvmsg().....1723
- red, rationale for omission .....3578
- redirect input.....3558
- redirect output .....3559
- redirecting input.....2260
- redirecting output.....2260
- redirection.....76, 2259, 3557
- redirection operator .....76
- reentrant function.....76
- referenced shared memory object.....76
- references .....3316
- refresh.....76
- regcomp() .....1726, 3593
- regerror .....1726
- regerror() .....3593
- regexec .....1726
- regexec() .....3593
- regfree .....1726
- regfree() .....3593
- region .....77
- register fork handlers.....1493
- REGTYPE.....391
- regular expression .....77, 3371
  - basic.....169
  - definitions .....3371
  - extended .....174
  - general requirements .....3372
  - grammar .....177, 3377
- regular expressions .....2378-2379, 2486, 2554, 2579
  - .....2608, 2647, 2671, 2711, 2755, 2868, 2878, 2889
  - .....2932, 3048, 3067, 3231, 3234, 3285
- related to shell patterns .....2278
- regular file .....77, 3335
- REG .....450
- REG\_constants
  - defined in <regex.h> .....303
  - error return values of regcomp .....1728
  - used in regcomp .....1726
- REG\_BADBR.....304, 1728
- REG\_BADPAT.....303, 1728
- REG\_BADRPT .....304, 1728
- REG\_EBRACE.....304, 1728
- REG\_EBRACK .....303, 1728
- REG\_ECOLLATE.....303, 1728
- REG\_ECTYPE .....303, 1728
- REG\_EESCAPE.....303, 1728
- REG\_EPAREN.....303, 1728
- REG\_ERANGE.....304, 1728
- REG\_ESPACE.....304, 1728
- REG\_ESUBREG .....303, 1728
- REG\_EXTENDED.....303, 1726

## Index

|                                   |                            |
|-----------------------------------|----------------------------|
| REG_ICASE.....                    | 303, 1726                  |
| REG_NEWLINE.....                  | 303, 1726                  |
| REG_NOMATCH.....                  | 303, 1728                  |
| REG_NOSUB.....                    | 303, 1726                  |
| REG_NOTBOL.....                   | 303, 1727                  |
| REG_NOTEOL.....                   | 303, 1727                  |
| rejected utilities.....           | 3576                       |
| relational database operator..... | 2741                       |
| relative pathname.....            | 77, 99                     |
| Release Batch Job Request.....    | 2337                       |
| relocatable file.....             | 77                         |
| relocation.....                   | 77                         |
| remainder().....                  | <b>1733</b>                |
| remainderf.....                   | 1733                       |
| remainderl.....                   | 1733                       |
| remove a directory.....           | 1750                       |
| remove directories.....           | 3054                       |
| remove directory entries.....     | 2107                       |
| remove files.....                 | 3047                       |
| remove().....                     | <b>1735</b> , 3338         |
| remque.....                       | 1085                       |
| remque().....                     | <b>1737</b>                |
| remquo().....                     | <b>1738</b>                |
| remquof.....                      | 1738                       |
| remquol.....                      | 1738                       |
| rename.....                       | 3588                       |
| rename a file.....                | 1743                       |
| rename().....                     | <b>1740</b> , 3338         |
| renameat.....                     | 1740                       |
| renice.....                       | <b>3043</b> , 3591         |
| replenishment period.....         | 3444                       |
| requested batch service.....      | 77                         |
| requested batch services.....     | 2333                       |
| requirements.....                 | 15                         |
| Rerun Batch Job Request.....      | 2338                       |
| Rerunable.....                    | 2330                       |
| reserved words.....               | 2249, 3549                 |
| Resource_List.....                | 2330                       |
| result overflows.....             | 104                        |
| result underflows.....            | 104                        |
| return.....                       | <b>2300</b>                |
| rewind().....                     | <b>1745</b>                |
| rewinddir.....                    | 3588                       |
| rewinddir().....                  | <b>1746</b>                |
| re.....                           | 450                        |
| RE_DUP_MAX.....                   | 255, 258, 2012, 2234, 3538 |
| rindex.....                       | 3522                       |
| rint().....                       | <b>1747</b>                |
| rintf.....                        | 1747                       |
| rintl.....                        | 1747                       |
| rlimit.....                       | <b>357</b>                 |
| RLIMIT_.....                      | 450                        |
| RLIMIT_AS.....                    | 358, 1030                  |
| RLIMIT_CORE.....                  | 357, 1029                  |
| RLIMIT_CPU.....                   | 357, 1029                  |
| RLIMIT_DATA.....                  | 358, 1029                  |
| RLIMIT_FSIZE.....                 | 358, 1029                  |
| RLIMIT_NOFILE.....                | 358, 1029, 1031            |
| RLIMIT_STACK.....                 | 358, 1030                  |
| rlim_.....                        | 450                        |
| RLIM_.....                        | 452                        |
| RLIM_INFINITY.....                | 357, 1029-1030             |
| RLIM_SAVED_CUR.....               | 357, 1030                  |
| RLIM_SAVED_MAX.....               | 357, 1030                  |
| rm.....                           | <b>3047</b> , 3543, 3592   |
| rmdel.....                        | <b>3052</b>                |
| rmdir.....                        | <b>3054</b> , 3588, 3592   |
| rmdir().....                      | <b>1749</b> , 3338, 3405   |
| RMSGD.....                        | 347, 1091                  |
| RMSGN.....                        | 347, 1091                  |
| RNORM.....                        | 347, 1091                  |
| robust mutex.....                 | 78                         |
| robust mutexes.....               | 487, 1595, 3470            |
| root directory.....               | 78, 2227, 3335, 3351       |
| root file system.....             | 3335                       |
| root of a file system.....        | 3335                       |
| round robin.....                  | 481                        |
| round().....                      | <b>1752</b>                |
| roundf.....                       | 1752                       |
| roundl.....                       | 1752                       |
| routing.....                      | 496, 3491                  |
| RPI.....                          | <b>9</b>                   |
| RPP.....                          | <b>9</b>                   |
| RPROTDAT.....                     | 347, 1091                  |
| RPROTDIS.....                     | 347, 1091                  |
| RPROTNORM.....                    | 347, 1091                  |
| RS.....                           | <b>10</b>                  |
| rsh, rationale for omission.....  | 3578                       |
| RS_HIPRI.....                     | 347, 996, 1090, 1679       |
| RTLD_.....                        | 450                        |
| RTLD_DEFAULT.....                 | 709                        |
| RTLD_GLOBAL.....                  | 219, 702, 706-707, 710     |
| RTLD_LAZY.....                    | 219, 706, 709              |
| RTLD_LOCAL.....                   | 219, 707                   |
| RTLD_NEXT.....                    | 709-710                    |
| RTLD_NOW.....                     | 219, 706-707               |
| RTSIG_MAX.....                    | 255, 313, 2012, 3603       |
| runnable process (or thread)..... | 78                         |
| running process (or thread).....  | 78                         |
| runtime values                    |                            |
| increasable.....                  | 257                        |
| invariant.....                    | 253                        |
| rusage.....                       | <b>357</b>                 |
| RUSAGE_.....                      | 450                        |
| RUSAGE_CHILDREN.....              | 357, 1032                  |

|                                    |                                 |                                      |                                                |
|------------------------------------|---------------------------------|--------------------------------------|------------------------------------------------|
| RUSAGE_SELF .....                  | 357, 1032                       | scheduling policy .....              | 79, 100, 3352                                  |
| ru_ .....                          | 450                             | round robin.....                     | 481                                            |
| R_ANCHOR.....                      | 177                             | SCHED_ .....                         | 450                                            |
| R_OK .....                         | 419                             | sched_.....                          | 450                                            |
| s6_ .....                          | 450                             | SCHED_FIFO .....                     | 305, 477, 480, 489, 747, 853                   |
| sact .....                         | <b>3057</b>                     | .....                                | 1016, 1338, 1505, 1507, 1574, 1613, 1635       |
| samefile().....                    | 3518                            | .....                                | 1782, 3442-3444, 3482, 3489, 3589              |
| saved resource limits.....         | 78                              | sched_getparam() .....               | <b>1759</b>                                    |
| saved set-group-ID.....            | 78, 2227                        | sched_getscheduler().....            | <b>1760</b>                                    |
| saved set-user-ID.....             | 78, 2227                        | sched_get_priority_max().....        | <b>1758</b>                                    |
| SA_ .....                          | 450                             | sched_get_priority_min.....          | 1758                                           |
| sa_ .....                          | 450-451                         | SCHED_OTHER.....                     | 305, 480, 483, 1016, 1507                      |
| SA_macros                          |                                 | .....                                | 1574, 3443                                     |
| declared in <signal.h> .....       | 314                             | SCHED_RR .....                       | 305, 477, 480-481, 489, 747                    |
| SA_NOCLDSTOP .....                 | 314, 465, 1870, 1874-1875, 3330 | .....                                | 853, 1016, 1338, 1505, 1507, 1574, 1635        |
| SA_NOCLDWAIT .....                 | 314, 523-524, 1032, 1872, 2130  | .....                                | 1782, 3442-3443, 3482, 3489, 3589              |
| SA_NODEFER .....                   | 314, 1872                       | sched_rr_get_interval().....         | <b>1761</b>                                    |
| SA_ONSTACK .....                   | 314, 747, 1871                  | sched_setparam().....                | <b>1762</b>                                    |
| SA_RESETHAND .....                 | 314, 1871-1872, 3520            | sched_setscheduler() .....           | <b>1764</b>                                    |
| SA_RESTART .....                   | 314, 1489, 1871, 1887, 3520     | SCHED_SPORADIC .....                 | 305, 477, 480, 482, 747                        |
| SA_SIGINFO.....                    | 314, 1870-1871, 1874            | .....                                | 1635, 1782, 3589                               |
| .....                              | 1897, 3411-3412                 | sched_yield() .....                  | <b>1766</b>                                    |
| scalb .....                        | 3522                            | SCM_ .....                           | 451                                            |
| scalbn() .....                     | <b>1754</b>                     | SCM_RIGHTS .....                     | 366                                            |
| scalbnf .....                      | 1754                            | SCN.....                             | 452                                            |
| scalbnl .....                      | 1754                            | scope .....                          | 3313                                           |
| scalbn .....                       | 1754                            | screen.....                          | 79                                             |
| scalbnf .....                      | 1754                            | scroll .....                         | 79                                             |
| scalbnl .....                      | 1754                            | SD.....                              | <b>10</b>                                      |
| scandir .....                      | 565                             | sdb, rationale for omission.....     | 3578                                           |
| scandir() .....                    | <b>1756</b>                     | sdiff, rationale for omission .....  | 3578                                           |
| scanf.....                         | 900                             | search pattern.....                  | 2498                                           |
| scanf() .....                      | <b>1757</b>                     | seconds since the Epoch .....        | 100, 3352                                      |
| sccs .....                         | <b>3060</b>                     | security considerations .....        | 525, 637, 918                                  |
| SCCS commands                      |                                 | .....                                | 1164, 1829, 3324, 3327, 3332, 3344, 3346, 3381 |
| admin .....                        | 2344                            | security, monolithic privileges..... | 3324                                           |
| delta .....                        | 2526                            | sed.....                             | <b>3065</b> , 3592                             |
| get .....                          | 2693                            | addresses .....                      | 3067                                           |
| prs .....                          | 2971                            | editing commands.....                | 3067                                           |
| rmdel .....                        | 3052                            | regular expressions .....            | 3067                                           |
| sact .....                         | 3057                            | seed48.....                          | 712                                            |
| sccs.....                          | 3060                            | seed48().....                        | <b>1767</b>                                    |
| unget.....                         | 3184                            | seekdir().....                       | <b>1768</b>                                    |
| val .....                          | 3212                            | SEEK_ .....                          | 452                                            |
| what.....                          | 3274                            | SEEK_CUR .....                       | 224, 334, 421, 780, 907, 1228                  |
| SCHAR_MAX .....                    | 262-263                         | SEEK_END .....                       | 224, 334, 421, 780, 907, 1228                  |
| SCHAR_MIN .....                    | 262-263                         | SEEK_GET .....                       | 1745                                           |
| schedule alarm .....               | 563                             | SEEK_SET .....                       | 224, 334, 421, 476, 553, 560                   |
| scheduling .....                   | 78                              | .....                                | 780, 907, 1228                                 |
| scheduling allocation domain ..... | 79, 3482                        | SEGV_ .....                          | 450, 452                                       |
| scheduling contention scope .....  | 79, 3482-3483                   | SEGV_ACCERR.....                     | 316                                            |
| scheduling documentation .....     | 490, 3482                       | SEGV_MAPERR .....                    | 316                                            |
|                                    |                                 | select .....                         | 1486                                           |



## Index

|                                            |                                             |
|--------------------------------------------|---------------------------------------------|
| Select Batch Jobs Request .....            | 2338                                        |
| select() .....                             | <b>1770</b>                                 |
| sem .....                                  | 451                                         |
| sem*() .....                               | 3417                                        |
| semaphore .....                            | 79, 101, 3353, 3421, 3589                   |
| semaphore lock operation .....             | 101                                         |
| semaphore unlock operation .....           | 101                                         |
| semctl() .....                             | <b>1791</b> , 3417                          |
| semget() .....                             | <b>1794</b> , 3417                          |
| semid .....                                | 475                                         |
| semop() .....                              | <b>1797</b> , 3417                          |
| SEM_ .....                                 | 450                                         |
| sem_ .....                                 | 450                                         |
| SEM_ .....                                 | 451                                         |
| sem_close() .....                          | <b>1771</b>                                 |
| sem_destroy() .....                        | <b>1773</b>                                 |
| SEM_FAILED .....                           | 309, 1780                                   |
| sem_getvalue() .....                       | <b>1775</b>                                 |
| sem_init() .....                           | <b>1777</b> , 3421                          |
| SEM_NSEMS_MAX .....                        | 255, 1777, 2012, 3603                       |
| sem_open() .....                           | <b>1779</b> , 3421                          |
| sem_perm .....                             | 475                                         |
| sem_post() .....                           | <b>1782</b>                                 |
| sem_timedwait() .....                      | <b>1784</b> , 3452                          |
| sem_trywait() .....                        | <b>1786</b> , 3406, 3423                    |
| SEM_UNDO .....                             | 361, 1797                                   |
| sem_unlink() .....                         | <b>1788</b>                                 |
| SEM_VALUE_MAX .....                        | 255, 1777, 1779, 2012, 3603                 |
| sem_wait .....                             | 1786                                        |
| sem_wait() .....                           | <b>1790</b> , 3406, 3423                    |
| send() .....                               | <b>1802</b>                                 |
| sendmsg() .....                            | <b>1804</b>                                 |
| sendto() .....                             | <b>1807</b>                                 |
| sequential lists .....                     | 2267, 3564                                  |
| Server Shutdown Request .....              | 2338                                        |
| Server Status Request .....                | 2339                                        |
| service name .....                         | 888                                         |
| session .....                              | 79, 526, 1164, 1829, 1841, 3330, 3334, 3380 |
| session leader .....                       | 80                                          |
| session lifetime .....                     | 80                                          |
| session membership .....                   | 2227                                        |
| set .....                                  | <b>2302</b> , 3551                          |
| set cancelability state .....              | 1655                                        |
| set file creation mask .....               | 2096                                        |
| set process group ID for job control ..... | 1829                                        |
| set-group-ID .....                         | 525, 633, 753, 785, 2227, 2488              |
| set-user-ID .....                          | 525, 753, 964, 1164, 2227, 2458, 2488       |
| set-user-ID scripts .....                  | 3087                                        |
| SETALL .....                               | 361, 1791, 1794                             |
| setbuf() .....                             | <b>1810</b>                                 |
| setegid() .....                            | <b>1811</b>                                 |
| setenv() .....                             | <b>1812</b>                                 |
| seteuid() .....                            | <b>1814</b>                                 |
| setgid() .....                             | <b>1815</b> , 3336                          |
| setgrent .....                             | 721                                         |
| setgrent() .....                           | <b>1817</b> , 3344                          |
| sethostent .....                           | 723                                         |
| sethostent() .....                         | <b>1818</b>                                 |
| setitimer .....                            | 990                                         |
| setitimer() .....                          | <b>1819</b>                                 |
| setjmp() .....                             | <b>1820</b> , 3590                          |
| setkey() .....                             | <b>1822</b>                                 |
| setlocale() .....                          | <b>1823</b> , 3590                          |
| setlogmask .....                           | 656                                         |
| setlogmask() .....                         | <b>1827</b> , 3593                          |
| setnetent .....                            | 725                                         |
| setnetent() .....                          | <b>1828</b>                                 |
| setpgid() .....                            | <b>1829</b> , 3329-3331, 3380               |
| setpgrp() .....                            | <b>1831</b>                                 |
| setpriority .....                          | 1016                                        |
| setpriority() .....                        | <b>1832</b> , 3442                          |
| setprotoent .....                          | 727                                         |
| setprotoent() .....                        | <b>1833</b>                                 |
| setpwent .....                             | 729                                         |
| setpwent() .....                           | <b>1834</b> , 3344                          |
| setregid() .....                           | <b>1835</b>                                 |
| setreuid() .....                           | <b>1837</b>                                 |
| setrlimit .....                            | 1029                                        |
| setrlimit() .....                          | <b>1839</b> , 3544                          |
| setservent .....                           | 731                                         |
| setservent() .....                         | <b>1840</b>                                 |
| setsid() .....                             | <b>1841</b> , 3380                          |
| setsockopt() .....                         | <b>1843</b>                                 |
| setstate .....                             | 1083                                        |
| setstate() .....                           | <b>1846</b>                                 |
| setuid() .....                             | <b>1847</b> , 3336                          |
| setutxent .....                            | 733                                         |
| setutxent() .....                          | <b>1850</b>                                 |
| SETVAL .....                               | 361, 1791, 1794                             |
| setvbuf() .....                            | <b>1851</b>                                 |
| sh .....                                   | <b>3074</b> , 3592, 3599                    |
| command history list .....                 | 3078                                        |
| command line editing .....                 | 3078                                        |
| vi line editing command mode .....         | 3079                                        |
| vi line editing insert mode .....          | 3079                                        |
| vi-mode command line editing .....         | 3078                                        |
| shall .....                                | 5, 3316                                     |
| shall, rationale .....                     | 3316                                        |
| shar, rationale for omission .....         | 3578                                        |
| shared memory .....                        | 3433                                        |
| shared memory object .....                 | 80                                          |
| shell .....                                | 80                                          |
| SHELL .....                                | <b>164</b>                                  |
| shell .....                                | 526, 752, 994, 1012, 1164, 1830             |
| .....                                      | 2136, 3329-3332                             |

|                                                    |                            |                                        |                                             |
|----------------------------------------------------|----------------------------|----------------------------------------|---------------------------------------------|
| SHELL.....                                         | 3370                       | word expansions.....                   | 2253                                        |
| shell.....                                         | 3380-3381, 3407, 3413-3414 | shell commands.....                    | 2263, 3560                                  |
| job.....                                           | 1164                       | shell errors.....                      | 3559                                        |
| login.....                                         | 994                        | shell execution environment.....       | 2277, 2350                                  |
| shell command language.....                        | 2245                       | .....                                  | 3042, 3171, 3549, 3568                      |
| alias substitution.....                            | 2248                       | shell grammar.....                     | 2271, 3566                                  |
| appending redirected output.....                   | 2260                       | lexical conventions.....               | 3567                                        |
| arithmetic expansion.....                          | 2257                       | rules.....                             | 3567                                        |
| command substitution.....                          | 2256                       | shell grammar lexical conventions..... | 2271                                        |
| compound commands.....                             | 2268                       | shell grammar rules.....               | 2272                                        |
| consequences of shell errors.....                  | 2262                       | shell introduction.....                | 2245                                        |
| double-quotes.....                                 | 2246                       | shell script.....                      | 80                                          |
| duplicating an input file descriptor.....          | 2261                       | shell scripts                          |                                             |
| duplicating an output file descriptor.....         | 2261                       | exec.....                              | 752                                         |
| escape character (backslash).....                  | 2246                       | shell variables.....                   | 2250, 3550                                  |
| exit status and errors.....                        | 2262                       | shell, job control.....                | 3329, 3407, 3414                            |
| exit status for commands.....                      | 2262                       | shell, login.....                      | 752                                         |
| field splitting.....                               | 2258                       | shell, the.....                        | 80                                          |
| function definition command.....                   | 2270                       | Shell_Path_List.....                   | 2330                                        |
| grammar.....                                       | 2271                       | shift.....                             | 2308                                        |
| here-document.....                                 | 2260                       | shl, rationale for omission.....       | 3578                                        |
| introduction.....                                  | 2245                       | SHM.....                               | 10, 451                                     |
| lists.....                                         | 2266                       | shm.....                               | 451                                         |
| open file descriptors for reading and writing..... | 2262                       | shm*().....                            | 3417                                        |
| parameter expansion.....                           | 2254                       | shmat().....                           | 1860                                        |
| parameters and variables.....                      | 2249                       | shmctl().....                          | 1862, 3417                                  |
| pathname expansion.....                            | 2259                       | shmdt().....                           | 1864, 3417                                  |
| pattern matching notation.....                     | 2278                       | shmget().....                          | 1866                                        |
| patterns matching a single character.....          | 2278                       | shm.....                               | 475                                         |
| patterns matching multiple characters.....         | 2279                       | SHMLBA.....                            | 363, 1860                                   |
| patterns used for filename expansion.....          | 2279                       | shm_.....                              | 450                                         |
| pipelines.....                                     | 2265                       | SHM_.....                              | 451                                         |
| positional parameters.....                         | 2249                       | shm_open().....                        | 1853, 3432-3435                             |
| quote removal.....                                 | 2259                       | shm_perm.....                          | 475                                         |
| quoting.....                                       | 2246                       | SHM_RDONLY.....                        | 363, 1860                                   |
| redirecting input.....                             | 2260                       | SHM_RND.....                           | 363, 1860                                   |
| redirecting output.....                            | 2260                       | shm_unlink().....                      | 1858, 3433-3435                             |
| redirection.....                                   | 2259                       | should.....                            | 6, 3316                                     |
| reserved words.....                                | 2249                       | should, rationale.....                 | 3316                                        |
| shell commands.....                                | 2263                       | SHRT_MAX.....                          | 263                                         |
| shell execution environment.....                   | 2277                       | SHRT_MIN.....                          | 263                                         |
| shell grammar lexical conventions.....             | 2271                       | shutdown().....                        | 1868                                        |
| shell grammar rules.....                           | 2272                       | SHUT_.....                             | 451                                         |
| shell variables.....                               | 2250                       | SHUT_RD.....                           | 367                                         |
| signals and error handling.....                    | 2277                       | SHUT_RDWR.....                         | 368                                         |
| simple commands.....                               | 2263                       | SHUT_WR.....                           | 368                                         |
| single-quotes.....                                 | 2246                       | SIGABRT.....                           | 313, 534, 3342, 3407                        |
| special built-in utilities.....                    | 2280                       | sigaction().....                       | 1870, 3409, 3411                            |
| special parameters.....                            | 2250                       | sigaddset().....                       | 1877                                        |
| tilde expansion.....                               | 2253                       | SIGALRM.....                           | 313, 563, 990, 1916                         |
| token recognition.....                             | 2247                       | sigaltstack().....                     | 1878                                        |
|                                                    |                            | SIGBUS.....                            | 313, 316, 478, 1277, 1280, 1662, 3342, 3407 |
|                                                    |                            | SIGCANCEL.....                         | 1534                                        |

## Index

- SIGCHLD .....313, 316, 523-524, 657, 1032, 1058  
.....1870, 1875, 1884, 2018, 2130, 2139, 3330  
.....3409, 3413
- SIGCLD.....1875, 3413
- SIGCONT.....313, 469, 524, 526, 1163-1164, 2576  
.....3330, 3409, 3413
- sigdelset().....**1880**
- sigemptyset().....**1881**
- SIGEMT.....3407
- SIGEV\_.....450
- sigev\_.....450
- SIGEV\_NONE.....312, 464, 477, 3410
- SIGEV\_SIGNAL.....312, 464, 2059, 3410
- SIGEV\_THREAD.....312, 464-465, 1186, 3410
- sigfillset().....**1883**
- SIGFPE .....313, 316, 1662, 1891, 3342, 3407  
.....3409, 3411
- sighold().....**1884**
- SIGHUP .....313, 523-524, 526, 651, 2554, 2576  
.....3215, 3254, 3414
- sigignore .....1884
- SIGILL.....313, 316, 1662, 1891, 3342, 3407
- siginfo\_t .....**315**
- SIGINT .....313, 855, 2018, 2277, 2528, 2554  
.....2575, 3265, 3331, 3480
- siginterrupt().....**1887**
- SIGIOT .....3407
- sigismember().....**1889**
- SIGKILL.....313, 1164, 1870, 1874-1875, 1884  
.....3407, 3409, 3414
- siglongjmp().....**1890**, 3404, 3415, 3590
- signal .....80, 3336
- signal acceptance.....3408
- signal actions.....3413
- Signal Batch Job Request.....2339
- signal concepts.....3406
- signal delivery.....3408
- signal generation.....3408
- signal generation and delivery.....463  
    realtime.....464
- signal handler.....1891
- signal names.....3407
- signal processes.....2746
- signal stack.....80
- signal().....**1891**, 3406, 3409
- signaling a condition.....1542
- signaling NaN.....232
- signals.....463, 3492, 3568
- signals and error handling.....2277
- sigbit().....**1893**
- sigpause.....1884
- sigpause().....**1894**
- sigpending().....**1895**
- SIGPIPE.....313, 778, 817, 877, 882, 908, 911  
.....1680, 2215, 3342, 3405
- SIGPOLL.....313, 316, 651, 1089-1090
- sigprocmask .....1662
- sigprocmask().....**1896**, 3408
- SIGPROF.....313, 990
- sigqueue().....**1897**
- SIGQUEUE\_MAX .....255, 1897, 2012
- SIGQUIT .....313, 2018, 2277, 2554
- sigrelse.....1884
- sigrelse().....**1899**
- SIGRTMAX.....312, 463, 465, 1873, 1897, 1904  
.....1908, 3411-3412
- SIGRTMIN.....312, 463, 465, 1873, 1897, 1904  
.....1908, 3411-3412
- SIGSEGV .....313, 316, 478, 1030, 1322, 1500  
.....1662, 1891, 3342, 3407, 3411
- sigset.....1884, 1899
- sigsetjmp().....**1900**, 3590
- sigset\_t.....3407
- SIGSTKSZ .....314, 1878
- SIGSTOP .....313, 463, 1870, 1875, 1884, 3413
- sigsuspend().....**1902**, 3413, 3416
- SIGSYS .....313, 3407
- SIGTERM.....313, 2576, 3407
- sigtimedwait().....**1904**, 3406, 3425, 3452
- SIGTRAP .....313, 316, 3407
- SIGTSTP.....313, 463, 3331, 3413-3414
- SIGTTIN.....313, 463, 820, 826, 1699, 3331  
.....3381, 3413-3414
- SIGTTOU .....313, 463, 777, 816, 877, 881, 908  
.....910, 2028, 2030, 2032, 2039, 2042, 2214  
.....3330, 3381, 3413-3414
- SIGURG .....313, 1090
- SIGUSR1 .....313, 3407
- SIGUSR2 .....313, 3407
- SIGVTALRM .....313, 990
- sigwait().....**1908**, 3406, 3486
- sigwaitinfo.....1904
- sigwaitinfo().....**1910**, 3406, 3425
- sigwait\_multiple().....3408
- SIGXCPU .....313, 1029
- SIGXFSZ.....313, 1029, 2085
- SIG\_.....452
- SIG\_ATOMIC\_MAX .....331
- SIG\_ATOMIC\_MIN.....331
- SIG\_BLOCK.....314, 1662
- SIG\_DFL.....312, 465, 746, 1030, 1870, 1872  
.....1891, 3409, 3413
- SIG\_ERR.....312, 1892
- SIG\_HOLD .....312, 1884
- SIG\_IGN .....312, 466, 523-524, 746, 753, 1032

|                                                     |                                   |                      |
|-----------------------------------------------------|-----------------------------------|----------------------|
| .....1870, 1891, 2130, 2277, 3330, 3409, 3413, 3415 | sockets .....                     | 496, 3491            |
| SIG_SETMASK .....                                   | address families .....            | 496                  |
| SIG_UNBLOCK .....                                   | addressing .....                  | 496                  |
| simple commands .....                               | asynchronous errors .....         | 500                  |
| sin() .....                                         | connection indication queue ..... | 499                  |
| sin6 .....                                          | Internet Protocols .....          | 503, 3492            |
| sinf .....                                          | IPv4 .....                        | 504, 3492            |
| single-quote .....                                  | IPv6 .....                        | 504, 3492            |
| single-quotes .....                                 | local UNIX connection .....       | 3492                 |
| sinh() .....                                        | local UNIX connections .....      | 503                  |
| sinhf .....                                         | options .....                     | 500                  |
| sinhl .....                                         | pending error .....               | 498                  |
| sinl .....                                          | protocols .....                   | 496                  |
| sinl() .....                                        | signals .....                     | 499                  |
| sin .....                                           | SOCK .....                        | 452                  |
| SIO .....                                           | SOCK_DGRAM .....                  | 366, 503, 1921, 1923 |
| SIOCATMARK .....                                    | SOCK_RAW .....                    | 366, 503             |
| sival .....                                         | SOCK_SEQPACKET .....              | 366, 503, 1921, 1923 |
| size, rationale for omission .....                  | SOCK_STREAM .....                 | 366, 503, 1921, 1923 |
| SIZE_MAX .....                                      | soft limit .....                  | 81                   |
| size_t .....                                        | software development .....        | 3586, 3593           |
| SI .....                                            | SOL_SOCKET .....                  | 366                  |
| si .....                                            | SOMAXCONN .....                   | 367                  |
| SI .....                                            | sort .....                        | <b>3093</b> , 3592   |
| SI_ASYNCIO .....                                    | source code .....                 | 81                   |
| SI_MSGQ .....                                       | SO_ACCEPTCONN .....               | 366                  |
| SI_QUEUE .....                                      | SO_BROADCAST .....                | 366                  |
| SI_TIMER .....                                      | SO_DEBUG .....                    | 367                  |
| SI_USER .....                                       | SO_DONTROUTE .....                | 367                  |
| slash .....                                         | SO_ERROR .....                    | 367                  |
| sleep .....                                         | SO_KEEPALIVE .....                | 367                  |
| sleep() .....                                       | SO_LINGER .....                   | 367                  |
| SLR(1) grammars .....                               | SO_OOBINLINE .....                | 367                  |
| sl .....                                            | SO_RCVBUF .....                   | 367                  |
| SND .....                                           | SO_RCVLOWAT .....                 | 367                  |
| SNDZERO .....                                       | SO_RCVTIMEO .....                 | 367                  |
| snprintf .....                                      | SO_REUSEADDR .....                | 367                  |
| snprintf() .....                                    | SO_SNDBUF .....                   | 367                  |
| SO .....                                            | SO_SNDLOWAT .....                 | 367                  |
| socketmark() .....                                  | SO_SNDTIMEO .....                 | 367                  |
| socket .....                                        | SO_TYPE .....                     | 367                  |
| socket address .....                                | space character .....             | 82                   |
| socket I/O mode .....                               | spawn .....                       | 82                   |
| socket out-of-band data .....                       | spawn example .....               | 3522                 |
| socket out-of-band data state .....                 | special built-in .....            | 82, 2479, 2887, 2900 |
| socket owner .....                                  | .....2984, 3087, 3142, 3159, 3565 |                      |
| socket queue limit .....                            | special built-in utilities .....  | 2280, 3569           |
| socket queue limits .....                           | break .....                       | 2281, 2285           |
| socket receive queue .....                          | characteristics .....             | 2280                 |
| socket types .....                                  | colon .....                       | 2283                 |
| socket() .....                                      | dot .....                         | 2287                 |
| socketpair() .....                                  | eval .....                        | 2289                 |
|                                                     | exec .....                        | 2291                 |

## Index

|                                         |                              |                                          |                                   |
|-----------------------------------------|------------------------------|------------------------------------------|-----------------------------------|
| exit .....                              | 2293                         | SS_REPL_MAX .....                        | 255, 3447                         |
| export .....                            | 2295                         | stack size .....                         | 1495                              |
| readonly .....                          | 2298                         | stack_t .....                            | <b>315</b>                        |
| return .....                            | 2300                         | standard error .....                     | 82, 2259                          |
| set .....                               | 2302                         | standard I/O streams .....               | 3416                              |
| shift .....                             | 2308                         | standard input .....                     | 82, 2259                          |
| times .....                             | 2310                         | standard output .....                    | 82, 2259                          |
| trap .....                              | 2312                         | standard utilities .....                 | 83                                |
| unset .....                             | 2316                         | START .....                              | 2030                              |
| special characters .....                | 3383                         | stat .....                               | 915, 3543, 3587-3588              |
| special control character .....         | 3384                         | stat data structure .....                | <b>370</b>                        |
| special parameter .....                 | 82                           | stat() .....                             | <b>1932</b> , 3326, 3432, 3543    |
| special parameters .....                | 2250, 3549                   | state-dependent character encoding ..... | 3355                              |
| special targets .....                   | 2835                         | statvfs .....                            | 920                               |
| specific implementation .....           | 3328                         | statvfs() .....                          | <b>1933</b> , 3544                |
| SPEC_CHAR .....                         | 178                          | stderr .....                             | 335, 1934                         |
| spell, rationale for omission .....     | 3579                         | STDERR_FILENO .....                      | 424, 1934                         |
| spin lock .....                         | 82                           | stdin .....                              | 335, 1934                         |
| spin locks .....                        | 3468-3469                    | STDIN_FILENO .....                       | 424, 1371, 1934                   |
| split .....                             | <b>3099</b> , 3592           | stdio locking functions .....            | 831                               |
| split files                             |                              | stdio with explicit client locking ..... | 959                               |
| csplit .....                            | 2494                         | stdout .....                             | 335, 1934                         |
| split .....                             | 3099                         | STDOUT_FILENO .....                      | 424, 1371, 1934                   |
| SPN .....                               | <b>10</b>                    | STOP .....                               | 2030                              |
| spoofing .....                          | 2299                         | stpcpy .....                             | 1946                              |
| sporadic server .....                   | 82                           | stpcpy() .....                           | <b>1936</b>                       |
| sporadic server policy                  |                              | stpncpy .....                            | 1970                              |
| execution capacity .....                | 482                          | stpncpy() .....                          | <b>1937</b>                       |
| replenishment period .....              | 482                          | STR .....                                | 452                               |
| sporadic server scheduling policy ..... | 3444                         | strbuf .....                             | <b>345</b>                        |
| sprintf .....                           | 864                          | strcasecmp() .....                       | <b>1938</b>                       |
| sprintf() .....                         | <b>1925</b>                  | strcasecmp_l .....                       | 1938                              |
| spurious wakeup .....                   | 1542                         | strcat() .....                           | <b>1940</b>                       |
| sqrt() .....                            | <b>1926</b>                  | strchr() .....                           | <b>1941</b>                       |
| sqrtf .....                             | 1926                         | strcmp() .....                           | <b>1942</b>                       |
| sqrtl .....                             | 1926                         | strcoll() .....                          | <b>1944</b>                       |
| srand .....                             | 1693                         | strcoll_l .....                          | 1944                              |
| srand() .....                           | <b>1928</b>                  | strcpy() .....                           | <b>1946</b>                       |
| srand48 .....                           | 712                          | strcsn() .....                           | <b>1949</b>                       |
| srand48() .....                         | <b>1929</b>                  | strdup() .....                           | <b>1950</b>                       |
| random .....                            | 1083                         | STREAM .....                             | 83                                |
| random() .....                          | <b>1930</b>                  | stream .....                             | 83                                |
| SS .....                                | <b>10</b>                    | STREAM .....                             | 221, 1092, 1094, 1679, 1698, 2213 |
| sscanf .....                            | 900                          | stream                                   |                                   |
| sscanf() .....                          | <b>1931</b>                  | byte-oriented .....                      | 472                               |
| SSIZE_MAX .....                         | 263, 381, 1295, 1311, 1697   | wide-oriented .....                      | 472                               |
| .....                                   | 1708, 1957, 2212, 3519, 3603 | STREAM end .....                         | 83                                |
| ssize_t .....                           | 380                          | STREAM head .....                        | 83                                |
| SS_ .....                               | 450                          | STREAM head/tail .....                   | <b>473</b>                        |
| ss_ .....                               | 450-451                      | stream-full-policy attribute .....       | 510-511, 513, 1428                |
| SS_DISABLE .....                        | 314, 1878-1879               | stream-min-size attribute .....          | 513, 1431                         |
| SS_ONSTACK .....                        | 314, 1878                    | STREAMS .....                            | 25, 345, 457                      |
|                                         |                              | streams .....                            | 469                               |

|                                         |                                |                                             |                                          |
|-----------------------------------------|--------------------------------|---------------------------------------------|------------------------------------------|
| STREAMS .....                           | 651, 767, 788, 996, 1088, 1104 | strtoll() .....                             | <b>1995</b>                              |
| .....                                   | 1346, 1367, 1486, 3416         | strtoul() .....                             | <b>1996</b>                              |
| access .....                            | 474                            | strtoull .....                              | 1996                                     |
| streams                                 |                                | strtoumax .....                             | 1988                                     |
| interaction with file descriptors ..... | 470                            | strtoxmax() .....                           | <b>1999</b>                              |
| STREAMS                                 |                                | structures, additions to .....              | 3399                                     |
| multiplexed .....                       | 1095                           | strxfrm() .....                             | <b>2000</b>                              |
| overview .....                          | 473                            | strxfrm_l .....                             | 2000                                     |
| streams                                 |                                | str_ .....                                  | 450                                      |
| stream orientation .....                | 472                            | str_list .....                              | <b>345</b>                               |
| STREAMS multiplexor .....               | 83                             | str_mlist .....                             | <b>346</b>                               |
| STREAM_MAX .....                        | 255, 792, 850, 1371, 2012      | stty .....                                  | <b>3107, 3370, 3593</b>                  |
| .....                                   | 2069, 3603                     | combination modes .....                     | 3112                                     |
| strerror() .....                        | <b>1952</b>                    | control modes .....                         | 3107                                     |
| strerror_l .....                        | 1952                           | input modes .....                           | 3108                                     |
| strerror_r .....                        | 1952                           | local modes .....                           | 3110                                     |
| strfdinsert .....                       | <b>345</b>                     | output modes .....                          | 3109                                     |
| strfmon() .....                         | <b>1955</b>                    | special control character assignments ..... | 3111                                     |
| strfmon_l .....                         | 1955                           | ST_ .....                                   | 451                                      |
| strftime() .....                        | <b>1959</b>                    | st_ .....                                   | 451                                      |
| strftime_l .....                        | 1959                           | st_gid .....                                | 2357                                     |
| string .....                            | 83                             | st_mode .....                               | 2357                                     |
| strings .....                           | <b>3102, 3593</b>              | st_mtime .....                              | 2357                                     |
| strioctl .....                          | <b>345</b>                     | ST_NOSUID .....                             | 375, 747, 920                            |
| strip .....                             | <b>3105, 3593</b>              | ST_RDONLY .....                             | 375, 920                                 |
| strlen() .....                          | <b>1965</b>                    | st_size .....                               | 2357                                     |
| strncasecmp .....                       | 1938                           | st_uid .....                                | 2357                                     |
| strncasecmp() .....                     | <b>1967</b>                    | su, rationale for omission .....            | 3579                                     |
| strncasecmp_l .....                     | 1938, 1967                     | subprofiling .....                          | 20, 3321                                 |
| strncat() .....                         | <b>1968</b>                    | subprofiling option groups .....            | 3607                                     |
| strncmp() .....                         | <b>1969</b>                    | subshell .....                              | 84                                       |
| strncpy() .....                         | <b>1970</b>                    | subshells .....                             | 3331                                     |
| strndup .....                           | 1950                           | successfully completed .....                | 3343                                     |
| strndup() .....                         | <b>1972</b>                    | successfully transferred .....              | 84                                       |
| strnlen() .....                         | <b>1973</b>                    | sum .....                                   | 3543                                     |
| strpbrk() .....                         | <b>1974</b>                    | sum, rationale for omission .....           | 3579                                     |
| strpeek .....                           | <b>345</b>                     | sun_ .....                                  | 451                                      |
| strptime() .....                        | <b>1975</b>                    | superuser .....                             | 540, 637, 1182, 2107, 2655               |
| strrchr() .....                         | <b>1980</b>                    | .....                                       | 2794, 2949, 3324, 3336, 3346, 3551, 3576 |
| strrecvfd .....                         | <b>345</b>                     | supplementary group ID .....                | 84, 3336                                 |
| strsignal() .....                       | <b>1981</b>                    | supplementary group IDs .....               | 2227                                     |
| strspn() .....                          | <b>1982</b>                    | supplementary groups .....                  | 637, 986, 3346                           |
| strstr() .....                          | <b>1983</b>                    | Supported Threads functions .....           | 3471                                     |
| strtod() .....                          | <b>1984</b>                    | suseconds_t .....                           | 380                                      |
| strtof .....                            | 1984                           | suspended job .....                         | 84                                       |
| strtoimax() .....                       | <b>1988</b>                    | SVID .....                                  | 1900                                     |
| strtok() .....                          | <b>1989</b>                    | SVR4 .....                                  | 1279, 1325                               |
| strtok_r .....                          | 1989                           | sv_ .....                                   | 450                                      |
| strtol() .....                          | <b>1992</b>                    | SV_ .....                                   | 452                                      |
| strtold .....                           | 1984                           | swab() .....                                | <b>2002</b>                              |
| strtold() .....                         | <b>1994</b>                    | swapcontext .....                           | 3521                                     |
| strtoll .....                           | 1992                           | swprintf .....                              | 940                                      |
|                                         |                                | swprintf() .....                            | <b>2003</b>                              |

## Index

|                                                 |                                           |                                               |
|-------------------------------------------------|-------------------------------------------|-----------------------------------------------|
| swscanf.....                                    | 949                                       | .....2034, 2098, 3326, 3332, 3407, 3409, 3413 |
| swscanf() .....                                 | <b>2004</b>                               | system()..... <b>2018</b> , 3590, 3593-3594   |
| SWTCH.....                                      | 452                                       | system-wide .....                             |
| symbolic constant.....                          | 84, 3318, 3337                            | S.....                                        |
| symbolic link.....                              | 84, 3337                                  | s.....                                        |
| symbolic name.....                              | 3318                                      | S.....                                        |
| symbols.....                                    | 3398                                      | S_ constants                                  |
| POSIX.1 .....                                   | 448                                       | defined in <sys/stat.h> .....                 |
| symlink() .....                                 | <b>2005</b>                               | S_ macros                                     |
| symlinkat.....                                  | 2005                                      | defined in <sys/stat.h> .....                 |
| symlinkat().....                                | <b>2008</b>                               | S_BANDURG.....                                |
| SYMLINK_MAX.....                                | 257, 265, 858, 2005                       | S_ERROR.....                                  |
| SYMLOOP_MAX.....                                | 255, 593, 666, 768, 788, 850              | S_HANGUP.....                                 |
| .....                                           | 895, 921, 927, 933, 937, 1170, 1265, 1335 | S_HIPRI.....                                  |
| .....                                           | 1715, 1806, 1809, 2012, 2086, 3404        | S_IFBLK.....                                  |
| SYMTYPE.....                                    | 391                                       | S_IFCHR.....                                  |
| sync() .....                                    | <b>2009</b>                               | S_IFDIR.....                                  |
| synchronized I/O.....                           | 3426, 3589                                | S_IFIFO.....                                  |
| data integrity completion.....                  | 3343, 3426                                | S_IFLNK.....                                  |
| file integrity completion.....                  | 3343, 3426                                | S_IFMT.....                                   |
| synchronized I/O completion.....                | 85                                        | S_IFREG.....                                  |
| synchronized I/O data integrity completion..... | 85                                        | S_IFSOCK.....                                 |
| synchronized I/O file integrity completion..... | 85                                        | S_INPUT.....                                  |
| synchronized I/O operation.....                 | 85                                        | S_IRGRP.....                                  |
| synchronized input and output.....              | 85                                        | S_IROTH.....                                  |
| synchronous I/O operation.....                  | 85                                        | S_IRUSR.....                                  |
| synchronously accept a signal.....              | 1905                                      | S_IRWXG.....                                  |
| synchronously-generated signal.....             | 85, 3342                                  | S_IRWXO.....                                  |
| sysconf() .....                                 | <b>2010</b> , 3328, 3429, 3431, 3433      | S_IRWXU.....                                  |
| .....                                           | 3480, 3537, 3587-3588                     | S_ISBLK.....                                  |
| syslog.....                                     | 656                                       | S_ISCHR.....                                  |
| syslog() .....                                  | <b>2017</b> , 3593                        | S_ISDIR.....                                  |
| system.....                                     | 86                                        | S_ISFIFO.....                                 |
| system boot.....                                | 86                                        | S_ISGID.....                                  |
| system call.....                                | 3342                                      | S_ISLNK.....                                  |
| system clock.....                               | 86                                        | S_ISREG.....                                  |
| system configuration values.....                | 2701                                      | S_ISSOCK.....                                 |
| system console.....                             | 86, 3343                                  | S_ISUID.....                                  |
| system crash.....                               | 86, 923                                   | S_ISVTX.....                                  |
| System database.....                            | 3343                                      | S_IWGRP.....                                  |
| system databases.....                           | 86                                        | S_IWOTH.....                                  |
| system documentation.....                       | 86, 3317                                  | S_IWUSR.....                                  |
| system environment.....                         | 3586, 3593                                | S_IXGRP.....                                  |
| System III.....                                 | 637, 2098, 3334, 3518                     | S_IXOTH.....                                  |
| system interfaces.....                          | 521, 3520                                 | S_IXUSR.....                                  |
| system name.....                                | 2098, 3175                                | S_MSG.....                                    |
| system process.....                             | 86, 3343                                  | S_OUTPUT.....                                 |
| system reboot.....                              | 87, 3343                                  | S_RDBAND.....                                 |
| system trace event.....                         | 87                                        | S_RDNORM.....                                 |
| system trace event type definitions.....        | 513                                       | S_TYPEISMQ.....                               |
| System V.....                                   | 526, 563, 637, 754, 784, 860              | S_TYPEISSEM.....                              |
| .....                                           | 1012, 1164, 1254, 1750, 1841, 1875, 1900  | S_TYPEISSHM.....                              |
|                                                 |                                           | S_TYPEISTMO.....                              |

|                                    |                          |                                           |                           |
|------------------------------------|--------------------------|-------------------------------------------|---------------------------|
| S_WRBAND .....                     | 347, 1089                | terminal access control.....              | 2034, 2042, 3381          |
| S_WRNORM .....                     | 347, 1089                | terminal characteristics                  |                           |
| tab character .....                | 87                       | stty .....                                | 3107                      |
| TABDLY .....                       | 394                      | tabs .....                                | 3116                      |
| TABn .....                         | 395                      | tput .....                                | 3149                      |
| tabs .....                         | <b>3116</b> , 3592       | tty .....                                 | 3163                      |
| TABSIZE .....                      | 595, 1226                | terminal device file.....                 | 3380                      |
| tag file creation.....             | 2498                     | closing .....                             | 3383                      |
| tail .....                         | <b>3120</b> , 3592       | terminal device name.....                 | 2089                      |
| talk .....                         | <b>3124</b> , 3592       | terminal type .....                       | 3378                      |
| tan() .....                        | <b>2023</b>              | terminal types .....                      | 184                       |
| tanf .....                         | 2023                     | terminate a process.....                  | 525                       |
| tanh() .....                       | <b>2025</b>              | terminate processes .....                 | 2746                      |
| tanhf .....                        | 2025                     | terminology .....                         | 3316                      |
| tanhL .....                        | 2025                     | termios .....                             | 185                       |
| tanl .....                         | 2023                     | canonical mode input processing.....      | 187                       |
| tanl() .....                       | <b>2027</b>              | control modes.....                        | 194                       |
| tar format .....                   | 2942                     | controlling terminal .....                | 186                       |
| tar, rationale for omission.....   | 3579                     | input modes .....                         | 191                       |
| target queue.....                  | 2336                     | local modes.....                          | 195                       |
| target rule.....                   | 2830                     | non-canonical mode input processing ..... | 188                       |
| tcdrain() .....                    | <b>2028</b>              | output modes .....                        | 192                       |
| tcflow() .....                     | <b>2030</b>              | process group .....                       | 185                       |
| tcflush() .....                    | <b>2032</b>              | special control characters .....          | 196                       |
| tcgetattr() .....                  | <b>2034</b> , 3330       | termios structure.....                    | 2034, 3383                |
| tcgetpgrp() .....                  | <b>2036</b> , 3331, 3380 | test .....                                | <b>3131</b> , 3591-3592   |
| tcgetsid() .....                   | <b>2038</b>              | TeX .....                                 | 3592                      |
| TCIFLUSH .....                     | 397, 2032                | text column.....                          | 87                        |
| TCIOFF .....                       | 397, 2030                | text file.....                            | 88, 3343                  |
| TCIOFLUSH .....                    | 397, 2032                | tfind .....                               | 2046                      |
| TCION .....                        | 397, 2030                | tfind() .....                             | <b>2053</b>               |
| TCOFLUSH .....                     | 2032                     | tgamma() .....                            | <b>2054</b>               |
| TCOOFF .....                       | 397, 2030                | tgammaf .....                             | 2054                      |
| TCOON .....                        | 397, 2030                | tgammal .....                             | 2054                      |
| TCP_ .....                         | 450                      | TGEXEC .....                              | 391                       |
| TCP_NODELAY .....                  | 291                      | TGREAD .....                              | 391                       |
| TCSADRAIN .....                    | 396, 2041                | TGWRITE .....                             | 391                       |
| TCSAFLUSH .....                    | 396, 2041                | THOUSEP .....                             | 250                       |
| TCSANOW .....                      | 396, 2041                | thread .....                              | 88, 3344                  |
| tcsendbreak() .....                | <b>2039</b>              | thread cancelability states .....         | 3487                      |
| tcsetattr() .....                  | <b>2041</b> , 3330, 3379 | thread cancelability type .....           | 3487                      |
| tcsetpgrp() .....                  | <b>2044</b> , 3330-3331  | thread cancellation .....                 | 3485, 3487                |
| TCT .....                          | <b>10</b>                | cleanup handlers .....                    | 494                       |
| tdelete() .....                    | <b>2046</b>              | thread cancellation points .....          | 3487                      |
| tee .....                          | <b>3128</b> , 3591       | thread concurrency level .....            | 3475                      |
| TEF .....                          | <b>11</b>                | thread creation .....                     | 1564                      |
| telldir() .....                    | <b>2050</b>              | thread creation attributes .....          | 1495, 3464                |
| tempnam() .....                    | <b>2051</b>              | thread ID .....                           | 88, 486, 1568, 3344, 3480 |
| TERM .....                         | <b>164</b> , 3370        | thread interactions.....                  | 3490                      |
| terminal                           |                          | thread list .....                         | 88                        |
| controlling .....                  | 186                      | thread mutex .....                        | 3480                      |
| terminal (or terminal device)..... | 87                       | thread mutexes .....                      | 487                       |
|                                    |                          | thread read-write lock .....              | 3489                      |



## Index

- thread scheduling.....488, 3480
- thread stack guard size.....3476
- thread termination.....1569
- thread-safe.....88, 3344
- thread-safe function.....3344
- thread-safety.....101, 486, 831, 3353, 3477
- thread-safety, rationale.....3353
- thread-specific data.....3466
- thread-specific data key.....88
- thread-specific data key creation.....1583
- thread-specific data key deletion.....1585
- thread-specific data management.....1578
- threads.....485, 3463
  - implementation models.....3465
  - regular file operations.....495
- threads extensions.....3472
- tilde.....89
- tilde expansion.....2253, 3552
- time.....3140, 3591, 3593
- time().....2056, 3404
- timeouts.....89, 3456
- timer.....89
- timer ID.....2061
- timer overrun.....89
- timers.....3448
- TIMER\_.....451
- timer\_.....451
- TIMER\_ABSTIME...404, 484, 646, 2063, 3448-3450
- timer\_create().....2059
- timer\_delete().....2062
- timer\_getoverrun().....2063
- timer\_gettime.....2063
- TIMER\_MAX.....255, 2012, 3603
- timer\_settime.....2063
- timer\_settime().....3448-3450
- timer\_t.....380
- times.....2310
- times().....2066, 3404, 3455, 3587
- timestamp clock.....3516
- timeval.....359, 377
- timezone().....2068
- time\_t.....380, 3352
- TMAGIC.....391
- TMAGLEN.....391
- TMPDIR.....164, 2929
- tmpfile().....2069
- tmpnam().....2071
- TMP\_MAX.....334, 2051, 2070-2071
- tms\_.....451
- tm\_.....451
- toascii().....2073
- TOEXEC.....391
- token.....89
- token recognition.....2247, 3548
- tolower().....2074
- tolower\_l.....2074
- TOREAD.....391
- TOSTOP.....396, 777, 816, 877, 881, 908
  - .....910, 2214, 3330
- touch.....3144, 3543, 3592
- toupper().....2076
- toupper\_l.....2076
- towctrans().....2078
- towctrans\_l.....2078
- tolower().....2080
- tolower\_l.....2080
- TOWRITE.....391
- towupper().....2082
- towupper\_l.....2082
- TPI.....11
- TPP.....11
- TPS.....11
- tput.....3149, 3593
- tr.....3152, 3592
- trace analyzer.....3505
- trace analyzer process.....89
- trace controller process.....89
- trace event.....89
- trace event type.....89
- trace event type mapping.....90
- trace event type-filtering.....3514
- trace event types.....3514
- trace event, POSIX\_TRACE\_ERROR.....514
- trace event, POSIX\_TRACE\_FILTER.....514, 1462
- trace event, POSIX\_TRACE\_OVERFLOW.....514
- trace event, POSIX\_TRACE\_RESUME.....514
- trace event, POSIX\_TRACE\_START.....514, 1471
- trace event, POSIX\_TRACE\_STOP.....514, 1471
- trace examples.....3503
- trace filter.....90
- trace functions.....517
- trace generation version.....90
- trace log.....90
- trace model.....3498
- trace operation control.....3503
- trace point.....90
- trace storage.....3502
- trace stream.....90
- trace stream attribute.....3507
- trace stream identifier.....90
- trace stream states.....3501
- trace system.....90
- trace-name attribute.....513, 1425
- traced process.....90
- TRACE\_EVENT\_NAME\_MAX.....256, 1450, 1452

|                                        |                                         |                                    |                              |
|----------------------------------------|-----------------------------------------|------------------------------------|------------------------------|
| TRACE_NAME_MAX .....                   | 256                                     | TYM.....                           | 12                           |
| TRACE_SYS_MAX.....                     | 256, 1447                               | type.....                          | 3165                         |
| TRACE_USER_EVENT_MAX.....              | 256, 1450, 1452                         | typed memory.....                  | 3435                         |
| tracing .....                          | 24, 101                                 | typed memory name space .....      | 91                           |
| TRACING.....                           | 1423, 1425, 1427, 1430, 1433-1442       | typed memory object .....          | 91                           |
| .....                                  | 1444, 1446, 1450, 1452, 1454-1455, 1457 | typed memory pool.....             | 91                           |
| .....                                  | 1459-1460, 1462, 1464-1465, 1468-1471   | typed memory port .....            | 91                           |
| .....                                  | 1473-1475                               | TZ.....                            | 164, 3370                    |
| tracing .....                          | 3353, 3492                              | tzname.....                        | 2092                         |
| tracing all processes .....            | 3501                                    | TZNAME_MAX .....                   | 256, 2012, 3603              |
| tracing status of a trace stream ..... | 90                                      | tzset.....                         | 2092                         |
| tracing, detailed objectives.....      | 3494                                    | tzset() .....                      | 2092, 3590                   |
| Track Batch Job Request .....          | 2339                                    | T_FMT .....                        | 250                          |
| trap.....                              | 2312                                    | T_FMT_AMPM.....                    | 250                          |
| TRAP.....                              | 450, 452                                | t_scalar_t.....                    | 345                          |
| TRAP_BRKPT .....                       | 316                                     | t_uscalar_t.....                   | 345, 1091                    |
| TRAP_TRACE.....                        | 316                                     | ualarm .....                       | 3522, 3587                   |
| TRC .....                              | 11                                      | UCHAR_MAX.....                     | 262-263                      |
| TRI .....                              | 11                                      | ucontext_t .....                   | 315                          |
| triggering .....                       | 3516                                    | uc.....                            | 450                          |
| TRL .....                              | 11                                      | UID_MAX.....                       | 3519                         |
| troff.....                             | 3592                                    | uid_t .....                        | 380, 3344                    |
| trojan horse.....                      | 2794, 3324                              | UINT.....                          | 452                          |
| true.....                              | 2244, 2264, 3158, 3545, 3591            | UINTMAX_MAX .....                  | 331                          |
| trunc() .....                          | 2084                                    | UINTN_MAX.....                     | 330                          |
| truncate().....                        | 2085                                    | UINTPTR_MAX .....                  | 331                          |
| truncation-status attribute.....       | 1450                                    | UINT_FASTN_MAX .....               | 330                          |
| truncf .....                           | 2084                                    | UINT_LEASTN_MAX .....              | 330                          |
| truncf().....                          | 2087                                    | UINT_MAX .....                     | 263, 563, 1917               |
| truncf_l .....                         | 2084, 2087                              | UIO_MAXIOV .....                   | 451                          |
| TSA .....                              | 11                                      | ulimit .....                       | 3167                         |
| tsearch .....                          | 2046                                    | ulimit().....                      | 2094                         |
| tsearch().....                         | 2088                                    | ULLONG_MAX.....                    | 263, 1997                    |
| TSGID.....                             | 391                                     | ULONG_MAX .....                    | 263, 1997, 2188, 3537        |
| TSH .....                              | 12                                      | UL .....                           | 451                          |
| tsort.....                             | 3160                                    | UL_GETFSIZE.....                   | 411, 2094                    |
| TSP.....                               | 12                                      | UL_SETFSIZE.....                   | 411, 2094                    |
| TSS.....                               | 12                                      | umask .....                        | 2244, 2264, 3169, 3545, 3593 |
| TSUID.....                             | 391                                     | umask().....                       | 2096, 3587                   |
| TSVTX .....                            | 391                                     | umount() .....                     | 3333                         |
| tty .....                              | 3163, 3593                              | unalias .....                      | 2244, 2264, 3173, 3545, 3591 |
| ttyname().....                         | 2089, 3477                              | uname.....                         | 3175, 3593                   |
| ttyname_r.....                         | 2089                                    | uname() .....                      | 2098, 3587                   |
| TTY_NAME_MAX .....                     | 256, 2012, 2089, 3603                   | unary primaries .....              | 3133                         |
| TUEXEC.....                            | 391                                     | unbind .....                       | 91                           |
| TUREAD.....                            | 391                                     | unbounded priority inversion ..... | 3484                         |
| TUWRITE .....                          | 391                                     | uncompress .....                   | 3178                         |
| TVERSION .....                         | 391                                     | undefined.....                     | 6, 3317                      |
| TVERSLEN .....                         | 391                                     | undefined, rationale .....         | 3317                         |
| tv_.....                               | 451                                     | underlying function .....          | 471                          |
| twalk.....                             | 2046                                    | unexpand .....                     | 3181, 3592                   |
| twalk() .....                          | 2091                                    | unset.....                         | 3184                         |
|                                        |                                         | unsetc().....                      | 2100                         |

## Index

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| ungetwc()                         | 2102                              |
| unicast                           | 504                               |
| uniq                              | 3187, 3592                        |
| unlink                            | 3191, 3588                        |
| unlink()                          | 2104, 3338, 3405, 3432-3433, 3435 |
| unlinkat                          | 2104                              |
| unlinkat()                        | 2109                              |
| unlockpt()                        | 2110                              |
| unpack, rationale for omission    | 3579                              |
| unsafe functions                  | 3414                              |
| unset                             | 2316                              |
| unsetenv()                        | 2111                              |
| unspecified                       | 6, 3317                           |
| unspecified, rationale            | 3317                              |
| until loop                        | 2270, 3565                        |
| UP                                | 12                                |
| upper multiplexing                | 83                                |
| upshifting                        | 91                                |
| US-ASCII                          | 1103                              |
| uselocale()                       | 2112                              |
| user database                     | 92, 3344                          |
| user database access              | 3345                              |
| user ID                           | 92                                |
| real and effective                | 1837                              |
| setting real and effective        | 1837                              |
| user identity                     |                                   |
| id                                | 2725                              |
| logname                           | 2781                              |
| newgrp                            | 2881                              |
| who                               | 3276                              |
| user name                         | 92                                |
| user requirements                 | 3583                              |
| user trace event                  | 92                                |
| user trace event type definitions | 516                               |
| User_List                         | 2331                              |
| USER_PROCESS                      | 433, 733-734                      |
| USHRT_MAX                         | 263                               |
| usleep                            | 3522, 3587                        |
| ustar format                      | 2942                              |
| UTC                               | 2092                              |
| utility                           | 92, 105, 3353                     |
| utility argument syntax           | 3384                              |
| utility conventions               | 3384                              |
| utility description defaults      | 3540                              |
| utility limits                    | 3537                              |
| utility option parsing            | 2706                              |
| Utility Syntax Guidelines         | 201                               |
| utility syntax guidelines         | 3386                              |
| utime                             | 3588                              |
| utime()                           | 2114, 3338                        |
| utimensat                         | 936                               |
| utimensat()                       | 2116                              |
| utimes                            | 936, 2116                         |
| UTIME_NOW                         | 372, 936                          |
| UTIME_OMIT                        | 372, 936                          |
| utim_                             | 451                               |
| utmpx                             | 433                               |
| uts_                              | 451                               |
| ut_                               | 451                               |
| UU                                | 12                                |
| uucp                              | 3193, 3592                        |
| uudecode                          | 3197, 3592                        |
| uuencode                          | 3200, 3592                        |
| uustat                            | 3205                              |
| uux                               | 3208                              |
| val                               | 3212                              |
| variable                          | 92                                |
| variable assignment               | 106, 3354                         |
| variables                         | 3549                              |
| Variable_List                     | 2331                              |
| va_arg()                          | 2117                              |
| va_copy                           | 2117                              |
| va_end                            | 2117                              |
| va_start                          | 2117                              |
| VDISCARD                          | 452                               |
| VDSUSP                            | 452                               |
| VEOF                              | 393, 3384                         |
| VEOL                              | 393, 3384                         |
| VERASE                            | 393                               |
| Version 7                         | 563, 637, 1164, 2098, 3351, 3546  |
| vertical-tab character            | 93                                |
| vfork                             | 3522                              |
| vfprintf()                        | 2118                              |
| VFS                               | 375                               |
| vfscanf()                         | 2119                              |
| vfwprintf()                       | 2120                              |
| vfwscanf()                        | 2121                              |
| vhangup()                         | 3332                              |
| vi                                | 3215, 3591-3592                   |
| <ESC>                             | 3253                              |
| append                            | 3235                              |
| change                            | 3236                              |
| change to end-of-line             | 3236                              |
| clear and redisplay               | 3223                              |
| command descriptions              | 3216                              |
| control-D                         | 3250                              |
| control-H                         | 3251                              |
| control-T                         | 3252                              |
| control-U                         | 3252                              |
| control-V                         | 3253                              |
| current and line above            | 3229                              |
| delete                            | 3237                              |
| delete character                  | 3246                              |
| delete to end-of-line             | 3237                              |
| display information               | 3222                              |

|                                       |                  |                                      |                                                 |
|---------------------------------------|------------------|--------------------------------------|-------------------------------------------------|
| edit the alternate file .....         | 3224             | shift right .....                    | 3234                                            |
| enter ex mode.....                    | 3243             | substitute character .....           | 3244                                            |
| execute.....                          | 3234             | substitute lines .....               | 3244                                            |
| execute an ex command .....           | 3233             | terminate command or input mode..... | 3224                                            |
| exit .....                            | 3248             | undo .....                           | 3245                                            |
| find character .....                  | 3238             | undo current line .....              | 3245                                            |
| find regular expression.....          | 3231             | yank .....                           | 3247                                            |
| Initialization .....                  | 3216             | yank current line.....               | 3247                                            |
| input mode commands.....              | 3248             | VINTR .....                          | 393                                             |
| insert.....                           | 3239             | virtual processor .....              | 3345                                            |
| insert empty line .....               | 3241             | VISIT .....                          | 2046, 2091                                      |
| join .....                            | 3240             | visual mode .....                    | 2573                                            |
| mark position.....                    | 3240             | VKILL.....                           | 393                                             |
| move back .....                       | 3229-3230, 3235  | VLNEXT.....                          | 452                                             |
| move cursor.....                      | 3222, 3225, 3244 | VMIN .....                           | 3384                                            |
| move down.....                        | 3222             | vprintf.....                         | 2118                                            |
| move forward.....                     | 3230             | vprintf() .....                      | <b>2122</b>                                     |
| move to bigword .....                 | 3238, 3245       | VQUIT .....                          | 393                                             |
| move to bottom of screen .....        | 3240             | VREPRINT.....                        | 452                                             |
| move to first character in line ..... | 3233             | vscanf .....                         | 2119                                            |
| move to first non-<blank> .....       | 3229             | vscanf() .....                       | <b>2123</b>                                     |
| move to line.....                     | 3239             | vsnprintf.....                       | 2118                                            |
| move to matching character.....       | 3226             | vsnprintf() .....                    | <b>2124</b>                                     |
| move to middle of screen .....        | 3240             | vsprintf .....                       | 2118, 2124                                      |
| move to next section .....            | 3228             | vsscanf .....                        | 2119                                            |
| move to specific column.....          | 3230             | vsscanf() .....                      | <b>2125</b>                                     |
| move to top of screen.....            | 3239             | VSTART.....                          | 393                                             |
| move to word .....                    | 3237, 3245       | VSTATUS .....                        | 452                                             |
| move up .....                         | 3223             | VSTOP .....                          | 393                                             |
| newline.....                          | 3251             | VSUSP .....                          | 393                                             |
| nul .....                             | 3250, 3253       | vswprintf .....                      | 2120                                            |
| page backwards .....                  | 3220             | vswprintf() .....                    | <b>2126</b>                                     |
| page forward .....                    | 3221             | vswscanf .....                       | 2121                                            |
| put from buffer .....                 | 3242             | vswscanf() .....                     | <b>2127</b>                                     |
| redraw screen.....                    | 3223             | VTDLY .....                          | 395                                             |
| redraw window .....                   | 3247             | VTIME.....                           | 3384                                            |
| regular expression .....              | 3234             | VTn .....                            | 395                                             |
| repeat.....                           | 3231             | VWERASE .....                        | 452                                             |
| repeat find .....                     | 3233, 3241       | vwprintf .....                       | 2120                                            |
| repeat substitution.....              | 3227             | vwprintf() .....                     | <b>2128</b>                                     |
| replace character .....               | 3243             | vwscanf .....                        | 2121                                            |
| replace text with command.....        | 3225             | vwscanf().....                       | <b>2129</b>                                     |
| return to previous context.....       | 3227             | wait .....                           | 2244, 2264, 3267, 3545, 3591                    |
| return to previous section .....      | 3228             | wait for process termination.....    | 2135                                            |
| reverse case.....                     | 3235             | wait for thread termination.....     | 1580                                            |
| reverse find character .....          | 3231             | wait() .....                         | <b>2130</b> , 3404, 3407, 3413-3414, 3416, 3587 |
| scroll backward.....                  | 3223             | waitid() .....                       | <b>2139</b> , 3414, 3587                        |
| scroll backward by line.....          | 3224             | waiting on a condition .....         | 1550                                            |
| scroll forward .....                  | 3221             | waitpid .....                        | 2130                                            |
| scroll forward by line .....          | 3221             | waitpid() .....                      | <b>2141</b> , 3330, 3334, 3414, 3518, 3587      |
| search for tagstring.....             | 3224             | wall, rationale for omission .....   | 3579                                            |
| shift left .....                      | 3233             | WARNING .....                        | 845                                             |

## Index

|                                       |                                                                                                                            |  |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------|--|
| warning                               |                                                                                                                            |  |
| OB                                    | .....9                                                                                                                     |  |
| OF                                    | .....9                                                                                                                     |  |
| wc                                    | .....3271, 3592                                                                                                            |  |
| WCHAR_MAX                             | .....331, 435                                                                                                              |  |
| WCHAR_MIN                             | .....331, 435                                                                                                              |  |
| WCONTINUED                            | .....387, 2130, 2139                                                                                                       |  |
| wcpcpy                                | .....2153                                                                                                                  |  |
| wcpcpy()                              | .....2142                                                                                                                  |  |
| wcpncpy                               | .....2162                                                                                                                  |  |
| wcpncpy()                             | .....2143                                                                                                                  |  |
| wrtomb()                              | .....2144                                                                                                                  |  |
| wscasecmp                             | .....2146                                                                                                                  |  |
| wscasecmp_l                           | .....2146                                                                                                                  |  |
| wscat()                               | .....2148                                                                                                                  |  |
| wchr()                                | .....2149                                                                                                                  |  |
| wscmp()                               | .....2150                                                                                                                  |  |
| wscoll()                              | .....2151                                                                                                                  |  |
| wscoll_l                              | .....2151                                                                                                                  |  |
| wscopy()                              | .....2153                                                                                                                  |  |
| wscspn()                              | .....2154                                                                                                                  |  |
| wsdup()                               | .....2155                                                                                                                  |  |
| wcsftime()                            | .....2156                                                                                                                  |  |
| wcslen()                              | .....2158                                                                                                                  |  |
| wscnscasecmp                          | .....2146                                                                                                                  |  |
| wscnscasecmp()                        | .....2159                                                                                                                  |  |
| wscnscasecmp_l                        | .....2146, 2159                                                                                                            |  |
| wscncat()                             | .....2160                                                                                                                  |  |
| wscncmp()                             | .....2161                                                                                                                  |  |
| wscncpy()                             | .....2162                                                                                                                  |  |
| wcsnlen                               | .....2158                                                                                                                  |  |
| wcsnlen()                             | .....2164                                                                                                                  |  |
| wcsnrtombs                            | .....2168                                                                                                                  |  |
| wcsnrtombs()                          | .....2165                                                                                                                  |  |
| wcspbrk()                             | .....2166                                                                                                                  |  |
| wcsrchr()                             | .....2167                                                                                                                  |  |
| wcsrtombs()                           | .....2168                                                                                                                  |  |
| wcsspn()                              | .....2170                                                                                                                  |  |
| wcsstr()                              | .....2171                                                                                                                  |  |
| wstod()                               | .....2172                                                                                                                  |  |
| wstof                                 | .....2172                                                                                                                  |  |
| wstoimax()                            | .....2176                                                                                                                  |  |
| wstok()                               | .....2178                                                                                                                  |  |
| wstol()                               | .....2180                                                                                                                  |  |
| wstold                                | .....2172                                                                                                                  |  |
| wstold()                              | .....2183                                                                                                                  |  |
| wstoll                                | .....2180                                                                                                                  |  |
| wstoll()                              | .....2184                                                                                                                  |  |
| wstombs()                             | .....2185                                                                                                                  |  |
| wstoul()                              | .....2187                                                                                                                  |  |
| wstoull                               | .....2187                                                                                                                  |  |
| wstoumax                              | .....2176                                                                                                                  |  |
| wcstoumax()                           | .....2190                                                                                                                  |  |
| wcswcs                                | .....3522                                                                                                                  |  |
| wcswidth()                            | .....2191                                                                                                                  |  |
| wcsxfrm()                             | .....2192                                                                                                                  |  |
| wcsxfrm_l                             | .....2192                                                                                                                  |  |
| wctob()                               | .....2194                                                                                                                  |  |
| wctomb()                              | .....2195                                                                                                                  |  |
| wctrans()                             | .....2197                                                                                                                  |  |
| wctrans_l                             | .....2197                                                                                                                  |  |
| wctype()                              | .....2199                                                                                                                  |  |
| wctype_l                              | .....2199                                                                                                                  |  |
| wcwidth()                             | .....2201                                                                                                                  |  |
| WEOF                                  | .....435, 439, 519, 1133, 1135, 1138, 1140<br>.....1142, 1144, 1146, 1148, 1150, 1152, 1154<br>.....1156, 2080, 2082, 2102 |  |
| WERASE                                | .....3382                                                                                                                  |  |
| WEXITED                               | .....387, 2139                                                                                                             |  |
| WEXITSTATUS                           | .....338, 387, 2131                                                                                                        |  |
| we_                                   | .....451                                                                                                                   |  |
| what                                  | .....3274                                                                                                                  |  |
| while loop                            | .....2270, 3565                                                                                                            |  |
| white space                           | .....93                                                                                                                    |  |
| who                                   | .....3276, 3592-3593                                                                                                       |  |
| wide characters                       | .....115                                                                                                                   |  |
| wide-character code (C language)      | .....93                                                                                                                    |  |
| wide-character codes                  | .....3355                                                                                                                  |  |
| wide-character input/output functions | .....93                                                                                                                    |  |
| wide-character string                 | .....93                                                                                                                    |  |
| wide-oriented stream                  | .....472                                                                                                                   |  |
| WIFCONTINUED                          | .....387, 2131                                                                                                             |  |
| WIFEXITED                             | .....338, 387, 2131                                                                                                        |  |
| WIFSIGNALED                           | .....338, 387, 2131                                                                                                        |  |
| WIFSTOPPED                            | .....338, 387, 2131, 2136                                                                                                  |  |
| WINT_MAX                              | .....331                                                                                                                   |  |
| WINT_MIN                              | .....331                                                                                                                   |  |
| wmemchr()                             | .....2202                                                                                                                  |  |
| wmemcmp()                             | .....2203                                                                                                                  |  |
| wmemcpy()                             | .....2204                                                                                                                  |  |
| wmemmove()                            | .....2205                                                                                                                  |  |
| wmemset()                             | .....2206                                                                                                                  |  |
| WNOHANG                               | .....338, 387, 1875, 2130, 2139                                                                                            |  |
| WNOWAIT                               | .....387, 2139                                                                                                             |  |
| word                                  | .....94                                                                                                                    |  |
| word counting                         | .....3271                                                                                                                  |  |
| word expansions                       | .....2253, 3552                                                                                                            |  |
| wordexp()                             | .....2207, 3593                                                                                                            |  |
| wordfree                              | .....2207                                                                                                                  |  |
| wordfree()                            | .....3593                                                                                                                  |  |
| WORD_BIT                              | .....262, 264                                                                                                              |  |
| working directory                     | .....94                                                                                                                    |  |
| worldwide portability interface       | .....94                                                                                                                    |  |
| wprintf                               | .....940                                                                                                                   |  |
| wprintf()                             | .....2211                                                                                                                  |  |

|                                        |                                                                                 |
|----------------------------------------|---------------------------------------------------------------------------------|
| WRDE_.....                             | 451                                                                             |
| WRDE_APPEND.....                       | 441, 2208                                                                       |
| WRDE_BADCHAR.....                      | 441, 2209                                                                       |
| WRDE_BADVAL.....                       | 441, 2209                                                                       |
| WRDE_CMDSUB.....                       | 441, 2209                                                                       |
| WRDE_DOOFFS.....                       | 441, 2208                                                                       |
| WRDE_NOCMD.....                        | 441, 2208                                                                       |
| WRDE_NOSPACE.....                      | 441, 2209                                                                       |
| WRDE_REUSE.....                        | 441, 2208                                                                       |
| WRDE_SHOWERR.....                      | 441, 2208                                                                       |
| WRDE_SYNTAX.....                       | 441, 2209                                                                       |
| WRDE_UNDEF.....                        | 441, 2208                                                                       |
| write.....                             | 94, 3280, 3588, 3591-3592                                                       |
| write lock.....                        | 3474                                                                            |
| write to a file.....                   | 2216                                                                            |
| write().....                           | 2212, 3329-3330, 3381, 3404<br>.....3413-3414, 3416, 3425-3427, 3431-3432, 3519 |
| writew().....                          | 2220                                                                            |
| writing data.....                      | 3383                                                                            |
| wscanf.....                            | 949                                                                             |
| wscanf().....                          | 2222                                                                            |
| WSTOPPED.....                          | 387, 2139                                                                       |
| WSTOPSIG.....                          | 338, 387, 2131                                                                  |
| WTERMSIG.....                          | 338, 387, 2131                                                                  |
| WUNTRACED.....                         | 338, 387, 2130, 2136, 3330                                                      |
| W_OK.....                              | 419                                                                             |
| xargs.....                             | 3283, 3591                                                                      |
| XOPEN_UNIX.....                        | 19, 28                                                                          |
| XOPEN_UUCP.....                        | 19, 28                                                                          |
| XSI.....                               | 12, 94, 3345                                                                    |
| conformance.....                       | 15                                                                              |
| XSI conformance.....                   | 19, 94                                                                          |
| XSI interprocess communication.....    | 474                                                                             |
| XSI IPC.....                           | 3417                                                                            |
| XSI options groups.....                | 21                                                                              |
| XSI STREAMS.....                       | 25                                                                              |
| XSI system interfaces                  |                                                                                 |
| conformance.....                       | 19                                                                              |
| XSI_C_LANG_SUPPORT.....                | 3611                                                                            |
| XSI_DBM.....                           | 3612                                                                            |
| XSI_DEVICE_IO.....                     | 3612                                                                            |
| XSI_DEVICE_SPECIFIC.....               | 3612                                                                            |
| XSI_FILE_SYSTEM.....                   | 3612                                                                            |
| XSI_IPC.....                           | 3612                                                                            |
| XSI_JUMP.....                          | 3612                                                                            |
| XSI_MATH.....                          | 3612                                                                            |
| XSI_MULTI_PROCESS.....                 | 3612                                                                            |
| XSI_SIGNALS.....                       | 3612                                                                            |
| XSI_SINGLE_PROCESS.....                | 3612                                                                            |
| XSI_SYSTEM_DATABASE.....               | 3612                                                                            |
| XSI_SYSTEM_LOGGING.....                | 3612                                                                            |
| XSI_THREADS_EXT.....                   | 3612                                                                            |
| XSI_TIMERS.....                        | 3612                                                                            |
| XSI_USER_GROUPS.....                   | 3612                                                                            |
| XSI_WIDE_CHAR.....                     | 3612                                                                            |
| XSR.....                               | 13                                                                              |
| X_OK.....                              | 419, 541                                                                        |
| y0().....                              | 2223                                                                            |
| y1.....                                | 2223                                                                            |
| yacc.....                              | 3290, 3593-3594                                                                 |
| algorithms.....                        | 3301                                                                            |
| code file.....                         | 3292                                                                            |
| completing the program.....            | 3300                                                                            |
| conflicts.....                         | 3298                                                                            |
| debugging the parser.....              | 3301                                                                            |
| declarations section.....              | 3293                                                                            |
| description file.....                  | 3292                                                                            |
| error handling.....                    | 3299                                                                            |
| grammar rules.....                     | 3295                                                                            |
| header file.....                       | 3292                                                                            |
| input grammar.....                     | 3296                                                                            |
| input language.....                    | 3292                                                                            |
| interface to the lexical analyzer..... | 3300                                                                            |
| lexical structure of the grammar.....  | 3293                                                                            |
| library.....                           | 3300                                                                            |
| limits.....                            | 3301                                                                            |
| programs section.....                  | 3296                                                                            |
| YESEXPR.....                           | 250                                                                             |
| YESSTR.....                            | 250                                                                             |
| yn.....                                | 2223                                                                            |
| zcat.....                              | 3306                                                                            |
| zombie process.....                    | 94, 523                                                                         |